به نام خدا

مؤسسه فرهنگی هنری
دیباگران تهران

برنامه‌نویسی VBA در Excel

مترجم
جواد قنبر

RWTUV
TÜV CERT
EN ISO 9001
دارنده گواهینامه ISO 9001/2000
در زمینه نشر کتاب و طراحی جلد

Excel VBA Macro Programming

برنامه‌نویسی VBA در Excel

مترجم : جواد قنبر

ناشر : مؤسسه فرهنگی هنری دیباگران تهران

حروفچینی و صفحه‌آرایی : مجتمع فنی تهران

طرح روی جلد : مجتمع فنی تهران

چاپ : ایران مصور

نوبت چاپ : اول

تاریخ نشر : شهریور ماه ۱۳۸۵

تیراژ : ۳۰۰۰ نسخه

قیمت : ۳۶۰۰۰ ریال

شابک : ۱- ۷۲۲-۳۵۴-۹۶۴

ISBN : 964-354-722-1

شپرد، ریچارد Shepherd, Richard

برنامه‌نویسی VBA [وی. بی. ای] در Excel [اکسل]/[ریچارد شپرد]؛ مترجم جواد قنبر.—تهران: مؤسسه فرهنگی هنری دیباگران تهران، ۱۳۸۵.

۳۷۲ ص.: مصور، جدول.

ISBN 964-354-722-1

فهرست‌نویسی بر اساس اطلاعات فیپا.

عنوان اصلی : Excel VBA Macro Programming. c2004.

۱. اکسل مایکروسافت (فایل کامپیوتر) ۲. ویژوال بیسیک مایکروسافت برای برنامه‌های کاربردی. ۳. برنامه‌نویسی ماکرو. ۴. بازرگانی— برنامه‌های کامپیوتری. ۵.نرم‌افزار— تولید. ۶. صفحه گسترده الکترونیکی. الف. قنبر، جواد، ۱۳۵۷- مترجم . ب. عنوان.

HF۵۵۴۸/۴/الف۷ش۲۴ ۰۰۵/۲۶۸

۱۳۸۵

کتابخانه ملی ایران ۸۵-۱۳۵۵۶م

آدرس: سعادت آباد، میدان کاج . خ سرو شرقی. روبه‌روی خ علامه . ساختمان شماره ۹۷

تلفن : ۷- ۲۲۰۹۸۴۴۶ صندوق پستی: ۹۴۳/ ۱۴۳۳۵

فهرست مطالب

بخش دوم : مدل‌های شئ

مقدمه ناشر

حمد و سپاس ایزد منان را که با الطاف بیکران خود این توفیق را به ما ارزانی داشت تا بتوانیم در راه ارتقای دانش عمومی و فرهنگ این مرز و بوم در زمینه چاپ و نشر کتب علمی دانشگاهی، علوم پایه و به ویژه علوم کامپیوتر و انفورماتیک گام‌هایی هر چند کوچک برداشته و در انجام رسالتی که بر عهده داریم مؤثر واقع شویم. گستردگی علوم و توسعه روزافزون آن، شرایطی را به وجود آورده که هر روز شاهد تحولات اساسی چشمگیری درسطح جهان هستیم. این گسترش و توسعه نیازبه منابع مختلف از جمله کتاب را به عنوان قدیمی‌ترین و راحت‌ترین راه دستیابی به اطلاعات و اطلاع‌رسانی، بیش از پیش روشن می‌کند. در این راستا، واحد انتشارات مؤسسه فرهنگی هنری دیباگران تهران با همکاری جمعی از اساتید، مؤلفان، مترجمان، متخصصان، پژوهشگران، محققان و نیز پرسنل ورزیده و ماهر در زمینه امور نشر درصدد هستند تا با تلاش‌های مستمر خود برای رفع کمبودها و نیازهای موجود، منابعی پربار، معتبر، و با کیفیت مناسب در اختیار علاقه‌مندان قرار دهند.

کتابی که در دست دارید با همت **"آقای جواد قنبر"** و تلاش جمعی از همکاران انتشارات میسر گشته که شایسته است از یکایک این گرامیان تشکر و قدردانی کنیم.

ویراستاری: شیوا غمگسار

ویرایش و صفحه‌آرایی کامپیوتری: مریم فرجیان

طرح جلد: مریم بیک‌زاده

امور چاپ و نشر: حیدر شفیعی

ناظر چاپ : کریم براغ

در خاتمه از خوانندگان عزیز و دانش پژوهان گرامی خواهشمندیم ما را با ارایه پیشنهادها و انتقادهای خود در بهبود کمی و کیفی کارهای انجام شده راهنمایی نمایند تا بتوانیم در آینده کتاب‌هایی با کیفیت بهتر تقدیم حضورشان کنیم.

مدیر انتشارات

مؤسسه فرهنگی هنری دیباگران تهران

Publishing@mftmail.com

مقدمه مترجم

« تقدیم به همسرم که دوست داشتن و عاشقانه زیستن را از او آموختم»

با توجه به قابلیت‌های زبان برنامه‌نویسی Visual Basic ، شرکت مایکروسافت از این نرم‌افزار برای بالا بردن توان مجموعه برنامه‌های Office استفاده کرده است. درتمام نرم‌افزارهای این مجموعه (Word، Excel و ...) ، امکان برنامه‌نویسی وجود دارد که خود باعث تحول شگرفی در خروجی این نرم‌افزارها شده است و آن‌ها را برای کاربران ، به‌صورت ابزاری فوق حرفه‌ای در آورده است.

در این کتاب چگونگی استفاده از VBA برای بالا بردن قابلیت‌های نرم‌افزار Excel را شرح می‌دهیم. هر چند مطالب به نحوی تنظیم شده‌اند که خوانندگان حتی بدون شناخت قبلی از برنامه Excel و زبان برنامه‌نویسی VB به مشکلی بر نمی‌خورند اما پیشنهاد می‌شود ابتدا Excel و VB را به‌صورت مقدماتی ودر حد آشنایی بیاموزند.

در هفت فصل نخست، بیشتر بر آموزش VB تأکید شده است و در ادامه تا فصل ۱۸ به بحث‌های تخصصی‌تر VB پرداخته‌ایم. از این فصل تا انتهای کتاب نیز به صورت عملی از VB در نرم‌افزار Excel استفاده کرده‌ایم.

بسیار خوشحال می‌شویم که نظرات و پیشنهادات خود را برای برطرف کردن اشکال‌ها و نقاط ضعف از طریق ایمیل زیر به اطلاع ما برسانید:

Birang 1357@yahoo.com

آسمان باشید؛ آبی و بلند

جواد قنبر

بخش اول

برنامه‌نویسی در VBA

در این بخش، همه چیز را درباره چگونگی عملکرد VBA (ویژوال بیسیک برای برنامه‌های کاربردی) خواهید آموخت. قوانین کدگذاری، چگونگی کدنویسی، چگونگی اشکال‌زدایی از کد (جستجوی خطاهای غیر قابل اجتناب) و چگونگی ساخت GUIها (رابط‌های گرافیکی کاربر) را که برای سادگی اجرای کد توسط کاربران مفید هستند، فرا خواهید گرفت.

فصل اول

مقدمات

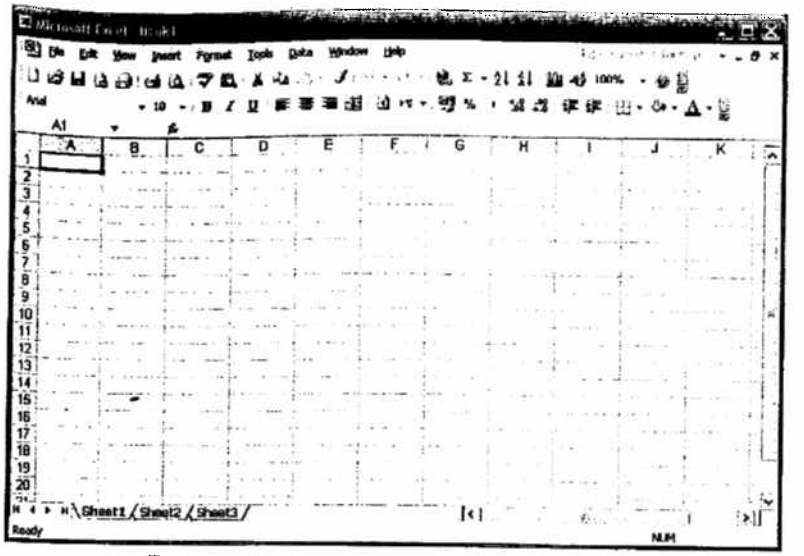
هدف این فصل، آموزش اصول اولیه به‌کارگیری پنجره Visual Basic Editor یا VBE و نوشتن یک کد ساده VBA است. در این فصل به شما نشان داده خواهد شد که چگونه از پنجره‌های کد نویسی Visual Basic Editor و Project Explorer استفاده کنید. هم‌چنین چگونگی نوشتن یک ماکروی ساده برای نمایش یک کادر پیغام "Hello World" را فرا خواهید گرفت.

بررسی Visual Basic Editor در Excel

اگر از Excel زیاد استفاده کرده باشید، با لایه‌بندی صفحه گسترده در آن آشنا هستید. وقتی Excel را باز می‌کنید، یک نمای استاندارد مثل تصویر ۱-۱ را خواهید دید. در این تصویر، یک نوار منو و نوارهای ابزار در قسمت بالای صفحه گسترده و زبانه‌هایی در قسمت پایین وجود دارند که به شما اجازه می‌دهند به کاربرگ‌های مختلف(Worksheets) دست پیدا کنید. می‌توانید در سلول‌ها، داده‌ها و فرمول‌ها را درج کنید، سلول‌ها یا تمام کاربرگ را قالب‌بندی کرده و تصاویر گرافیکی و نمودارها را نیز درج کنید. حتی می‌توانید با استفاده از Tools | Macro | Record New Macro، یک ماکرو را ضبط کنید.

بیشتر کاربران نمی‌دانند که علاوه بر برنامه کاربردی صفحه گسترده Excel، یک زبان برنامه‌نویسی فوق‌العاده قوی در Excel در نظر گرفته شده که می‌توانند از آن برای طراحی برنامه‌های کاربردی دلخواه خود نیز استفاده کنند. می‌توانید از کد نویسی VBA برای نوشتن برنامه‌های کاربردی ماکرو که کارهای بسیار جالبی انجام می‌دهد، استفاده کنید.

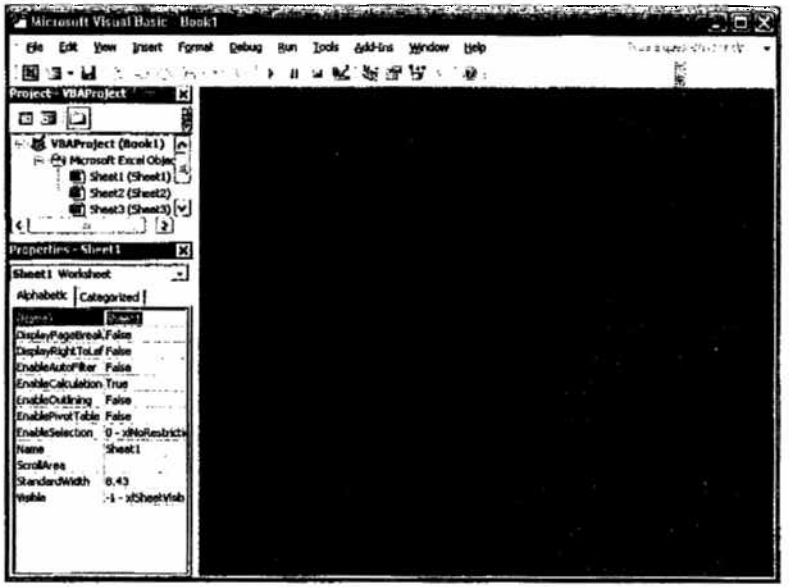
ماکرو، یک روال نوشته شده (Procedure) در کد VBA است که عملیات‌های خاصی انجام می‌دهد. این عملیات می‌تواند چیزی مثل مرتب کردن تمام کاربرگ‌ها (Worksheet) در یک کارپوشه (Workbook) و برحسب ترتیب الفبا یا اضافه کردن ساختارهای منوی جدید به منوی Excel باشد. به هر حال، ماکروها اقدام به خودکار کردن عملیات‌ها می‌کنند و انجام امور را برای شما و کاربران کارپوشه شما راحت تر می‌کند.



تصویر ۱-۱ صفحه نمایش استاندارد صفحه گسترده Excel

نکته: کارپوشه یا Workbook، فایلی در برنامه صفحه گسترده است که حاوی تعدادی کاربرگ مرتبط به هم است.

با این وجود ، پیش از آن‌که برنامه‌نویسی در Excel را آغاز کنید، لازم است بدانید ماکروها در کجا ذخیره می‌شوند. در زبان قدیمی ماکرونویسی، تنها یک برگه ماکرو را درج کرده و در هر جای آن که می‌خواستیم دستورها را وارد می‌کردیم. حال که زبان ماکرونویسی گسترش یافته و تبدیل به یک زبان برنامه‌نویسی کامل شئ‌گرا شده، شیوه ذخیره سازی آن هم تغییر کرده است. ماکروها در حال حاضر در VBA Project های پنهان که در کارپوشه نگهداری و ذخیره می‌شوند، نگهداری می‌شوند. به این VBA Projectها می‌توان از طریق یک برنامه دیگر که VBE (ویراستار ویژوال بیسیک) نامیده می‌شود، دست پیدا کرد. برای دیدن پنجره‌ای که در تصویر ۲-۱ نشان داده شده است، کلید ترکیبی ALT+F11 را فشار دهید.



تصویر ۲-۱ پنجره استاندارد Visual Basic

در نگاه اول، این پنجره با نوار منوی جدید خود که حاوی منوهایی برای File, Edit, View, Insert, Format, Debug, Run, Tools, Window و Help است، ممکن است گیج‌کننده به نظر برسد. این پنجره به صورت یک پنجره مجزا باز می‌شود اما هنوزهم تا حد زیادی، بخشی از برنامه Excel به حساب می‌آید.

پنجره‌های VBA Project Explorer و Code

Project Explorer که ساختار درختی Project را نشان می‌دهد، در گوشه سمت چپ - بالای صفحه نمایش قرار دارد و پروژه VBA را برای کارپوشه فعال نشان می‌دهد و جزییات را به صورت ساختار درختی نمایش می‌دهد تا بتوانیم به سادگی بین آن‌ها حرکت کنیم. اگر روی یکی از شاخه‌های ساختار درختی کلیک کنیم، از VBA وارد آن کارپوشه یا کاربرگ خاص خواهیم شد. پروژه VBA، ریشه ساختار درختی است و کارپوشه و کاربرگ‌ها، شاخه‌هایی هستند که از ساختار درختی بیرون می‌آیند.

به موازات آن‌که کارپوشه یا کاربرگ‌ها را اضافه یا حذف می‌کنیم، شاخه‌های ساختار درختی نیز تغییر می‌کنند تا کاربرگ‌ها یا کارپوشه‌هایی را که جدید ایجاد شده‌اند، نشان دهند. در واقع آن‌چه می‌بینید لیستی از کتاب‌های کاری و کاربرگ‌هایی است که در حال حاضر در رابطی شبیه Explorer، بارگذاری شده‌اند.

به یاد داشته باشید که VBA، یک زبان برنامه نویسی شیءگراست. نخستین شاخه‌ای که از ریشه ساختار درختی پروژه VBA بیرون می‌آید،Microsoft Excel Objects را مطرح می‌کنند. آن‌چه از این شاخه منشعب می‌شود،اشیایی برای کارپوشه است که حاوی کاربرگ‌ها هستند. اشیای دیگری مانند User Forms، ماژول‌ها و ماژول‌های کلاس (Class module) را نیز می‌توانیم در ساختار درختی Project درج کنیم.

درک این قضیه بسیار مهم است که چون کارپوشه، شیئی است که می‌توانیم به آن ارجاع کنیم، هر کار برگ نیز یک شئ است که می‌توانیم به آن ارجاع کنیم. این‌ها ، تنها اشیا موجود در Excel نیستند اما اگر به سادگی به Project Explorer نگاه کنیم، این اشیا در آن‌جا نشان داده شده‌اند.

با دابل کلیک روی This Work Book، پنجره کد نویسی برای شئ Workbook باز می‌شود. در ابتدا چیز زیادی در آن وجود ندارد و ممکن است متوجه نشوید که باید چه کاری انجام دهید. اگر به صورت تصادفی، چیزی را تایپ کنیم (مثلا "What do I do now?") و ENTER را فشار دهیم، یک خطای ترجمه (Compile error) دریافت می‌کنیم که به خاطر قوانین و شرایط خاص کدنویسی است. هر چیزی که این‌جا تایپ کنیم به کامپایلر ویژوال بیسیک ارسال می‌شود که آن‌چه را نوشته‌ایم تفسیر کرده و آن‌را به دستورالعمل‌هایی تبدیل می‌کند که کامپیوتر آن‌ها را درک کند.

در کادر پیغام Compile Error روی OK کلیک کرده و دستور خود را پاک کنید. توجه کنید که خط دستور در هنگام بروز خطای کامپایل به رنگ قرمز درمی‌آید تا توجه شما را به قسمت مشکل‌دار جلب کند. حتی اگر کاری روی آن خط انجام ندهید باز هم به رنگ قرمز باقی می‌ماند تا به عنوان یک اعلام خطر به شما یاد آوری کند که در کدتان یک مشکل وجود دارد.

لیست بازشو در گوشه بالا- سمت چپ پنجره، General را نشان می‌دهد. روی آن کلیک کنید تا گزینه انتخابی دیگر یعنی Workbook را ببینید. روی Workbook کلیک کنید تا کد رویداد Open کارپوشه را مشاهده کنید. حالا صفحه نمایش شما شبیه تصویر ۳-۱ خواهد بود.

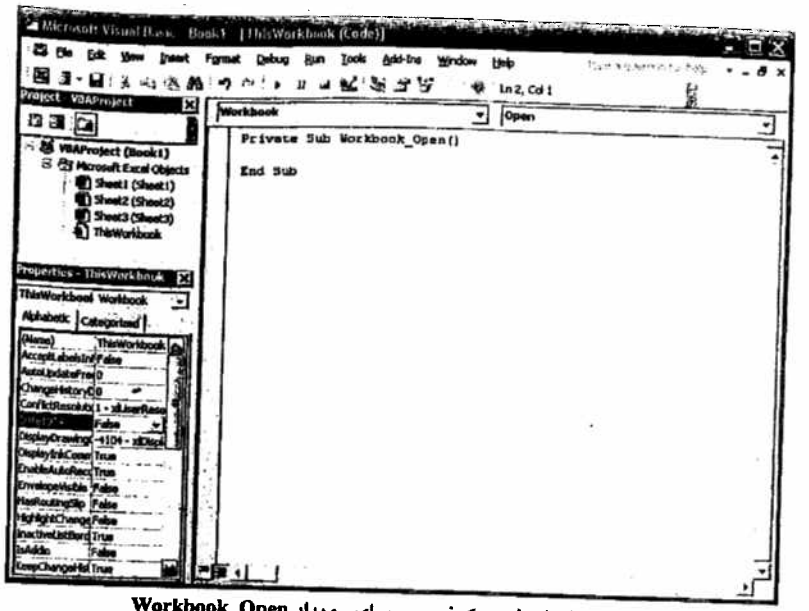
این‌ها چه معنایی می‌دهند؟ خیلی ساده است، وقتی این کارپوشه خاص را باز می‌کنید رویداد Workbook_Open اتفاق می‌افتد. این امر زمانی رخ می‌دهد که File | Open را انتخاب کرده سپس فایلی را انتخاب و آن‌را بارگذاری می‌کنیم. پنجره کد نویسی به طور خودکار دستورات Private Sub Workbook_Open() و End Sub را نشان می‌دهد:


```
Private Sub Workbook_Open( )

End Sub
```

دستورات فوق،منطقه کدنویسی برای نوشتن کد VBA برای رویداد Workbook_Open را در اختیار شما قرار می‌دهد. اگر روی لیست بازشو کلیک کنید متوجه می‌شوید رویدادهای دیگر هم وجود دارند که می‌توانید برای آن‌ها هم کدنویسی کنید. اما در حال حاضر تنها روی Workbook_Openمتمرکز می‌شویم.

در VBA، اگر از این رویداد استفاده نکنید، به این کد نیازی نخواهید داشت. هر بار که در لیست بازشو روی یک رویداد کلیک می‌کنید، دو دستور فوق به طور خودکار درج می‌شوند. هر کدی که برای رویدادها نوشته می‌شود باید در بین این دو دستور قرار گیرد در غیر این صورت، یک خطای کامپایل (ترجمه)صادر می‌شود. آن‌ها مثل خط شروع و پایان یک مسابقه به حساب می‌آیند و به کامپایلر اعلام می‌کنند که کد از کجا شروع می‌شود و در کجا خاتمه می‌یابد. اگر نخواهید برای آن رویداد، کدی بنویسید می‌توانید آن‌ها را پاک کنید و البته در صورتی که هر دوی آن‌ها را پاک نکنید یک پیغام اخطار کامپایل دریافت می‌کنید. کدی که نوشته می‌شود نیز باید به درستی ساختاربندی شده باشد.

در گوشه سمت راست از لیست بازشو، روی Open کلیک کنید (کادری که یک پیکان رو به پایین در سمت راست آن قرار دارد) تا لیستی از رویدادها را برای کارپوشه که می‌توانید برای آن کدنویسی کنید، ببینید. هر کدام از این رویدادها در یک زمان خاص و متناسب با کاربرد خاص کارپوشه فراخوانده شده و کد مرتبط با آن‌ها اجرا می‌شود. یک مثال، رویداد Workbook_Open است. تمام کدی که برای این رویداد وارد می‌کنید در هنگام باز شدن کارپوشه اجرا خواهد شد و اگر یک خطا وجود داشته باشد، پیغام خطای مربوطه نمایش داده خواهد شد.



تصویر ۳-۱ پنجره کدنویسی برای رویداد Workbook_Open

در حال حاضر، یک رویداد را شروع وتمام کرده‌اید. اگر چه بین دستورات **Sub** و **End Sub** چیزی وجود ندارد اما روال کار تغییری نمی‌کند و هر بار که کارپوشه باز می‌شود، **این دستورها اجرا می‌شوند.** البته چون هیچ کدی در رویداد وجود ندارد، کاری هم انجام نمی‌شود.

اولین ماکروی VBA در Excel

کتاب‌های برنامه‌نویسی به صورت سنتی با نوشتن یک کد ساده **برای نمایش پیغام "Hello World"** به شما کمک می‌کنند تا نخستین گام را در برنامه نویسی بردارید **و این کتاب نیز از این قاعده** مستثنی نیست. ما از دستور MsgBox برای نمایش این عبارت استفاده می‌کنیم. این دستور یک رابط ساده کاربر است که عبارت را همراه با یک دکمه OK نمایش می‌دهد (قبلاً در ویندوز دیده‌اید).

برای این مثال از رویداد دیگری برای شئ Workbook استفاده خواهیم کرد. شما می‌توانید از شئ فعلی Workbook_Open استفاده کنید اما این‌کار مستلزم این است که کارپوشه را بسته و مجدداً باز کنید، چون این رویداد تنها موقع باز بودن کارپوشه اجرا می‌شود. در عوض ما از رویداد NewSheet استفاده خواهیم کرد که هر بار یک کاربرگ جدید را در کتاب کاری درج کنیم، اجرا می‌شود.

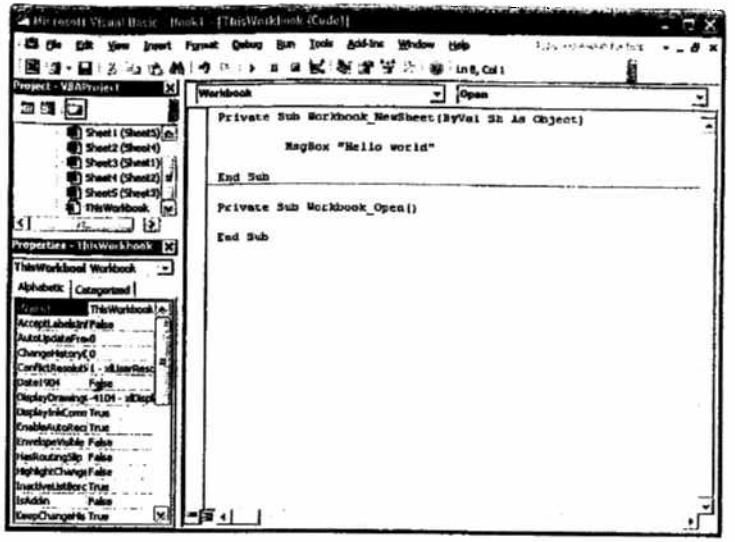
از منوی بازشو(کادر گوشه بالایی- سمت راست که یک پیکان رو به پایین بر روی آن قرار دارد)، رویداد NewSheet را انتخاب کنید. حال پنجره شما دارای دو دستور اضافی خواهد بود که به رویداد Workbook_ NewSheet اضافه شده است. مثل رویداد Workbook_Open، این‌ها نیز شبیه یک خط شروع و پایان هستند.

درزیر Private Sub Workbook_NewSheet(ByVal Sh As Object) و قبل از End Sub،عبارت msgbox "Hello World" را تایپ کنید. در هنگام تایپ "msgbox" از کلید SHIFT استفاده نکنید و ببینید چه اتفاقی می‌افتد. با توجه به این‌که msgbox یکی از کلمات دستوری تعریف شده در VBA است، پس از فشردن ENTER، به صورت "MsgBox" در می‌آید. البته باید به املای آن دقت کنید چون در صورت غلط تایپ کردن،یک خطای کامپایل رخ خواهد داد:

```
Private Sub Workbook_NewSheet(ByVal Sh As Object)
   MsgBox "Hello World"
End Sub
```

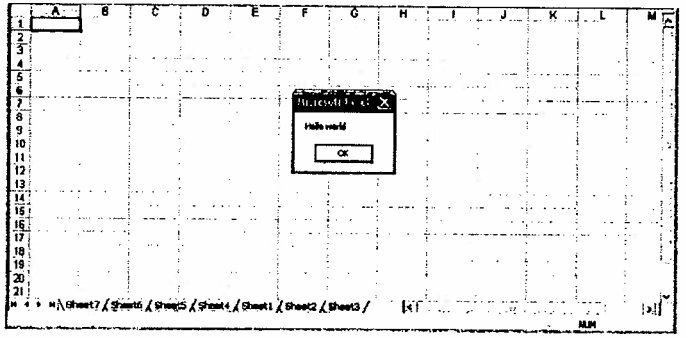
توجه کنید که دستور MsgBox را با فشردن کلید TAB، کمی تورفته‌تر از دیگر قسمت‌ها تایپ کرده‌ایم. این شیوه کمک می‌کند تا شروع و پایان یک مجموعه از دستورات را به راحتی دنبال کنیم. دستور MsgBox، یک روش ساده برای برقراری ارتباط با کاربر توسط نمایش یک پیغام و یک دکمه OK است که حتماً شما هم تا به حال، نمونه‌هایی از آن‌ها را دیده‌اید. البته در فصل ۵ با نمونه‌های پیچیده‌تر و جزییات بیشتری درباره آن‌ها آشنا خواهید شد.

پس از تایپ کلمه msgbox ،کادری نمایش داده می‌شود که حاوی تمام پارامترهای این دستور است. در این تمرین چون تنها یک رشته متنی را نمایش می‌دهیم توجهی به این کادر نداریم. با وجود این دانستن این‌که چه پارامترهایی برای یک تابع مورد نیاز هستند و این‌که به چه صورتی ظاهر می‌شوند سودمند است. برای مثال اگر می خواستیم یک عنوان (title) یا یک آیکن خاص را برای کادر پیغام تعیین کنیم، کادر پارامترها به ما کمک می‌کرد تا این‌کار را به درستی انجام دهیم. کادر پارامترها، یک لیست است که وقتی به پارامتر مربوط به آن آیکن می‌رسیم ظاهر می‌شود و لیستی از گزینه‌های قابل انتخاب برای آیکن را نمایش می‌دهد. در هنگام استفاده از ویراستار VBA می‌توانیم از این نوع راهنمایی برای تمام توابع استفاده کنیم. در حال حاضر، پنجره کد شما باید شبیه تصویر ۴-۱ باشد.



تصویر ۴-۱

برای اجرای این ماکروی رویدادگرا و نمایش پیغام، به پنجره Excel برگردید و با کلیک روی آیکن Excel در نوار وظیفه Windows در پایین صفحه نمایش و یا با کلیک روی دکمه View Microsoft Excel واقع در نوار ابزار VBE (نخستین دکمه)، یک کاربرگ جدید درج کنید. سپس Insert | Worksheet را انتخاب کنید. همان‌طورکه در تصویر ۵-۱ می‌بینید،کادر پیغام Hello World همراه با دکمه OK روی آن ظاهر می‌شود.



تصویر ۵-۱

این تمرین، توضیح ساده اضافه کردن کد به یک رویداد است و اگر قرار بود هر بار که‌یک برگ جدید به کارپوشه اضافه می‌شود، پیغام "Hello World" نمایش داده شود، حتماً کاربران عصبانی می‌شدند.

رویدادها زمانی فعال می‌شوند که تغییرات مربوط به آن‌ها روی صفحه گسترده Excel رخ دهد. می‌توانید کدی بنویسید که در هنگام وقوع یک رویداد خاص، مانند تغییر دادن صفحه گسترده توسط کاربر، فعال شود و هر بار که آن رویداد رخ دهد، کد شما اجرا خواهد شد.

تا زمان کلیک OK برای خاتمه اجرای ماکرو، نمی‌توانید کاری را در صفحه گسترده انجام دهید. این امر به خاطر آن است که تمرکز کد روی پنجره کادر پیغام شماست و تا زمانی‌که کادر پیغام بسته نشود نمی‌توانید به جای دیگری از صفحه Excel بروید.

برای این‌که کادر پیغام با هر بار بارگذاری یک صفحه جدید، ظاهر نشود باید خط MsgBox "Hello World" را با فشار کلید DELETE پاک کنید. روش دیگر جلوگیری از اجرای یک کد این است که با قرار دادن یک کاراکتر نقل قول (') در جلوی کد، آن‌را تبدیل به یک عبارت توضیحی (Comment) کنید:

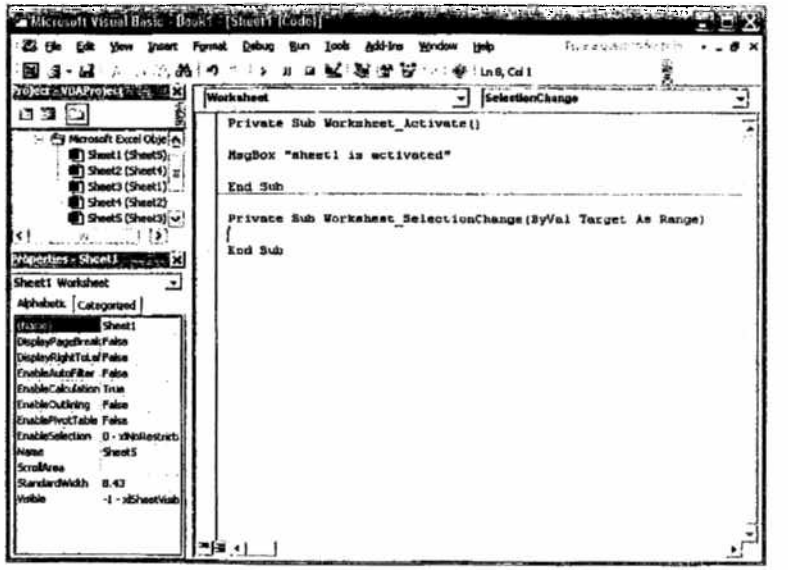
```
'MsgBox "Hello World"
```

از این پس،این خط به رنگ سبز در می‌آید و به عنوان یک دستور در نظر گرفته نمی‌شود. عبارات توضیحی برای قرار دادن شرح و تفسیر کد شما در یک ماکرو به‌کار گرفته می‌شوند تا کاربران بعدی بتوانند به راحتی آن‌را درک کنند.

آشنایی بیشتر با پنجره پروژه VBA

اگر مجدداً به سراغ Project Explorer و ساختار درختی نمایش دهنده پروژه برویم، اشیای دیگری نیز برای هر کاربرگ موجود در کارپوشه وجود دارند. حال باید یک شئ جدید برای کاربرگی که در مثال قبلی درج کردید در ساختار درختی نمایش داده شده باشد.

اگر روی هر کدام از این اشیا دابل کلیک کنیم، یک پنجره کدنویسی جدید ظاهر خواهد شد. باز هم در قسمت نمودار بازشو در گوشه بالای سمت چپ، گزینه General را می‌بینیم که می‌توان با کلیک روی آن، آن‌را به Worksheet تغییر داد. کاربرگ‌ها دارای رویدادهای متفاوت و کمتری نسبت به شئ WorkBook هستند. کد را می‌توان طوری تنظیم کرد تا در زمان فعال یا غیر فعال بودن برگه یا در زمان محاسبه آن اجرا شود. برای مثال می‌توانید کد نمایش داده شده در تصویر ۶-۱ را بنویسید و از آن پس، هر بار که Sheet1 از طریق کنترل‌های زبانه که در پایین پنجره Excel قرار دارند انتخاب شود، پیغام "Sheet1 is selected" به نمایش در می‌آید.



تصویر ۶-۱

فصل دوم

متغیرها، آرایه‌ها، ثابت‌ها و انواع داده‌ها

اگر چه معنای داشتن مکانی برای ذخیره کردن داده‌ها در حین اجرای برنامه، ساده و صریح است اما متغیرها[1]، آرایه‌ها[2] و ثابت‌ها[3] دارای قوانین نسبتاً پیچیده‌ای هستند. آن‌ها برای هر برنامه‌ای، الزامی هستند و بدین خاطر، یک فصل کامل را به تشریح آن‌ها اختصاص داده‌ایم.

متغیرها

ایجاد ساختاری از متغیرها شبیه ساخت یک قفسه نگهداری فایل‌ها در منزل است. ممکن است فایل‌های کاری، ورقه‌های بیمه، مالیات و اسناد شخصی داشته باشیم. حجم این فایل‌ها ممکن است خیلی بزرگ باشد. بعضی از این فایل‌ها ممکن است تنها حاوی یک سند باشند (مانند اسناد بیمه) در حالی‌که بقیه ممکن است اطلاعات زیادی را در خود جای داده باشند (مانند فایل مالیات).

نکته مهم این است که هر فایل، دسته خاصی از اسناد را در خود جا می‌دهد. مثلاً شما هیچ وقت پاسپورت خود را در فایل اسناد مالیات قرار نمی‌دهید. پس برای دسترسی ساده و راحت،باید بدانید هر فایل، چه اسنادی را در خود جای می‌دهد.

یک متغیر را می‌توان مشابه با یک فایل خاص که حاوی نوع مشخصی از اطلاعات است در نظر گرفت. متغیر ممکن است یک عدد یا یک متن باشد که برنامه در حین اجرا به آن‌ها نیاز دارد. متغیر می‌تواند یک آرایه کامل از اطلاعات (تقریباً شبیه یک صفحه گسترده) نیز باشد. یک صفحه گسترده چندین سلول دارد که می‌تواند اطلاعات را در خود جای دهد و یک آرایه می‌تواند ایجاد شود تا مقدار موجود در چندین سلول را در خود جای دهد.

یک متغیر می‌تواند مقداری داشته باشد که توسط برنامه،در حین اجرای آن،تغییر کند و بدین خاطر است که متغیر نامیده می‌شود. اگر یک متغیر را از نوع خاصی تعریف کنیم، داده‌های از نوع دیگر را قبول نخواهد کرد. برای مثال اگر یک متغیر را از نوع عدد صحیح (integer) تعریف کنیم، نمی‌توانیم

1- Variables
2- Arrays
3- Constants

عبارات متنی (string) را به آن اختصاص دهیم یا اگر یک عدد اعشاری را به آن اختصاص دهیم، قسمت اعشاری آن از بین خواهد رفت و عدد گرد می‌شود.

معمولاً در زمان اجرای برنامه، به جایی برای ذخیره سازی موقت داده‌ها نیاز دارید. در گذشته، برنامه‌نویسان ماکرو اغلب این داده‌ها را روی خود صفحه گسترده ذخیره می‌کردند . با VBA نیز می‌توان همین کار را انجام داد و داده‌ها را در سلول‌های موجود روی خود صفحه گسترده نوشت. اما این کفایت نمی‌کند و در این حالت، ما (به عنوان کدنویس) نیازمند داشتن حافظه‌ای خوب برای سازمان‌دهی ذهنی هستیم جایی که هر تکه از داده‌های موجود روی صفحه گسترده در آنجا ذخیره می‌شود. هم‌چنین مشکل دیگر این‌است که چون افراد به تغییر صفحات گسترده تمایل دارند و هر شخص نیز می‌تواند به سادگی، متغیرها را پاک یا بازنویسی کند و سپس باعث می‌شود که برنامه از کار بیفتد یا این‌که نتایج نادرستی را برگرداند.

ما می‌توانیم از متغیرها، برای ذخیره سازی مقادیر، در حین اجرای کد خود استفاده کنیم. در یک رویه(Procedure) می‌توانیم با استفاده از دستور Dim، یک متغیر را اعلان کرده و نامی برای آن تعیین کنیم:

Dim variablename [As type]

نام‌گذاری متغیرها از قوانین زیر تبعیت می‌کند:

- باید با یک حرف آغاز شوند.
- تنها باید حاوی حروف، اعداد یا کاراکتر زیر خط (_) باشند و در آن‌ها نباید از کاراکتر فاصله(space) استفاده کرد.
- نام‌ها نباید بیش از ۴۰ کاراکتر داشته باشند.
- نباید از کلمات ذخیره شده[1] به عنوان نام متغیرها استفاده کنیم.

عبارت اختیاری As type به ما اجازه می‌دهد تا نوع داده متغیری را که اعلان می‌کنیم، تعریف نماییم. اگر نوع داده متغیری را مشخص نکنیم، به صورت پیش‌فرض نوع Variant که در بخش بعدی بررسی خواهد شد برای آن به‌کار گرفته می‌شود:

```
Dim MyInteger as Integer
```

تعریف ضمنی متغیر (Implicit Declaration)

مجبور نیستیم قبل از استفاده از یک متغیر، آن‌را تعریف کنیم. کافی است دستور زیر را به کد اضافه

1- Reserved Word

کنیم:

```
TempVal=6
```

به طور خودکار، متغیری با نام TempVal از نوع Variant (نوع پیش‌فرض) ایجاد می‌شود و مقدار عددی 6 به آن اختصاص می‌یابد.

با وجود این، مشکل فعلی این است که در صورتی که نام متغیر را در دستور بعدی، غلط تایپ کنیم، منجر به بروز خطاهای اساسی در کد شما خواهد شد. برای مثال اگر به جای tempval از نام temval استفاده کنید،VBA متوجه نمی‌شود که شما از هر دو کلمه یک منظور را دنبال می‌کنید و فرض می‌کند که temval یک متغیر جدید است. متغیر قبلی (یعنی tempval) هنوز هم وجود دارد اما دیگر مورد استفاده قرار نمی‌گیرد. اکنون دو متغیر متفاوت داریم، هرچند ما فکر می‌کنیم تنها یک متغیر داریم. این امر می‌تواند منجر به بروز مشکلات جدی شود که یافتن و برطرف کردن آن‌ها در کد، مدت زمانی را تلف می‌کند.

تعریف صریح متغیر (Explicit Declaration)

برای پرهیز از بروز مشکل در نام‌گذاری نادرست متغیرها، می‌توانید VBA را طوری تنظیم کنید که همیشه در مواجهه با متغیری که اعلان نشده است، یک پیغام خطا صادر کند. برای انجام این کار باید به بخش تعریف‌ها (Declarations) در ماژول کد بروید. اگر در پنجره VB Editor در قسمت Module کلیک کنید، خواهید دید که یک عنوان با نام (General) در بالا و سمت چپ پنجره ماژول و یک عنوان با نام (Declarations) در بالا و سمت راست پنجره ماژول وجود دارد. دستور زیر را تایپ کنید. به محض آن‌که یک اعلان متغیر را تایپ کنید،خطی به صورت خودکار در زیر آن ظاهر می‌شود تا مشخص کند که در بخش تعریف‌ها قرار دارید:

```
Option Explicit
```

این کار باعث جلوگیری از به‌کارگیری تعریف‌های ضمنی متغیر می‌شود. حال باید TempVal را تعریف کنید:

```
Dim TempVal
```

نکته: تعریف متغیر به صورت صریح را باید برای هر ماژول به طور جداگانه و در قسمت Declarations آن ماژول انجام دهید مگر آن‌که بخواهید متغیر را به صورت عمومی (global) تعریف کنید.

استفاده از شیوه تعریف فوق (ضمنی یا صریح) بستگی به سلیقه شخصی کاربر دارد. در اغلب موارد، استفاده از تعریف ضمنی متغیر باعث بالا رفتن سرعت کدنویسی می‌شود، چون مجبور نیستید پیش از استفاده از متغیرها، آن‌ها را تعریف کنید. کافی است شما دستورات را صادر کنید، VBA خودش بقیه کارها را انجام می‌دهد. البته همان‌طور که قبلاً گفته شد این‌کار می‌تواند منجر به بروز خطاهایی شود مگر آن‌که حافظه خوبی برای به خاطر سپردن متغیرهایی که استفاده می‌کنید داشته باشید و از تجربه کافی در رابطه با کاری که انجام می‌دهید نیز برخوردار باشید. استفاده از تعریف ضمنی متغیرها می‌تواند فهم کد ما را برای دیگر کاربران نیز مشکل کند. استفاده ازOption Explicit ،بهترین راه حل است و کمک می‌کند تا خطاهای زمان اجرای کمتری (runtime errors) رخ دهد.

محدوده (Space) و طول عمر (Lifetime) متغیرها

اگر متغیری را در یک رویه تعریف کنیم، تنها کد موجود در آن رویه است که می‌تواند به آن متغیر دسترسی پیدا کند. اما اغلب به متغیرهایی نیاز داریم که بتوان از آن‌ها در چندین رویه یا حتی کل برنامه استفاده کرد. به این خاطر، می‌توانیم یک متغیر را در سه سطح محلی، ماژول یا سراسری تعریف کنیم.

متغیرهای محلی

یک متغیر محلی از دستور Static،Dim یا ReDim (فقط برای آرایه‌ها) برای تعریف یک متغیر در یک رویه استفاده می‌کند. ممکن است درچند رویه، متغیری با نام temp وجود داشته باشد اما چون هر متغیر نسبت به رویه‌ای که در آن قرار دارد محلی فرض می‌شود، تمام آن‌ها مستقل از یکدیگر عمل می‌کنند و می‌توانند حاوی مقادیر متفاوتی باشند. متغیرهای محلی که با دستور Dim تعریف شده باشند تنها تا پایان اجرای رویه، باقی می‌مانند. متغیرهای محلی که با دستورStatic ساخته شده باشند، تا پایان اجرای برنامه باقی می‌مانند چون ممکن است بخواهیم مقدار یک متغیر را در سراسر برنامه حفظ کنیم. اگر نگاهی به فصل ۱۸ بیندازید، مثالی را خواهید دید که نشان می‌دهد یک متغیر استاتیک چگونه کد ما را تغییر می‌دهد:

```
Dim TempVal
Static TempVal
```

هم‌چنین می‌توانیم یک متغیر را به صورت آرایه‌ای حاوی چندین عنصر و یا حتی آرایه‌ای چند بعدی تعریف کنیم. از لحاظ مفهومی، آرایه تقریباً شبیه یک صفحه گسترده است. می‌توانیم یک آرایه را با 10

عنصر تعریف کنیم به شکلی که 10 سلول یا خانه برای ذخیره کردن اطلاعات داشته باشد. هم‌چنین می‌توانیم آن‌را به صورت دوبعدی نیز تعریف کنیم تا به صورت یک آرایه 10 در 10 در آید که دارای 100 سلول یا خانه برای ذخیره کردن اطلاعات است.آرایه‌ها از لحاظ ذخیره کردن داده‌ها انعطاف پذیری زیادی دارند. برای مثال اگر شما تمام زیرفهرست‌های یک درایو را جستجو کنید، به یک آرایه برای ذخیره کردن تمام مسیرهای یافت‌شده در درایو مورد نظر احتیاج خواهید داشت تا در برنامه بتوانید به سادگی به آن‌ها دسترسی داشته باشید:

```
Dim A (3)
ReDim A (10)
ReDim Preserve A (12)
```

برای استفاده از ReDim، باید متغیر را در ابتدا به صورت آرایه تعریف کرده باشید (به مبحث "Arrays" در ادامه همین فصل رجوع شود). Dim A (3)، یک آرایه کوچک با چهار عنصر (عنصر 3-0) می‌سازد. پس در اصل چهار متغیر A وجود خواهد داشت. دستور ReDim A (10)، باعث می‌شود آرایه فوق به یک آرایه 11 عنصری تبدیل شود و البته تمام داده‌های قبلی آن از بین می‌رود. دستور ReDim A(12) Preserve، یک آرایه 13 عنصری است و در عین حال، اطلاعات موجود در اعضای اولیه آرایه (قبل از تغییر تعداد اعضا) را حفظ می‌کند. توجه کنید که شماره اندیس اعضا از صفر آغاز می‌شود.

دستور ReDim برای زمانی مفید است که به آرایه‌ای برای ذخیره سازی داده‌ها احتیاج داشته باشیم اما تعداد عناصر مورد نیاز آن آرایه را ندانیم. برای مثال اگر در حال جستجو در فهرست‌ها (directory) و پوشه‌های هارد خود هستید مسلماً از قبل نمی‌دانید تعداد آن‌ها چند عدد است پس کار را با یک آرایه کوچک، مثلاً 10 عضوی آغاز می‌کنیم. وقتی این آرایه پر شد به راحتی می‌توانیم به کمک یک ReDim و Preserve ،داده‌هایی را که از قبل در آن بوده است، حفظ کنیم.

متغیرها در سطح ماژول

یک متغیر در سطح ماژول برای یک ماژول خاص تعریف شده است. این متغیر برای تمام رویه‌های موجود در آن ماژول قابل دسترسی است اما در بقیه برنامه قابل استفاده نیست. متغیرها در سطح ماژول تا آخر برنامه باقی مانده و مقدار خود را حفظ می‌کنند.

```
Dim TempVal
```

دستور بالا به جای آن‌که در بخش declarations از یک رویه قرار گیرد، در بخش declarations از خود ماژول قرار می‌گیرد.

متغیرهای سراسری

این متغیرها، با دستور Global در بخش declarations ماژول تعریف می‌شوند اما در همه جای برنامه می‌توان به آن‌ها دسترسی پیدا کرد. متغیرهای سراسری در سراسر برنامه باقی مانده و مقدار خود را حفظ می‌کنند.

```
Global TempVal
```

این دستور را نیز می‌توان در بخش declarations هر ماژولی قرار دارد. چون مشخص کرده‌ایم که این متغیر، سراسری (Global) است. پس می‌توانیم از هر جای کد به آن دسترسی داشته باشیم.

تناقض در نام متغیرها

در هنگام اجرای کد نمی‌توان حوزه یک متغیر را تغییر داد. البته می‌توان متغیرها را با یک نام در حوزه یا ماژول متفاوتی به کار گرفت. به عنوان مثال می‌توانیم متغیری سراسری با نام Temp داشته باشیم و در عین حال یک متغیر محلی با نام temp نیز در یکی از رویه‌هایمان داشته باشیم. اگر در داخل رویه به متغیر temp ارجاع کنیم، به متغیر محلی temp دسترسی خواهیم داشت. در چنین حالتی اصطلاحاً می‌گوییم کد متغیر محلی در زیر سایه متغیر سراسری قرار دارد. تنها روشی که باعث می‌شود مطمئن شویم متغیر دلخواه ما به کار گرفته شده، این است که از نام‌های متفاوت برای هر کدام از آن‌ها استفاده کنیم.

نام متغیرها در سطح ماژول و یا در سطح سراسری می‌تواند تناقض‌هایی با نام رویه‌ها نیز داشته باشد. یک رویه(یا زیرروال) دارای حوزه عمومی است،مگر آن‌که آن را به طور خصوصی تعریف کنیم (یک مثال از این حالت را در فصل ۳ خواهیم دید). یک متغیر سراسری نمی‌تواند در هیچ ماژولی نام مشابه با یک رویه محلی داشته باشد.

متغیرهای ایستا (Static Variables)

متغیرها بر مبنای حوزه خود، طول عمر (Lifetime) مشخصی دارند. متغیرهای ماژول و سراسری در سراسر برنامه باقی می‌مانند و این به معنای آن است که آن‌ها مقادیر خود را تا زمانی که اجرای برنامه خاتمه پذیرد نگه می‌دارند. متغیرهای محلی که با Dim تعریف شده‌اند، تنها تا زمان پایان اجرای رویه خود باقی می‌مانند. وقتی اجرای آن رویه متوقف شود مقادیر موجود در آن متغیرها از بین رفته و حافظه اختصاص یافته به آن‌ها نیز آزاد می‌شود. در مرتبه بعدی اجرای آن رویه، متغیرها نیز مجدداً

فعال می‌شوند و البته خبری از مقادیر قبلی آن‌ها نخواهد بود. حتماً باید برای حفظ مقادیری که تنها در یک رویه خاص کاربرد دارند از متغیرهای محلی استفاده کنیم. اگر می‌خواهید از رویه‌های دیگر هم به آن متغیرها دسترسی داشته باشید، لازم است متغیرهای مورد نظر را به صورت سراسری تعریف کنید.البته می‌توانید از کلمه کلیدی Static برای تعریف و حفظ یک متغیر محلی استفاده کنید:

```
Static Temp
```

با قرار دادن کلمه کلیدی Static در آغاز عنوان یک رویه، می‌توانید تمام متغیرهای موجود در آن‌را به صورت ایستا تعریف کنید:

```
Static Sub Test_Static()
```

انواع داده‌ها (Data Types)

برای تعیین نوع اطلاعاتی که در یک متغیر قابل ذخیره کردن است، می‌توانیم یک نوع داده برای آن مشخص کنیم. اگر هیچ نوع داده‌ای برای یک متغیر تعریف نشده باشد، نوع Variant برای آن در نظر گرفته می‌شود.

نوع داده قابل تغییر (Variant)

یک Variant می‌تواند هر نوع داده‌ای را در خود جای دهد (متن، عدد، تاریخ یا هر اطلاعات دیگر) و حتی می‌تواند یک آرایه کامل را در خود جای دهد. یک متغیر از نوع Variant می‌تواند آزادانه در حین اجرای برنامه، نوع خود را تغییر دهد در صورتی که انواع دیگر را نمی‌توان در حین اجرای برنامه تغییر داد. می‌توانید از تابع VarType برای یافتن نوع داده موجود در یک متغیر از نوع Variant، استفاده کنید:

```
Sub TestVariables()
stemp = "richard"
MsgBox VarType(stemp)
stemp = 4
MsgBox VarType(stemp)
End Sub
```

کادر پیغام، ابتدا مقدار 8 را نمایش می‌دهد که به معنای آن است که آن متغیر، یک رشته است و سپس مقدار 2 را نشان می‌دهد که به معنای آن است که آن متغیر، عدد صحیح (integer) است.

جدول ۱-۲ نشان‌دهنده مقادیر برگشتی به ازای انواع مختلف داده‌ای توسط تابع VarType است.

جدول ۱-۲ مقادیر برگشتی تابع VarType

نوع داده	مقدار برگشتی
Empty	0
Null	1
Integer	2
Long	3
Single	4
Double	5
Currency	6
Date / Time	7
String	8

VBA همیشه از مؤثرترین ابزار برای ذخیره سازی داده‌ها در یک متغیر از نوع Variant استفاده می‌کند. همان‌گونه که در مثال قبلی می‌توان دید،VBA بطور خودکار تغییرات را اعمال می‌کند تا با داده‌های ذخیره شده در متغیر Variant تطبیق یابد.

اگر یک عملیات ریاضی را در یک متغیر از نوع Variant که حاوی مقدار عددی نیست اجرا کنیم، یک خطای Type MisMatch صادر خواهد شد. این پیغام به معنای آن‌است که می‌خواهیم نوع داده‌ای را در متغیری قرار دهیم که برای آن نوع داده تنظیم نشده است و در مثال ما به معنای آن است که می‌خواهیم یک عملیات ریاضی را روی متغیری که حاوی یک رشته متنی است اجرا کنیم.

از تابع IsNumeric برای آزمایش این‌که آیا مقدار Variant، عددی است یا خیر استفاده می‌کنیم.

نکته: خروجی این تابع در صورتی که مقدار متغیر، عددی باشد True و در غیر این صورت، False را برمی‌گرداند.

```
Sub TestNumeric( )
temp = "richard"
MsgBox IsNumeric(temp)
End Sub
```

جواب کد فوق، False است.

مقادیر Date/Time (تاریخ/زمان) که در متغیرهایی از نوع Variant ذخیره شده‌اند

متغیرهای Variant می‌توانند حاوی مقادیر Date/Time نیز باشند. این نوع متغیر، یک متغیر اعشاری است که قسمت صحیح آن نشان دهنده تعداد روزهای طی شده از تاریخ 13 دسامبر 1899 و قسمت اعشاری آن نشان دهنده ساعات،دقایق و ثانیه‌های طی شده است که به صورت بخشی از 24 ساعت بیان می‌شود. برای مثال 37786.75 نشان دهنده ساعت 18:00 در تاریخ 14 ژوئن 2003 است. تفاوت بین 31 دسامبر 1899 و 14 ژوئن 2003، 37786 روز است و 0.75 از 24 ساعت برابر با 18 ساعت است. اضافه یا کم کردن از این عدد، باعث بالا یا پایین رفتن روزها است. اضافه کردن به بخش اعشاری باعث زیاد شدن ساعت می‌شود. توجه کنید که تفسیر روز و ماه، وابسته به تنظیمات Regional Options در بخش Control Panel سیستم عامل ویندوز است. اگر در Regional Options، نمایش تاریخ را به صورت mm/dd/yy تنظیم کرده باشید، نمایش پیش‌فرض روز و ماه به این صورت در خواهد آمد.

همان‌طور که می‌توانیم از تابع IsNumeric استفاده کنیم تا مشخص کنیم که مقدار متغیر، یک مقدار عددی است می‌توانیم از تابع IsDate نیز برای تعیین این‌که آیا مقدار یک متغیر، تاریخ است یا خیر استفاده کنیم:

```
temp = "01-Feb-2002"
MsgBox IsDate(temp)
```

کد فوق، مقدار True (غیرصفر) را برمی‌گرداند.

مقدار تهی (Empty Value)

یک متغیر Variant که هنوز مقداری به آن اختصاص نیافته دارای مقدار تهی است. این امر را می‌توانیم با تابع IsEmpty کنترل کنیم:

```
MsgBox IsEmpty(temp)
```

این کد، مقدار True (غیر صفر) را برمی‌گرداند چون هنوز به متغیر temp، مقداری اختصاص نیافته است.

مقدار پوچ (Null Value)

یک متغیر Variant می‌تواند حاوی مقدار خاص Null باشد. مقدار Null برای این استفاده شده است که داده‌های مجهول یا داده‌های از بین رفته را نشان دهد. متغیرها، حاوی مقدار Null نخواهند بود مگر آن‌که کدی برای این کار بنویسید. اگر در برنامه خود از Null استفاده نمی‌کنید، لازم نیست درباره این نوع مقداردهی نگران باشید.

ساده‌ترین راه برای کنترل این‌که یک مقدار Null در کد شما وجود دارد یا خیر، استفاده از تابع IsNull است. روش‌های دیگر مانند دستور Is Null ممکن است نتایج نادرستی برگردانند:

```
Sub TestNull ( )
temp=Null
MsgBox IsNull(temp)
End Sub
```

نتیجه کد فوق،True (غیر صفر) است.

دیگر انواع داده

چرا باید از نوع داده دیگری غیر از Variant استفاده کنیم؟ چون Variant ممکن است بهترین نوع داده برای منظور ما نباشد. اگر بخواهید یک کد کوتاه و کارآمد ایجاد کنید به انواع دیگر داده نیز احتیاج خواهید داشت. برای مثال اگر بخواهید محاسبات ریاضی پیچیده‌ای روی اعداد صحیح کوچک انجام دهید، با استفاده از نوع داده Integer به جای Variant، می‌توانید از سرعت زیادی در اجرای کار بهره‌مند شوید.

شما می‌توانید به صورت پیش‌فرض از نوع Variant استفاده کنید اما این متغیر الزاماً فرض نمی‌کند که شما از یک عدد صحیح (integer) استفاده می‌کنید، بلکه فرض می‌کند از یک عدد اعشاری استفاده می‌کنید و این باعث می‌شود تا محاسبات در زمان طولانی‌تری انجام شود. هرچند در نتیجه و بازده کار، تغییری حاصل نمی‌شود.

در رابطه با نوع داده‌ها باید به ملاحظات موجود درمورد حافظه کامپیوتر نیز توجه شود. هر متغیر از نوع Double، ۸ بایت از حافظه را اشغال می‌کند. هر چند این مقدار زیاد به نظر نمی‌رسد، اما در یک آرایه بزرگ، مقدار زیادی از RAM را اشغال خواهد کرد که باعث کندی فرآیند انجام عملیات می‌شود. این فضای اشغال شده در حافظه کامپیوتر می‌تواند توسط سیستم عامل ویندوز به عنوان حافظه مجازی برای نمایش گرافیکی آن به کار گرفته شود.

انواع داده VBA

چند نوع داده وجود دارند که در VBA قابل استفاده هستند و جزییات آن‌ها در جدول ۲-۲ آمده است.

جدول ۲-۲ انواع داده‌ها در VBA

نام	شرح	کاراکتر معرف	دامنه
Integer	عدد صحیح 2 بایتی	%	32767-تا32768
Long	عدد صحیح 4 بایتی	&	2147438647-تا2147483648
Single	عدد اعشاری 4 بایتی	!	1.401298E-45-تا3.402823E38 (اعداد منفی) 3.402823E38تا1.401298E-45 (اعداد مثبت)
Double	عدد اعشاری 8 بایتی	#	-1.79769313486232D308 -4.94065645841247D-324 (مقادیر مثبت) 4.94065645841247D-324تا 1.79769313486232D308 (مقادیر مثبت)
Currency	عدد 8 بایتی با اعشار ثابت	@	922337203685477.5808-تا 922337203685477.5807
Fixed Length String	رشته‌ای از کاراکترها با طول ثابت	$	صفر تا تقریباً 65500 کاراکتر
Variale Length String	رشته‌ای از کاراکترها با طول متغیر	$	صفر تا تقریباً 2 بیلیون کاراکتر
Variant	هر داده‌ای می‌تواند باشد	ندارد	همه نوع داده

انواع عددی

اگر با اعداد صحیح کار می‌کنید، لازم است بسته به اندازه متغیرها، از نوع Integer یا Long استفاده کنید. عملیات ریاضی برای این انواع داده، سریع‌تر و مقدار حافظه مورد نیاز نیز کمتر خواهد بود.
اگر با اعداد اعشاری کار می‌کنید باید از نوع داده Double، Single یا Currency استفاده کنید. نوع داده Currency، اعدادی را قبول می‌کند که قسمت صحیح آن‌ها تا ۱۵ رقم و قسمت اعشاری آن‌ها تا ۴ رقم باشد. این محدودیت در مورد داده‌های از نوع Single و Double وجود ندارد:

```
Dim temp as Integer
Dim temp as Long
Dim temp as Currency
Dim temp as Single
Dim temp as Double
```

انواع رشته‌ای (String Types)

اگر قرار است متغیرها حاوی متن باشند، بهتر است آن‌را از نوع رشته‌ای String تعریف کنید:

```
Dim temp as String
```

در ادامه کار می‌توانید از توابع رشته‌ای برای دست‌کاری آن استفاده کنید. می‌توانید بخش‌هایی از آن را جدا کرده و در آن به دنبال کاراکتر خاصی بگردید یا این‌که تمام حروف آن را به بزرگ یا کوچک تبدیل کنید. برای توضیحات بیشتر به بخش Function در فصل ۵ رجوع کنید.
یک متغیر رشته‌ای به صورت پیش‌فرض، دارای طول متغیر است. متغیرهای رشته‌ای بر طبق میزان داده (متن) موجود در آن، بزرگ یا کوچک می‌شوند. اگر نخواهید چنین امری رخ دهد می‌توانید با استفاده از دستور String*size، یک متغیر رشته‌ای را با طول ثابت تعریف کنید:

```
Dim temp as String * 50
```

این دستور باعث می‌شود، طول متغیر temp برابر با 50 کاراکتر باشد. اگر طول رشته شما کمتر از 50 باشد، کاراکترهای باقیمانده، با فاصله خالی پر می‌شوند. پس هر چند که دستور فوق به ما کمک می‌کند تا کنترل میزان حافظه به کار گرفته شده را به دست گیریم اما همیشه این خطر وجود دارد که مقداری از داده‌ها از بین بروند.

آرایه‌ها

تا این‌جای کار روی متغیرهای تکی بحث کردیم. می‌توانیم یک متغیر را تعریف کرده و مقداری (عددی یا متنی) به آن اختصاص دهیم. یک مثال ساده این قضیه، نام یک کارمند است. می‌توانیم متغیری با نام employee ایجاد کرده و رشته‌ای (نام کارمند) را به آن اختصاص دهیم. اما تکلیف کارمندان دیگر چه می‌شود؟ فرض کنید قرار است برنامه‌ای بنویسید که برای هر ۲۶ کارمند یک شرکت اجرا می‌شود. اگر بخواهید برای نام‌گذاری هر کدام از آن‌ها، یک متغیر تعریف کنید، در ادامه برای ارجاع به آن‌ها با مشکل مواجه خواهید شد. احتمالاً دستوری شبیه نمونه زیر خواهید داشت:

```
Dim employee1 as String, employee2 as String, employee3 as String, .....
```

این کد بسیار پر دردسر و ناکارآمد است. مسأله مهم این‌جاست که اگر قرار باشد بعداً کارمندانی به تعداد فعلی اضافه شوند چه اتفاقی رخ خواهد داد؟ حتماً برنامه با مشکل مواجه خواهد شد.
خوشبختانه،یک متغیر را می‌توان به صورت آرایه هم تعریف کرد. تمام آن‌چه مورد نیاز است،کدی شبیه کد زیر است:

```
Dim employee(25) as String
```

همان‌طورکه قبلاً هم گفتیم، یک آرایه را می‌توان شبیه مجموعه‌ای از سلول‌ها در یک صفحه گسترده در نظر گرفت که می‌توان با استفاده از شماره اندیس آن آرایه، به خانه مورد نظر رسیده و داده‌ها را خواند یا در آن نوشت. برای نشان دادن این‌که به کدام آرایه استناد می‌کنیم، شماره اندیس آن‌را داخل پرانتز می‌نویسیم. هم‌چنین در صورتی که در برنامه به این‌کار نیاز پیدا کنیم می‌توانیم با دستور ReDim، ابعاد آن‌را تغییر دهیم.
این مثال، یک آرایه 26 عنصری ایجاد می‌کند که از 0 تا 25 شماره‌گذاری شده است و می‌توان 26 رشته را به عنوان نام کارمندان در اعضا آن ذخیره کرد.
می‌توانیم برای فهرست کردن نام‌ها در آرایه از یک حلقه For..Next استفاده کنیم:

```
Dim employee(25) as String
For n = 0 To 25
        Employee(n) = Chr(n+65)
Next n
For n = 0 To 25
        MsgBox employee(n)
Next n
```

در این ماکرو، ابتدا آرایه‌ای با نام employee را از انواع رشته‌ای با 26 عنصر تعریف می‌کنیم.
حلقه For..Next نخست، داده‌ها را در آرایه قرار می‌دهد. کد اسکی برای حرف A، 65 است و این به مقدار n، که در آغاز صفراست اضافه می‌شود. تابع Chr، این عدد را به یک کاراکتر تبدیل می‌کند که در محل مناسبی از آرایه درج می‌شود. در حلقه نخست، کاراکتر A در عنصر نخست آرایه درج می‌شود چون در این نقطه، n برابر صفر است. مقدار 65 (که کد اسکی کاراکتر A است) به آن اضافه می‌شود.
حلقه دوم For..Next، هر عنصر آرایه employee را به ترتیب نمایش می‌دهد. وقتی کد این مثال را اجرا کنید، حروف A تا Z را به عنوان خروجی برمی‌گرداند.
آرایه‌ها از همان قوانین متغیرهای معمولی تبعیت می‌کنند. آن‌ها می‌توانند محلی، ماژولی یا سراسری باشند و می‌توانند هر نوع داده‌ای را قبول کنند. اندازه آرایه بر حسب نوع عناصر موجود در آن، محدود به یک عدد صحیح (در دامنه 32767 تا 32768-) است. حد پایینی پیش‌فرض برای آرایه، صفر است اما این حد را می‌توان با استفاده از دستور Option Base در بخش declarations ماژول تغییر داد:

```
Option Base 1
```

پس از اجرای دستور فوق، تمام آرایه‌های موجود در ماژول از 1 آغاز می‌شوند.
با استفاده از کلمه کلیدی To نیز می‌توانید حد پایینی آرایه را مشخص کنید:

```
Dim temp (1 To 15) as String
```

آرایه‌های چند بعدی

در بخش قبلی تنها آرایه‌های تک بعدی را تعریف کردیم اما می‌توانیم یک آرایه چند بعدی نیز داشته باشیم. چنین آرایه‌ای را می‌توان مانند یک صفحه گسترده در نظر گرفت. همان‌طور که یک صفحه گسترده دارای ردیف و ستون است، آرایه‌های چند بعدی هم چنین حالتی دارند:

```
Dim temp(10,4) as String
```

اگر این آرایه، یک صفحه گسترده بود 11 ستون و 5 ردیف و مجموعاً 55 سلول برای دریافت داده داشت.
یک آرایه سه بعدی را به صورت زیر تعریف می‌کنیم:

```
Dim temp(10,4,3) as String
```

باز هم می‌توانیم این آرایه را یک صفحه گسترده در نظر بگیریم که دارای 11 ستون و 5 ردیف است و سلول‌های آن در بین 4 کاربرگ پخش شده و مجموعاً 220 سلول را تشکیل می‌دهند. به یاد داشته

باشید هرکدام از این عناصر می‌توانند رشته‌ای تا 65500 کاراکتر را قبول کنند و در نظر بگیرید که چنین آرایه ساده‌ای چقدر از حافظه را اشغال می‌کند و چه میزان داده را می‌توان در آن ذخیره کرد.
با تعیین ابعاد یک آرایه بلافاصله حافظه مورد نیاز به آن اختصاص می‌یابد (این امر در هنگام نوشتن برنامه مهم است). اشغال کردن مقدار زیادی از حافظه می‌تواند باعث ایجاد مشکل در عملکرد برنامه شما و سیستم عامل ویندوز شود. چون ویندوز، یک برنامه کاربردی گرافیکی است، مقدار زیادی از RAM را برای نگه داشتن اطلاعات مورد نیاز خود، اشغال می‌کند. ممکن است در ادامه متوجه شوید استفاده از یک آرایه بزرگ باعث کند شدن ویندوز و دیگر برنامه‌های کاربردی می‌شود و مدت زمان بیشتری طول می‌کشد تا اطلاعات پردازش شوند.
این مکان وجود دارد که ابعاد آرایه را افزایش داد اما با این کار کنترل آرایه و دست‌کاری آن مشکل می‌شود. معمولاً حداکثر ابعاد یک آرایه را پنج بعد در نظر می‌گیریم. اگر این آرایه را هم یک صفحه گسترده در نظر بگیریم، حتماً متوجه می‌شوید که یک صفحه گسترده که پنج کاربرگ دارد، چقدر پیچیده است.
در آرایه‌های چند بعدی هم می‌توان از دستور ReDim برای تغییر اندازه آرایه استفاده کرد. اما نمی‌توان از آن برای تغییر تعداد ابعاد آرایه و یا نوع آرایه استفاده کرد.

آرایه‌های دینامیکی

گاهی اوقات نمی‌دانید به یک آرایه با چه اندازه‌ای نیاز دارید. به عنوان مثال وقتی قرار است مسیرهای موجود در یک هارددیسک خود را در عناصر یک آرایه ذخیره کنید، نمی‌دانید چه تعداد زیر شاخه در آن‌جا وجود دارد. می‌توانید یک آرایه 1000 عنصری بسازید و فضای ارزشمند حافظه را به آن اختصاص دهید اما بعداً متوجه شوید که تنها به 500 عضو احتیاج داشته‌اید. البته ممکن است بر عکس این قضیه رخ داده و 2000 زیر شاخه وجود داشته باشد و این‌کار ما را با کمبود فضا روبه‌رو کند.
آرایه‌های دینامیکی را دقیقاً شبیه آرایه‌های معمولی می‌سازیم (استفاده از دستور Dim در سطح سراسری، ماژول یا محلی و دستور Static در سطح محلی). با دستور زیر، آرایه‌ای بدون بعد می‌سازیم:

```
Dim temp( )
```

سپس از دستور ReDim در رویه خود استفاده می‌کنیم تا تعداد عناصر موجود در آرایه را تغییر دهیم. دستور ReDim تنها می‌تواند در یک رویه به کار گرفته شود و نمی‌توانید تعداد ابعاد آن‌را تغییر دهید:

```
ReDim temp(100)
```

شما می‌توانید کدی بنویسید تا تعداد مقادیر جمع آوری شده را کنترل کرده و در صورتی که به مرز بالایی آرایه نزدیک باشد، آن‌را تغییر اندازه دهد(UBound و LBound). این توابع، توابعی هستند که برای برگرداندن حد بالایی و پایینی ابعاد یک آرایه با مشخص کردن شماره آرایه به صورت یک اندیس به کار می‌روند:

```
Dim temp(10)

MsgBox LBound(temp)

MsgBox UBound(temp)
```

در دستور بالا، LBound مقدار 0 و UBound مقدار 10 را برمی‌گردانند.
ReDim به طور خودکار تمام مقادیر موجود در آرایه را پاک می‌کند مگر آن‌که از کلمه کلیدی Preserve استفاده کنیم:

```
ReDim Preserve temp(100)
```

با دستور فوق، داده‌هایی که از قبل در temp بوده‌اند نگه داشته خواهند شد.

انواع داده تعریف شده توسط کاربر

شما می‌توانید با استفاده از کلمه کلیدی Type و به کارگیری انواع متغیر موجود، نوع داده دلخواه‌تان را برای متغیرها تعریف کنید. این‌کار را در بخش declarations ماژول انجام می‌دهیم:

```
Type  Employee
        Name as String
        Salary as Currency
        Year as Integer
End Type
```

دستور فوق باعث ایجاد نوع داده جدیدی می‌شود که Employee نامیده می‌شود و اطلاعات Name، Salary و Years of Service را در خود نگه می‌دارد. این نوع داده را نیز دقیقاً مانند دیگر انواع داده‌ای می‌توان به‌کار برد. این داده حتی به طور خودکار در لیست‌های موجود در ویراستار VBA اضافه می‌شود. شما می‌توانید از این‌ها به عنوان متغیرهای نرمال استفاده کنید، همچنان‌که در مثال زیر نشان داده شده است:

```
Dim temp As employee
temp.Name = "Richard Shepherd"
temp.Salary = 10000
```



```
temp.Years = 5
MsgBox temp.Name
MsgBox temp.Salary
MsgBox temp.Years
```

توجه کنید که نام متغیر، یک کادر فهرست دارد که به موازات تایپ کردن نام متغیر در کد، خصوصیات یا فیلدهای این نوع داده را نشان می‌دهد.
می‌توانید آرایه را از نوع تعریف شده توسط خود تعریف کرده و از آن استفاده کنید.

ثابت‌ها (Constants)

ثابت‌ها، متغیرهایی هستند که تغییر نمی‌کنند. آن‌ها حاوی مقادیری هستند که در طی اجرای کد به‌کار گرفته می‌شوند. برای ایجاد ثابت‌ها باید از همان قوانین ایجاد متغیرها استفاده کنیم اما نمی‌توان مانند یک متغیر از داخل کد به آن مقدار جدید اختصاص داد:

```
Const Path_Name = "C:\temp\"
```

دستور فوق، ثابتی با نام Path_Name ایجاد می‌کند و همیشه دارای مقدار C:\temp\ است. هربار بخواهیم در برنامه از این مسیر استفاده کنیم این ثابت را به کار می‌گیریم.
در مدل شئ Excel، ثابت‌های از پیش تعریف شده‌ای نیز وجود دارند که می‌توان آن‌ها را با استفاده از Object Browser مرور کرد (به فصل ۱۲ مراجعه کنید). در مدل شئ Excel، تمام ثابت‌ها با حرف xl آغاز می‌شوند (مانند xlSaveChanges و xlDoNotSaveChanges).
Object Browser نشان دهنده مقدار واقعی ثابت در انتهای پنجره مرورگر نیز هست.

کلمات ذخیره شده (Reserved Words)

احتمالاً متوجه شده‌اید که چندین کلمه کلیدی در VBA وجود دارند که ساختار زبان ویژوال بیسیک را شکل می‌دهند (مانند For، Next، Do و Loop). از این کلمات نمی‌توان برای نام‌گذاری متغیرها، زیرروال‌ها یا توابع استفاده کرد، چون آن‌ها کلمات ذخیره شده محسوب می‌شوند و این به معنای آن است که آن‌ها قسمتی از خود زبان VBA به حساب می‌آیند و در صورت استفاده از آن‌ها، مشکلات جدی به‌وجود می‌آید. خوشبختانه هرآن‌چه تایپ می‌کنیم VBA کنترل کرده و در صورت بروز هرگونه خطایی، پیغام ارسال می‌کند. کد زیر را تایپ کنید:

```
Dim Loop as String
```

Loop قسمتی از VBA است و در دستور Do...Loop به کار گرفته می‌شود. اجرای این دستور، غیر ممکن است و بلافاصله با یک پیغام خطا مواجه می‌شویم و خط کد مربوط به آن قرمز می‌شود. البته می‌توانید اخطار را نادیده بگیرید اما به محض آن‌که سعی کنید کد را اجرا کنید، دوباره با پیغام خطا مواجه خواهید شد. کد زیر را اجرا کنید:

```
Sub ReDim( )
```

یک پیغام خطا دریافت خواهید کرد و کد نیز به رنگ قرمز در می‌آید چون ReDim یکی از کلمات کلیدی است.

فصل سوم

ماژول‌ها، توابع و زیرروال‌ها

ماژول‌ها، مکانی هستند که کد را در آن می‌نویسیم. توابع و زیر روال‌ها نیز دو روش متفاوت ایجاد کد هستند.

ماژول‌ها

ماژول‌ها، برگه‌های کد هستند که مختص برنامه کاربردی می‌باشند. آن‌ها مستقیماً توسط وقایعی که روی صفحه گسترده اجرا می‌شوند راه اندازی نمی‌گردند (وقایعی مانند اضافه کردن یک کاربرگ جدید یا بستن یک کارپوشه) بلکه باید آن‌ها را مستقیماً فراخوانی کنیم. آن‌ها به جای اجرا شدن در یک شئ مانند یک کارپوشه یا کاربرگ، ابزاری برای ایجاد رویه‌ها هستند. شما می‌توانید آن‌ها را به چند روش فراخوانی کنید:

- از یک دستور اختصاصی در منو یا یک دستور اختصاصی در نوار ابزار استفاده کنید (به فصل ۱۱ رجوع شود).
- یک کنترل VBA را مستقیماً از جعبه ابزار Control در صفحه گسترده وارد کرده و کد خود را به آن وصل کنید.
- از منوی Excel، Tools | Macro | Macros را انتخاب کنید. از لیست موجود نام ماکرو را انتخاب کرده و روی Run کلیک کنید. البته برای یک برنامه حرفه‌ای، این عمل را پیشنهاد نمی‌کنیم. کاربران دوست ندارند مجبور به انتخاب یک ماکرو از یک کادر لیست باشند. اغلب کاربران می‌خواهند کارها به راحت‌ترین شکل ممکن انجام شود (مثل از طریق کلیک کردن).
- کد را از UserForm اجرا کنید. در فصل ۹ یاد خواهید گرفت که چگونه فرم‌های دلخواه‌تان را تعریف کنید. از این فرم‌ها می‌توانید برای مشخص کردن گزینش‌های کاربران استفاده کنید. وقتی کاربر در فرمی بر روی دکمه OK کلیک کند، ماکروی شما اجرا شده و گزینه‌های انتخابی کاربر را هم در نظر می‌گیرد.
- کد خود را از یک ماکروی دیگر فرا بخوانید. کد می‌تواند زیر روال‌ها یا توابعی را شکل دهد که بعداً در دیگر ماکروهای نوشته شده در همان صفحه گسترده به‌کار گرفته شوند. برای مثال فرض کنید می‌خواهیم در یک رشته متن به دنبال کاراکتر خاصی بگردیم، برای انجام

این کار، یک زیر روال می‌نویسیم و برای انتقال متن به زیرروال از یک پارامتر استفاده می‌کنیم.

- کد را به صورت یک تابع نوشته و با درج تابع در یک سلول، آن‌را به صورت مستقیم فرا می‌خوانیم. در ادامه همین فصل یاد خواهید گرفت که چگونه یک تابع ساده صفحه گسترده بنویسیم که بتوان از آن به طور مستقیم در یک سلول استفاده کرد.
- مستقیماً روی کد کلیک کرده و کلید F5 را فشار دهید. برای مثال اگر روی یک زیر روال به صورت مجزا کار می‌کنید ممکن است بخواهید برای دیدن چگونگی کارکرد، آن‌را اجرا کنید.

در ادامه این کتاب، تمام متدهای فوق را به تفصیل بررسی خواهیم کرد.

پروژه VBA به طور نرمال حداقل از یک ماژول برای ذخیره سازی توابع مورد نیاز و زیر روال هایی که به عنوان رویه‌ها شناخته می‌شوند، استفاده می‌کند. برای درج ماژول جدید کافی است منوی VBE، Insert | Module را انتخاب کنیم. بلافاصله ماژول جدید ظاهر می‌شود. این ماژول در ابتدا فقط حاوی یک منطقه عمومی است و هیچ رویدادی روی آن وجود ندارد. شما می‌توانید توابع یا زیر روال‌ها را در این‌جا وارد کرده و آن‌ها را به صورت عمومی (Public) یا خصوصی (Private) درآورید. فرق بین خصوصی و عمومی در این است که آیا ماژول‌های دیگر در همان کارپوشه می‌توانند به آن رویه دسترسی داشته باشند یا خیر. اگر کد خصوصی باشد فقط می‌توانیم از آن در کارپوشه جاری که کد در آن‌جا واقع است استفاده کنیم. اگر یک زیر روال داشته باشیم که قرار نیست در جای دیگری از کد به‌کارگرفته شود، آن‌را به صورت یک زیر روال خصوصی در آورید. پیش‌فرض کار، همیشه عمومی است.

تفاوت بین زیر روال‌ها و توابع

دو نوع رویه کدنویسی وجود دارد:
زیرروال‌ها (Subroutines) و توابع (Functions)که در نگاه اول هر دوی آن‌ها مثل هم به نظر می‌رسند اما در اصل،تفاوت زیادی با هم دارند.

زیر روال، بخشی از کد است که مجموعه‌ای از عملیات یا محاسبات (یا هر دو) را انجام می‌دهد و می‌تواند یک " واحد مستقل" (building block) را در برنامه شکل دهد. زیر روال را می‌توان به چند شیوه فراخواند. برنامه‌نویس تنها یک بار زیر روال را می‌نویسد و از آن به بعد، هرچند بار که بخواهد آن‌را در برنامه فرامی‌خواند. البته زیر روال‌ها، مقداری را به عنوان خروجی بر نمی‌گردانند. به عنوان مثال اگر یک محاسبه را انجام دهند، برای یافتن نتیجه آن‌هاروشی مستقیم وجود ندارد. زیر روال را با درج دستور Call در کد فرا می‌خوانیم:


```
Sub Main()
    Call MySub    'Calls another marco/procedure called MySub
End Sub
```

مجبور نیستیم برای استفاده از زیر روال MySub از کلمه کلیدی Call استفاده کنیم. مثال زیر نتیجه‌ای مانند مثال قبل بر می‌گرداند:

```
Sub Main()
    MySub    'Calls another marco/procedure called MySub
End Sub
```

تابع دقیقاً شبیه زیر روال است و تنها تفاوت آن‌ها در این است که تابع، مقداری را بر می‌گرداند. توابع به جای کلمه Sub با Function آغاز و به جای End Sub با End Function خاتمه می‌یابند. این به معنای آن است که توابع را باید با یک متغیر فراخوانی کنیم تا مقدار برگشتی تابع به آن متغیر اختصاص یابد. (به فصل ۲ رجوع کنید):

```
x=Now()
```

در مثال فوق تاریخ فعلی در متغیر x ذخیره خواهد شد. این یک مثال بسیار ساده از فراخوانی یک تابع داخلی است.
توابع داخلی متعددی وجود دارند که به همین صورت قابل استفاده هستند. شما نمی‌توانید از توابع فرمولی Excel استفاده کنید، اما بسیاری از آن‌ها به عنوان بخشی از زبان برنامه‌نویسی، در VBA نیز وجود دارند. توابعی را که می‌نویسیم در فرمول‌های صفحه گسترده هم می‌توانیم به کار ببریم.
هم به زیر روال‌ها و توابع، می‌توان پارامترها یا مقادیری را انتقال داد. البته آن‌ها باید حتماً داخل پرانتز قرار گیرند.

نوشتن یک زیر روال ساده

یک زیر روال از آن جهت با تابع فرق دارد که نمی‌تواند چیزی را به صورت مستقیم برگرداند و نیز نمی‌توان از آن مانند یک تابع، در صفحه گسترده استفاده کرد. یک زیر روال معمولاً یک بلوک مستقل است که قسمتی از کد را شکل می‌دهد که چندین مرتبه و احتمالاً از نقاط مختلف برنامه فراخوانده خواهد شد و این یکی از نقاط قوت زیر روال‌ها محسوب می‌شود. وقتی زیر روال فراخوانده می‌شود، آدرس برگشت (از جایی که زیر روال فراخوانده شده است) ذخیره می‌شود و وقتی اجرای زیر روال به اتمام رسید کنترل برنامه مجدداً به آدرس برگشت باز می‌گردد و اجرای ادامه کد را از سر می‌گیرد.

می‌توانیم پارامترهایی به زیر روال انتقال دهیم اما آن‌ها به صورت داخلی در خود کد به‌کار گرفته می‌شوند. مجدداً به Module1 رفته و کد زیر را به آن اضافه می‌کنیم:

```
Sub Display(target)
    MsgBox target
End Sub
```

توجه کنید که این زیر روال دارای یک پارامتر شناسه برای متغیر target است. این امر به این خاطر است که طبق قرار زیر روال از رویه دیگری فراخوانده شده و متغیری به آن انتقال می‌یابد.
در Module1، خطی برای جداسازی زیر روال جدید ترسیم می‌شود و زیر روالی که نوشته می‌شود به طور خودکار به لیست بازشوی گوشه بالا- چپ پنجره اضافه می‌شود. روی شئ This Workbook کلیک کنید و به سراغ مثال فصل ۱ (Hello World) بروید. در رویداد Workbook_Newsheet، کد زیر را اضافه کنید:

```
Private Sub Workbook_NewSheet(ByVal Sh As Object)
'MsgBox "Hello World"
x = Multiply(3, 5)
MsgBox x
Call Display("my subroutine")
End Sub
```

حال روی کاربرگ Excel کلیک کرده و از منو، Insert | Worksheet را انتخاب کنید. کادر پیغامی ظاهر می‌شود که عدد 15 را نشان می‌دهد و بعد از آن هم کادر پیغام دیگری باز می‌شود که نشان‌دهنده "my subroutine" است.
دستور Call، زیر روال شما را فرامی‌خواند و پارامترهای مورد نیاز را به آن انتقال می‌دهد. سپس کد موجود در آن را اجرا می‌کند و به سراغ دستور بعدی که پس از دستور Call قرار دارد می‌رود. در این مثال، رشته متنی"my subroutine" را به متغیری که target نامیده می‌شود، انتقال می‌دهد.
اگر زیر روالی که نوشته‌اید از پارامترها (آرگومان‌ها) استفاده نمی‌کند، می‌توانید با انتخاب Run | Run Sub | UserForm از منوی VBE (ویراستار ویژوال بیسیک) و فشار دادن کلید F5 یا کلیک روی آیکن گرافیکی Run در نوار ابزار، آن‌را از صفحه کدنویسی اجرا کنید. مکان‌نما باید روی زیر روالی باشد که می‌خواهید آن را اجرا کنید. این شیوه، روش مفید و مطمئنی برای آزمایش کدی که نوشته‌اید و کشف خطاهای احتمالی موجود در آن است.
با توجه به این‌که تقسیم کردن یک مسأله بزرگ به چندین بخش کوچک باعث تسهیل در انجام عملیات می‌شود، زیر روال‌ها نیز در تجزیه پروژه‌های بزرگ به اجزای کوچک‌تر و ساده شدن کار، بسیار

سودمند هستند. اگر نخواهید از زیر روال‌ها استفاده کنید باید یک کد بزرگ بنویسید که علاوه بر پیچیدگی، پتانسیل بروز مشکلات و خطاهای متعدد را هم دارا باشد.

نوشتن یک تابع ساده

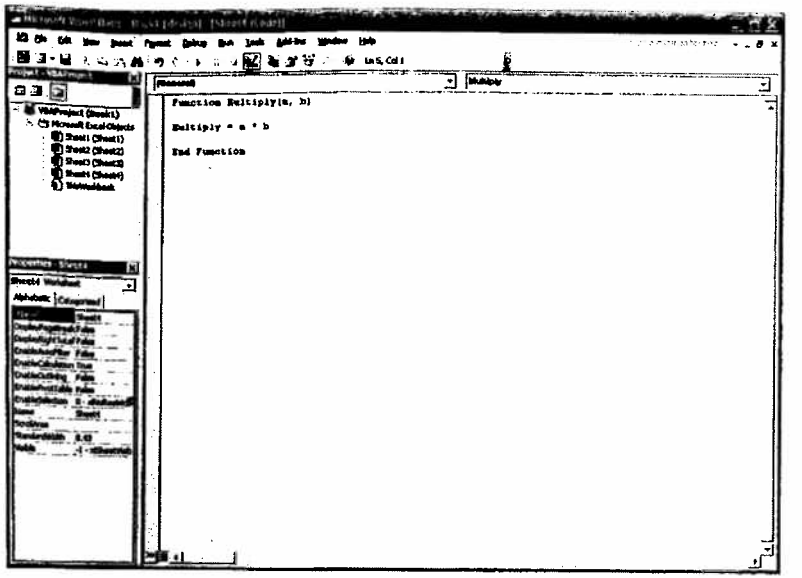
هدف این تمرین، ایجاد تابعی است که دو عدد را قبول می‌کند، آن‌ها را در یکدیگر ضرب می‌کند و نتیجه را برمی‌گرداند. این تابع را Multiply می‌نامیم. جدول زیر، عملیات اصلی ریاضی را که در هنگام نوشتن توابع و زیرروال‌ها در VBA استفاده می‌کنیم، نشان می‌دهد.

Add	+
Subtract	-
Multiply	*
Divide	/

کد این تابع به شرح زیر است:

```
Function Multiply(a, b)

    Multiply = a * b

End Function
```

این کد شبیه تصویر ۱-۳ است. مثل زیر روال، تابع نیز دارای یک خط شروع (هدر) و یک خط پایان (پانویس) است.
هدر، دو پارامتر (a, b) را با نشان دادن آن‌ها در داخل پرانتز پس از عنوان تابع، معرفی می‌کند. دو آرگومان با کاما از هم جدا شده‌اند. این آرگومان‌ها، دو عددی را که قرار است در هم ضرب شوند، نشان می‌دهد.



تصویر ۱-۳ ایجاد تابع ساده Multiply

نام تابع، Multiply است و این نام به عنوان یک متغیر برای برگرداندن پاسخ استفاده می‌شود. این تنها روش برگرداندن پاسخ به روالی است که تابع را فراخوانده است. توجه کنید که از این پس نام تابع در لیست بازشوی گوشه سمت راست در پنجره کد ظاهر می‌شود. دلیل این امر آن است که از این پس، این تابع هم در کد VBA و هم در Excel یک تابع رسمی محسوب می‌شود.

می‌توانیم از این تابع به دو صورت استفاده کنیم:

با فراخوانی مستقیم آن به صورت یک تابع در صفحه گسترده، یا استفاده از آن در کد VBA.

روی صفحه گسترده کلیک کرده و در یک سلول، دستور =multiply(3,4) را تایپ کنید. پاسخ 12 در آن سلول ظاهر خواهد شد. می‌بینید که استفاده از فرمول‌ها در Excel چقدر ساده است.

اکنون به سراغ روش دوم استفاده از تابع یعنی: فراخوانی آن از طریق کد VBA می‌رویم. برای سادگی کار در ادامه، از همان رویدادی که مثال Hello World را فراخواندیم، تابع جدیدی فراخوانیم.

روی شیء ThisWorkbook کلیک کنید و به مثال اول در فصل ۱ (Hello World) برگردید. دستور "Hello World" را با قرار دادن یک کاراکتر نقل قول تکی (') قبل از آن، به یک عبارت توضیحی تبدیل کرده و سپس کد زیر را در ادامه آن تایپ کنید تا به صورت زیر درآید:

```
Private Sub Workbook_NewSheet(ByVal Sh As Object)
    'MsgBox "Hello World"
    x = Multiply(3, 5)
    MsgBox x
End Sub
```

توجه کنید که وقتی کلمه Multiply را تایپ می‌کنیم و پرانتزها باز می‌شوند، VBA به صورت خودکار، پارامترهای موردنظر برای دریافت را، نمایش می‌دهد. با درج تابع در کد، در واقع با استفاده از پارامترهای 3 و 5 به جای پارامترهای a و b، تابع را فرامی‌خوانیم. نتیجه به متغیر x بر گردانده می‌شود. اکنون روی کاربرگ کلیک کرده و از منو، Insert | Worksheet را انتخاب کنید. یک کادر پیغام با مقدار 15 که پاسخ است، ظاهر می‌شود.

توابع و زیر روال‌های عمومی و خصوصی

VBA به شما اجازه می‌دهد تا با استفاده از کلمات کلیدی Private و Public، توابع یا زیر روال‌ها را به صورت عمومی یا خصوصی تعریف کنید. برای مثال:

```
Private Sub PrivateSub()
End Sub
```

هر زیر روال یا تابعی که ایجاد می‌کنید به صورت پیش فرض، عمومی است. این به معنای آن است که می‌توانید آن‌ها را در تمام ماژول‌های موجود در برنامه خود به‌کار بگیرید و کاربران کاربرگ نیز می‌توانند با انتخاب Tools | Macro | Macros به زیر روال‌ها به عنوان ماکرو دسترسی پیدا کنند. کاربران هم‌چنین قادرند در صورتی که روی آیکن گرافیکی فرمول که روی نوار ابزار صفحه گسترده قرار دارد کلیک کنند به توابع عمومی دسترسی پیدا کنند و آن‌ها را به صورت فهرست شده در بخش Custom ببینند.

یک استثنا برای این مورد وجود دارد و آن هم UserForms است. همان‌طور که در فصل ۹ خواهید دید UserForm ها، ماژول‌های خاص خود را دارند و نشان‌دهنده فرم‌های محاوره‌ای هستند. یک تابع یا زیر روال عمومی روی یک UserForm را می‌توان از داخل کد با ارجاع به شیء فرم (مثلاً

(UserForm1.MysubRoutine) فراخواند اما آن تابع یا زیر روال در لیست تابع یا ماکروی صفحه گسترده ظاهر نمی‌شود و هر تابع نوشته شده روی یک UserForm را نمی‌توان روی خود صفحه گسترده استفاده کرد.

هم‌چنین اگر یک برنامه الحاقی در Excel ایجاد کنیم، رویه‌های عمومی در آن برنامه الحاقی را می‌توان با ارجاع به شیءadd-in در دیگر ماژول‌ها هم به‌کار برد. این امر حتی اگر برای آن برنامه الحاقی کلمه عبور (password) تعیین کنیم نیز ممکن است. این قضیه در مواقعی مفید است که بخواهیم یک برنامه الحاقی در رویه‌ها بنویسیم که توسط دیگر برنامه نویسان VBA نیز قابل استفاده باشد بدون این‌که بتوانند کد به‌کار گرفته شده را کشف کنند. البته این مورد جزو نقایص کار نیز محسوب می‌شود یعنی اگر نخواهیم افراد دیگری که به رویه‌های عمومی ما دسترسی دارند از این برنامه‌های الحاقی استفاده کنند نمی‌توانیم جلوی آن‌ها را بگیریم. اگر برنامه الحاقی بارگذاری شود، رویه‌های عمومی موجود در آن برنامه در اختیار تمام ماژول‌ها قرار می‌گیرند.

با استفاده از تعریف‌های خصوصی می‌توانید رویه‌هایی داشته باشید که همنام هستند اما در ماژول‌های متفاوتی قرار دارند. چنین رویه‌هایی برای ماژول‌های خود خصوصی محسوب می‌شوند و توسط دیگر ماژول‌ها قابل مشاهده نیستند. از همه مهم‌تر این‌که، کاربر صفحه گسترده نمی‌تواند آن‌ها را ببیند یا اجرا کند.

انواع داده برای آرگومان[1]

وقتی آرگومانی برای یک رویه مشخص می‌کنیم، به صورت پیش‌فرض از نوع Variant خواهد بود که نوع متغیر پیش‌فرض برای تمام متغیرها در VBA است. هم‌چنین می‌توانیم بر مبنای انواع داده‌ای که در فصل قبل دیدیم، پارامترها را از انواع دیگر نیز تعریف کنیم.

مزیت تعریف متغیرها با انواع مختلف داده‌ای این است که آن‌ها باعث منظم شدن اطلاعات موجود در رویه می‌شوند. اگر نوع متغیر را مشخص نکرده و از نوع Variant استفاده کنیم، رویه شما هر چیزی (چه عدد و چه رشته) را قبول می‌کند. اگر ما در رویه، انتظار دریافت یک رشته را داشته باشیم و به جای آن یک عدد وارد شود (یا برعکس)، چنین موردی باعث بروز نتایج نامطلوبی خواهد شد. اگر پارامتری را به عنوان رشته (string) مشخص کنیم در صورتی‌که یک مقدار رشته‌ای به آن اختصاص نیابد پیغام خطا ارسال خواهد شد:

```
Function (Target as String)
```

1 - Argument

این امر در هنگام نوشتن یک تابع اختصاصی صفحه گسترده بسیار سودمند است. وقتی کاربران، تابع شما را در یک سلول وارد می‌کنند باید پارامترها را بر طبق نوع داده مشخص شده وارد کنند. اگر برای پارامترها، نوع رشته‌ای مشخص شده باشد آن‌ها باید مقدار ورودی را در بین علامت نقل قول (" ") قرار دهند یا این‌که به سلولی ارجاع کنند که حاوی یک مقدار متنی است.

آرگومان‌های اختیاری

با استفاده از کلمه کلیدی Optional می‌توانیم آرگومان‌های خاصی را به صورت اختیاری در آوریم:

```
Function MyFunction (Target as String, Optional Flag as Integer)
```

در این مثال، کاربر باید پارامتر Target را به عنوان یک رشته مشخص کند، اما دو طرف پارامتر Flag،کاراکترهای [] قرار می‌گیرند و نیازی به مشخص کردن آن‌ها نیست. پارامترهای اختیاری را باید بعد از پارامترهای اجباری قرار داد.

انتقال آرگومان‌ها از طریق مقدار

با استفاده از کلمات کلیدی ByVal و ByRef نیز می‌توانیم چگونگی انتقال پارامترها به رویه‌ها را مشخص کنیم:

```
Function MyFunction (ByVal Target as String)
```

کلمه کلیدی ByVal ما را مطمئن می‌سازد که پارامترها به جای ارجاع به مقدار یک متغیر، توسط مقدار خاص خودشان انتقال می‌یابند. انتقال یک مقدار توسط ارجاع می‌تواند به سادگی منجر به نفوذ اشکال‌ها و خطاهایی در کد شئ شود. برای تشریح این قضیه فرض کنید با استفاده از نام متغیر، انتقال توسط ارجاع را انجام می‌دهید. مقدار آن متغیر می‌تواند توسط رویه‌ای که متغیر به آن انتقال می‌یابد تغییر کند. برای مثال:

```
x = 100
z = Adjust (x)
Function Adjust (ByRef Target as Integer)
```

متغیر x با توجه به آن‌چه در تابع Adjust رخ می‌دهد تغییر می‌کند. با این حال اگر انتقال توسط مقدار انجام شود، تنها یک کپی از متغیر به رویه انتقال می‌یابد:

```
x = 100
z = Adjust (x)
```

```
Function Adjust (ByVal Target as Integer)
```

اگر رویه باعث تغییر این مقدار شود، تغییر حاصل تنها روی کپی اثر می‌گذارد و خود متغیر را دچار تغییر نمی‌کند و متغیر x همیشه حاوی مقدار 100 خواهد بود.

معمولاً ما انتظار نداریم که تابع، یک آرگومان را تغییر دهد، اما چنین امری ممکن است رخ دهد و می‌تواند منجر به بروز خطاهایی در کد شود که یافتن آن‌ها بسیار مشکل است. برای پرهیز از بروز چنین خطاهایی بهتر است از اعلان ByVal استفاده کنیم.

فصل چهارم

اصول برنامه‌نویسی : تصمیم‌ها و حلقه‌سازی

در هنگام نوشتن برنامه‌ها، درک چگونگی اتخاذ تصمیم‌ها و اجرای حلقه‌ها توسط برنامه از اهمیت زیادی برخوردار است. حلقه سازی، فرآیند انجام مجموعه‌ای مشابه از دستورالعمل‌ها تا زمان برقرار بودن شرایط خاص است.

همه با تصمیم و تصمیم‌گیری آشنا هستیم و روزانه چندین بار آن را تجربه می‌کنیم. برای مثال وقتی صبح بیدار می‌شوید تصمیم می‌گیرید چه پیراهنی بپوشید. این تصمیم را بر اساس موارد متعددی مثل اوضاع جوی و کاری که قرار است در طی روز انجام شود می‌گیریم.

برنامه‌ها نیز بر مبنای پارامترهایی که به آن‌ها دسترسی دارند، تصمیم می‌گیرند. برای مثال اگر یک برنامه برای بارگذاری یک فایل کارپوشه تلاش کند و فایل پیدا نشود، چه تصمیمی باید برای انجام مراحل بعدی کار گرفته شود؟ آیا برنامه باید تنها یک پیغام خطا نمایش دهد و اجرای آن متوقف شود یا این‌که باید کمی هوش به خرج دهد و به کاربر اعلام کند که فایل مذکور وجود ندارد و او باید عملیات دیگری را اجرا کند؟

هوش مصنوعی چیزی است که در دورههای کامپیوتر زیاد درباره آن صحبت می‌شود. اگر بتوانیم کاری کنیم که برنامه‌ها، تصمیم‌گیری کنند در برنامه خود، هوش مصنوعی به‌وجود آورده‌ایم. در واقع این هوش ماست که در کد قرار می‌گیرد و به برنامه می‌گوید در صورت وقوع موارد مختلف چه واکنشی نشان دهد.

حلقه سازی نیز چیزی است که شما هر روز بدون فکر درباره آن، آن را انجام می‌دهید. وقتی غذا می‌خورید، روال مشابه برداشتن غذا از ظرف و گذاشتن آن در دهان را انجام می‌دهید. برنامه‌های کامپیوتری چندین مرتبه روی یک قطعه کد اجرا می‌شوند تا زمانی‌که شرط خاص حلقه برقرار باشد.

تصمیم‌ها

برنامه‌ها، به جز آن‌هایی که خیلی ساده‌اند معمولاً باید بر طبق داده‌های حاصل یا ورودی کاربر، تصمیم‌هایی بگیرند. تصمیم‌گیری، یکی از مهم‌ترین حوزه‌های برنامه‌نویسی است چون مشخص می‌کند که در صورت وقوع رویدادهای مختلف، چه اتفاقی رخ می‌دهد.

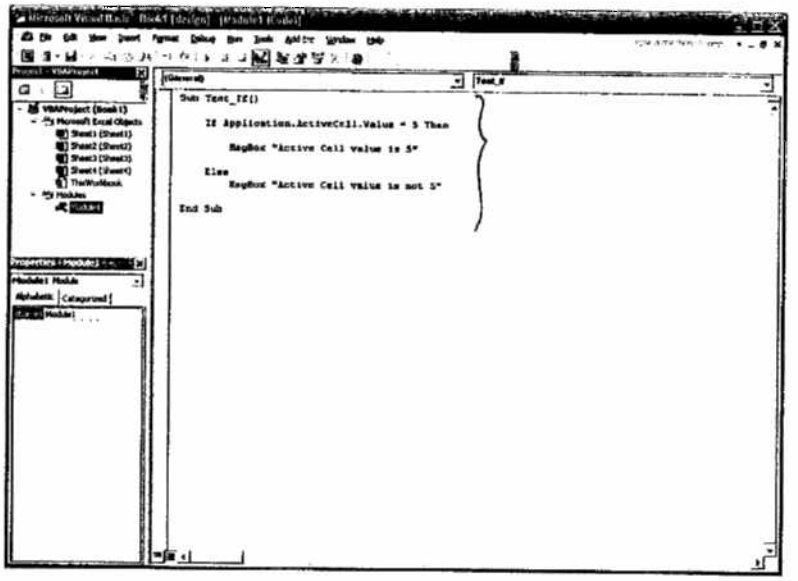
یک مثال خوب از یک تصمیم متعارف برنامه نویسی، شرط If (اگر)است:

اگر شرطی برقرار باشد(If) آن‌گاه (Then) عملیات I انجام شود و در غیر این صورت (Else) عملیات II اجرا شود. مثال این شرط در زندگی روزمره این است که" اگر (If) باران ببارد آن‌گاه (Then) باید چتر با خود ببریم در غیر این صورت (Else) (یعنی اگر شرط برقرار نباشد و باران نبارد) باید عینک آفتابی ببریم."

در این‌جا مثال ساده‌ای می‌آوریم تا نشان دهیم دستور شرطی If.. Then.. Else چگونه کار می‌کند و چطور نتایج متفاوت حاصل می‌شود. این کد را در ماژولی که در فصل ۲ ایجاد کردیم وارد کنید. تصویر ۱-۴ را برای این‌که ببینید پنجره کد شما به چه شکلی است مشاهده کنید:

```
Sub test_if( )
If Application.ActiveCell = 5 Then

   MsgBox "Cell is 5"

Else

   Msgbox "Cell is not 5"

End If
End sub
```

این مثال با خط کدApplication.ActiveCell به سلول فعال موجود در شیٔ برنامه کاربردی Excel اشاره می‌کند. این سلول همان است که مکان‌نما در آن قرار دارد. روی کاربرگ کلیک کرده و عدد 5 را درسلول فعلی وارد کنید. مطمئن شوید که مکان‌نما در آن‌جا قرار دارد. به پنجره کد برگردید و F5 را فشار دهید. یک کادر پیغام دریافت خواهید کرد که نشان می‌دهد آن سلول برابر 5 است. حال مجدداً به صفحه گسترده برگردید و مقدار سلول را تغییر دهید یا مقدار آن را حذف کنید. مجدداً ماکرو را اجرا کنید. آن‌گاه پیغامی دریافت خواهید کرد که می‌گوید: "مقدار سلول، 5 نیست".



تصویر ۱-۴ کدنویسی برای دستور شرطی If

در تصویر ۱-۴ توجه کنید که هدف من در کدنویسی، جداسازی قسمت‌های اصلی دستور شرطی بوده است که باعث ساده‌تر شدن خواندن کد و اشکال زدایی آن می‌شود. می‌توانیم دستورIf را به صورت تو در تو بنویسیم یعنی یک دستور If را در یک دستور If دیگر قرار دهیم. این شیوه را در برنامه‌های پیچیده، زیاد به کار می‌بریم و این‌که بتوانیم ببینیم دستورIf از کجا آغاز و در کجا خاتمه می‌یابد.

عبارت End..If نشان می‌دهد که دستورات شرطی If در کجا خاتمه می‌یابند یا این‌که می‌توانیم کل دستور If را در یک خط قرار دهیم که در آن صورت نیازی به عبارت End..If نخواهد بود:

```
If Application.ActiveCell = 5 Then MsgBox "Cell is 5"
```

اگر قرار باشد چندین دستورالعمل اجرا شوند، می‌توانید دستورها را در یک خط بنویسید و آن‌ها را با ویرگول از هم جدا کنید. البته خواندن چنین کدی بسیار مشکل است و به سختی می‌توانیم آن را عیب‌یابی کنیم. اپراتورهای شرطی که می‌توانیم به‌کار ببریم به این شرح هستند:

جدول ۱-۴ اپراتورهای شرطی

عملگر	مفهوم عملگر
=	یعنی هر دو عدد با هر دو مقدار با هم مساوی
<	یعنی عدد اول از عدد دوم کوچک‌تر است.
>	یعنی عدد اول از عدد دوم بزرگ‌تر است.
<=	یعنی عدد اول کوچک‌تر یا مساوی با عدد دوم است.
>=	یعنی عدد اول بزرگ‌تر یا مساوی با عدد دوم است.
<>	یعنی دو عدد با دو مقدار با هم مساوی نباشند.

یک عبارت مثل x=1 ، یک مقدار بولین تلقی می‌شود که مقدار آن، True یا False (صحیح یا غلط) و یا Zero و Non_Zero (صفر و غیر صفر) است. این به معنای آن است که شما همیشه مجبور نیستید از یک اپراتور استفاده کنید. اگر فقط علاقه‌مند هستید بدانید که آیا یک سلول، مقدار غیر صفر در خود دارد یا خیر، می‌توانید از کد زیر استفاده کنید:

```
If Application.ActiveCell Then MsgBox "cell has a value"
```

دستورهای شرطی چندگانه

در دستورهای قبلی فقط از یک دستور شرطی، شکل ...If x=1 Then استفاده کردیم. شما می‌توانید با استفاده از یک اپراتور منطقی از چندین دستور شرطی استفاده کنید. برای اطلاعات بیشتر درباره اپراتورهای منطقی به فصل ۶ مراجعه کنید.

دستورهای شرطی چندگانه، ساده هستند و تقریباً شبیه عبارت‌های انگلیسی ساده عمل می‌کنند. آن‌ها از اپراتورهای And (و) و Or (یا) استفاده می‌کنند.

اگر قرار باشد در دستور شرطی، دو شرط را کنترل کنیم می‌توانیم دستور If را به صورت زیر بنویسیم:

```
If x = 1 And y > 5 Then

   MsgBox "x=1 and y>5"

End if
```


کادر پیغام در صورتی نمایش داده می‌شود که هر دو شرط (x=1 و y<5) برقرار باشند. برای مثال اگر x>1 اما y برابر 4 باشد کادر پیغام به نمایش در نخواهد آمد. به صورت مشابه می‌توانیم از این دستور نیز استفاده کنیم:

```
If x = 1 Or y > 5 Then

MsgBox "x=1 or y>5"

End If
```

در مثال فوق، کادر پیغام در صورت برقرار بودن حتی یکی از شرط‌ها هم نمایش داده می‌شود. برای مثال اگر x=1 یا y>5 باشد کادر پیغام به نمایش درمی‌آید. بنابراین x می‌تواند صفر باشد و y می‌تواند 6 باشد. یا x می‌تواند 1 باشد و y برابر 4 باشد و در هر دو مورد هم کادر پیغام به نمایش درخواهد آمد.

می‌توانیم چندین And و Or را در عبارت شرطی خود به کار ببریم اما اگر از 3 تا بیشتر باشند باعث پیچیدگی زیاد دستور شرطی می‌شوند.

دستور شرطی Select Case (انتخاب یک گزینه)

دستور دیگری که برای پردازش شرط‌ها در VBA وجود دارد دستور Select Case است. اگر یک متغیر داشته باشیم و بخواهیم بر مبنای مقدار آن متغیر، عملیات متفاوتی انجام شود می‌توانیم از یک سری دستور If به شرح زیر استفاده کنیم:

```
If x=1 then MsgBox "x=1"
If x=2 then MsgBox "x=2"
If x=3 then MsgBox "x=3"
```

با وجود این در مثال فوق می‌توانیم با استفاده از دستور Select Case، کد را بسیار ساده‌تر و خواناتر کنیم:

```
x = 23
Select Case (x)
      Case (1)
        MsgBox "x=1"
      Case (23)
        MsgBox "x=23"
End Select
```

مفهوم دستور Select Case بسیار ساده است:یک متغیر خاص را در نظر می‌گیرید و بر طبق مقدار موجود در آن واکنش نشان می‌دهید. عبارت Select Case (x)، متغیری را که قرار است انتخاب شود با عنوان x تعریف می‌کند و شروع بلوک کدنویسی برای این رویه خواهد بود.(1) Case، در صورتی فعال می‌شود که مقدار x برابر 1 باشد و کادر پیغام "x=1" را نشان خواهد داد. (23) Case در صورتی که مقدار x برابر 23 باشد، فعال می‌شود و کادر پیغام "x=23" را نشان خواهد داد.

در عبارت Case می‌توانیم از عبارات To و Is نیز استفاده کنیم:

```
Function Test_Case (Grade)
    Select Case Grade
        Case 1
            MsgBox "Grade 1"
        Case 2,3
            MsgBox "Grade 2 or 3"
        Case 4 to 6
            MsgBox "Grade 4,5 or 6"
        Case Is > 8
            MsgBox "Grade is above 8"
        Case Else
            MsgBox "Grade not in conditional statements"
    End Select
End Function
```

حلقه‌سازی

بدون حلقه سازی، برنامه‌ها بسیار خسته کننده و دردسر ساز خواهند بود. حلقه‌سازی اجازه می‌دهد تا در زمان برقرار بودن یک شرط یا رسیدن به یک مقدار مشخص، یک بلوک کد اجرا شود. برای مثال فرض کنید که می‌خواهید اعداد 1 تا 5 را نمایش دهید و می‌توانید برنامه را به صورت زیر بنویسید:

```
MsgBox "1"
MsgBox "2"
MsgBox "3"
MsgBox "4"
MsgBox "5"
```

این برنامه جواب می‌دهد اما کارآمد نیست و از کارآیی VBA نیز به خوبی استفاده نمی‌کند. اگر بخواهید اعداد بیشتری را به کمک این کد نمایش دهید مجبور خواهید بود کد طولانی‌تری بنویسید، مثلاً اگر بخواهید اعداد1 تا 1000 را نمایش دهید باید 995 خط دیگر را به کد فوق اضافه کنید!

حلقه FOR..Next

با استفاده از دستور حلقه سازی FOR..Next می‌توانیم کد فوق را ساده‌تر بنویسیم:

```
For n = 1 to 5
        MsgBox  n
Next n
```

کادر پیغام 5 مرتبه ظاهر خواهد شد و مقادیر n را که از 1 تا 5 است، نشان می‌دهد.

متغیر به کار گرفته شده، هر چیزی می‌تواند باشد و هر چند که من از متغیر n استفاده کرده‌ام اما می‌توانید حتی از کلمه‌ای مثل num هم استفاده کنید با این شرط که تا آخر حلقه، بدون تغییر باقی بماند. نمی‌توانیم از دستور For n = 1 to 5 استفاده کنیم و سپس سعی کنیم تا از شاخصی مانند m استفاده نماییم بلکه باید حتماً n را به کار ببریم. هم‌چنین نباید از یکی از کلمات کلیدی و ذخیره شده VBA به عنوان نام متغیر استفاده کنیم. دستورات به کار گرفته شده در بین For و Next می‌توانند به هر تعدادی باشند و حتی می‌توانیم در بین آن‌ها، توابع و زیرروال‌ها را نیز فرا بخوانیم. مقادیر شروع و خاتمه در حلقه FOR..Next هم می‌توانند متفاوت باشند و اجباری نیست که حتماً با 1 شروع و با 5 خاتمه یابند.

گزینه Step، کارآیی بیشتری را برای حلقه FOR..Next ایجاد می‌کند. حتماً متوجه شده‌اید که قرار نیست همیشه متغیر n در هر بار چرخش حلقه 1 واحد افزایش یابد (هر چند که گام یا Step حلقه به صورت پیش‌فرض برابر با 1 است). می‌توانید با استفاده از گزینه Step، گام حلقه را بالا ببرید. Step به ما اجازه می‌دهد تا اندازه افزایش و هم‌چنین جهت حرکت حلقه (مثبت یا منفی) را مشخص کنیم. به مثال زیر دقت کنید:

```
For n = 3 to 12 step 3

        MsgBox  n

Next n
```

خروجی کد زیر3، 6، 9، 12 خواهد بود چون گام حلقه 3 است.

برای این که ببینید Step چطور به صورت معکوس عمل می‌کند به مثال زیر دقت کنید:

```
For n = 10 to 1 step -1
        MsgBox  n
Next n
```

نتیجه کد فوق، 10 ، 9 ، 8 ، 7 ، 6 ، 5 ، 4 ، 3 ، 2 ، 1 خواهد بود.

حلقه‌های FOR..Next برای خواندن اعداد ستونی یا اعداد ردیفی بسیار مناسب هستند. می‌توانیم برای افزایش خودکار شماره ردیف در یک آدرس سلول از حلقه FOR..Next استفاده کنیم.

حلقه‌های FOR..Next می‌توانند درداخل یکدیگر به صورت تو در تو نیز به کار گرفته شوند. برای مثال اگر بخواهیم به مقدارموجود در یک صفحه گسترده دست پیدا کنیم می‌توانیم از یک حلقه FOR..Next برای رسیدن به ستون‌ها و از حلقه FOR..Next دوم برای رسیدن به ردیف‌ها استفاده کنیم.

در ذیل، یک مثال آورده‌ایم که حلقه‌هایی دارد تا مقادیر n و m تکرار شوند. حلقه m در داخل حلقه n قرار گرفته و این به معنای آن است که ابتدا مقدار نخست n اجرا می‌شود، سپس تمام مقادیر m اجرا می‌شوند. تورفتگی‌ها کمک می‌کنند که اگر بعداً به سراغ کد بیایید، روال کار را گم نکنید:

```
Sub test_loop ( )
      For n = 1 To 4
            For m = 1 To 5

                  MsgBox "n = "  & n
                  MsgBox "m = "  & m

            Next m
      Next n
End Sub
```

حلقه For Each

حلقه For Each بسیار شبیه حلقه FOR..Next است و تنها تفاوت آن‌ها در این است که حلقه For Each روی مجموعه‌ها یا آرایه‌ها عمل می‌کند و به شما کمک می‌کند هر کدام از اعضا مجموعه را به صورت تک تک مرور کنید. در این حلقه از شاخص (مثل n در مثال قبلی) استفاده نمی‌کنیم چون خودش به صورت خودکار، تمام اجزامجموعه را مرور می‌کند. این حلقه زمانی سودمند است که بخواهیم در یک مجموعه به دنبال مورد خاصی بگردیم و سپس آن را از مجموعه حذف کنیم چون موقعیت حلقه در مجموعه، پس از حذف، بدون تغییر باقی می‌ماند. اما اگر از حلقه FOR..Next که شاخص دارد استفاده کنید و یک شئ را حذف کنید، شاخص، یک واحد افزایش می‌یابد، حلقه در چرخه‌ای بی‌پایان گرفتار می‌شود و پیغام خطا صادر می‌شود.

مثال زیر با استفاده از یک حلقه For Each، نام کاربرگ‌ها را نشان می‌دهد:

```
Sub ShowName ( )
      Dim oWSheet As Worksheet
      For Each oWSheet In WorkSheets
            MsgBox oWSheet.Name
      Next oWSheet
End Sub
```


حلقه Do Until

حلقه Do Until تا زمان برقرار بودن یک شرط خاص، تکرار را ادامه می‌دهد. معمولاً این شرط، رسیدن یک متغیر به مقداری خاص است. وقتی شرط برقرار شود، حلقه متوقف می‌شود و برنامه به اجرای دستورالعمل بعد از حلقه می‌پردازد. هم‌چنین می‌توانیم از دستور While استفاده کنیم به طوری‌که تا زمان برقرار بودن یک شرط، کد حلقه اجرا شود. در این جا یک مثال ساده ذکر می‌کنیم:

```
Sub test_do ( )
x = 0
Do Until x = 100
      x = x + 1
Loop
MsgBox  x
End Sub
```

ابتدا مقدار متغیر x را برابر صفر در نظر می‌گیریم. سپس شرط x = 100 را به عنوان ملاک توقف حلقه Do در نظر می‌گیریم. متغیر (x) در هر مرتبه اجرای حلقه، یک واحد افزایش می‌یابد پس حلقه تا زمان رسیدن به شرط x=100 ، 100 بار اجرا می‌شود. زمانی که مقدار x برابر با 100 شود، کادر پیغامی به نمایش درمی‌آید که نشان می‌دهد مقدار x برابر با 100 است.

حلقه While..Wend

در آخر، حلقه While..Wend را مطرح می‌کنیم. اجرای این حلقه تا زمانی‌که یک شرط خاص برقرار باشد،ادامه می‌یابد و به محض آن‌که شرط برقرار نباشد، اجرای حلقه متوقف می‌شود. در این جا، یک مثال ساده ذکر می‌کنیم که بسیار مشابه حلقه قبلی (Do Until) است:

```
Sub test_do ( )
x = 0
While x < 50
    x = x + 1
Wend
MsgBox x
End Sub
```

باز هم متغیر x برابر صفر تنظیم می‌شود. شرط، کمتر بودن متغیرx از 50 است و x در هر بار اجرای

حلقه، یک واحد افزایش می‌یابد. وقتی x=50 باشد، دیگر متغیر x کمتر از 50 نخواهد بود و یک کادر پیغام نمایش داده می‌شود که نشان می‌دهد مقدار x برابر 50 است.

خروج زود هنگام (Exit) از حلقه‌ها

تحت بعضی شرایط خاص ممکن است بخواهیم پیش از پایان روال معمولی حلقه و در زمانی که هنوز شرط حلقه برقرار است از رویه خود خارج شویم. مثلاً فرض کنید که در یک آرایه به دنبال یک رشته خاص کاراکترها می‌گردیم. ممکن است در آن آرایه، ۲۵ مورد وجود داشته باشد اما وقتی رویه،آن‌چه را که به دنبالش می‌گردد، پیدا کند دیگر نیازی نیست تا ادامه آرایه را مرور کند. حال فرض کنید که آرایه‌ای شامل هزاران رکورد دارید که باید در بین آن‌ها مرور کنید و در این حالت اگر در ابتدای کار، مورد جستجو پیدا شود دیگر نیازی نیست تا مدت زمان زیادی صرف شود و بیهوده تا آخر آرایه را مرور کنیم. در مورد یک حلقه FOR..Next، مقدار شاخص نیز حفظ می‌شود و این به معنای آن است که می‌توانیم از این شاخص برای تعیین موقعیت جایی که شرط در آن‌جا تحقق می‌یابد استفاده کنیم. در این‌جا یک مثال ساده ذکر می‌کنیم:

```
Sub test_exit( )

    For x = 1 To 100
        If x = 50 Then
            Exit For
        End If
    Next x
      MsgBox x
End Sub
```

می‌توانیم با استفاده از دستور Exit For، زودتر از موعد از دستور For..Next یا For..Each خارج شویم. برای خروج زود هنگام از دستور Do Until از دستور Exit Do استفاده می‌کنیم.

فصل پنجم

رشته‌ها، توابع و کادرهای پیغام

این فصل به بررسی چگونگی کار روی کادرهای متن، چگونگی استفاده از توابع داخلی VBA و چگونگی طراحی کادرهای پیغام حرفه‌ای می‌پردازد.

رشته‌ها

اگر قبلاً زیاد از Excel استفاده کرده باشید می‌دانید که رشته، در واقع مجموعه‌ای از کاراکترها است. رشته‌ها محدود به حروف الفبا نیستند بلکه هر کاراکتری در دامنه ۰ تا ۲۵۵ را هم در برمی‌گیرد. البته این امر شامل کاراکترهای کنترلی و کاراکترهای الفبایی- عددی نیز می‌شود. بر طبق زبان مورد استفاده برای کد نویسی، رشته‌ها ممکن است متفاوت باشند اما نهایتاً تمام رشته‌ها حداکثر ۲۵۶ کاراکتر را شامل می‌شوند. رشته‌ها برای نمایش پیغام به کاربر یا ساخت یک متن تشریحی به کار می‌روند. "Richard" و "1234" هر دو مثالی از رشته‌ها هستند.

VBA ، چند تابع برای الحاق (اتصال) رشته‌ها به یکدیگر، حذف قسمتی از رشته‌ها، جستجوی کاراکترهای خاص و بزرگ و کوچک کردن کاراکترها دارد. برای مثال اگر رشته "Your answer" و "is wrong" را داشته باشیم می‌توانیم آن‌ها را به صورت زیر الحاق کنیم:

"Your answer is wrong"

هم‌چنین می‌توانیم از تابعی استفاده کنیم تا کل رشته با کاراکترهای بزرگ تایپ شود و به صورت زیر در آید. "YOUR ANSWER IS WRONG" یا می‌توانیم کلمه یا کاراکترهای خاصی را در رشته جستجو کنیم.

الحاق

الحاق، چگونگی اتصال رشته‌ها به یکدیگر است که معمولاً با استفاده از علامت & انجام می‌پذیرد. وقتی می‌خواهیم پیغام‌هایی را برای کاربران نمایش دهیم این تابع بسیار مفید است. فرض کنید برنامه‌ای می‌نویسیم که تعداد کاربرگ‌های موجود در یک کارپوشه را نشان می‌دهد. برنامه، کاربرگ‌ها را

شمرده و عدد مربوطه را در یک متغیر ذخیره می‌کند. به سادگی می‌توانیم مقدار متغیر را به کاربر نشان بدهیم اما نکته مهم این است که این عدد چه معنایی برای کاربر دارد؟
در هنگام نوشتن نرم افزار، همیشه می‌خواهیم پیغام واضحی برای کاربر نمایش داده شود مثل:
"There are n worksheets in your workbook"

" n تعداد کاربرگ، در کتاب کاری شما وجود دارد".

می توانید این کار را با الحاق رشته "There are"، متغیر n (که حاوی تعداد کاربرگ‌هاست) و رشته "worksheets in your workbook" انجام دهید. هم‌چنین می‌توانید کدی را بنویسید تا در صورتی‌که مقدار n برابر با ۱ بود، رشته نخست را به صورت "There is" چاپ کند تا از لحاظ گرامری نیز مشکلی وجود نداشته باشد:

```
MsgBox "There are " & n & " worksheets within the workbook"
```

مثال ساده حلقه For...next را که در فصل ۴ مطرح شد، در نظر بگیرید. کد آن به صورت زیر است:

```
For n = 1 to 5
        MsgBox n
Next n
```

کادر پیغام، مقدار n را در هر مرتبه تکرار حلقه، نمایش می‌دهد. اما مشکل این‌جاست که این فقط یک کادر پیغام است که حاوی یک مقدار عددی است و این عدد هیچ مفهومی ندارد. با اضافه کردن یک رشته می‌توانیم پیغام را کاربر پسندتر کنیم.

```
For n = 1 to 5
        MsgBox "The value of n is " & n
Next n
```

حال آنچه در کادر پیغام به نمایش درمی‌آید به صورت زیر است:

"The value of n is 1"

این پیغام پنج مرتبه به نمایش درمی‌آید و هر بار مقدار n را هم نشان می‌دهد. دقت کنید که حتماً بعد از is یک جای خالی قرار دهید وگرنه به خاطر چسبیدن عدد به is شکل ظاهری پیغام زیبا نخواهد بود.
از لحاظ تعداد رشته‌ها و مقادیری که به این صورت می‌توانید به یکدیگر الحاق کنید محدودیتی وجود ندارد. توجه کنید که هر چند n یک مقدار عددی است اما VBA به طور خودکار آن را به یک رشته کاراکتری تبدیل می‌کند تا در عملیات الحاق شرکت کند.

تقسیم کردن رشته‌ها

ممکن است در کد فقط به بخشی از یک رشته نیاز داشته باشید. برای مثال فرض کنید در آغاز رشته، یک شماره مرجع دو قسمتی دارید که قرار است در جایی از برنامه به کار گرفته شود، اما تنها می‌خواهید نام موجود در آن را نشان دهید:

"12Richard"

برای جدا کردن قسمت نام، از دستور Mid استفاده می‌کنید:

```
x = Mid("12Richard",3)
```

این کد از کاراکتر سوم آغاز شده و تا انتهای رشته ادامه یافته و نتیجه را در متغیر x قرار می‌دهد. از این پس متغیر x حاوی رشته "Richard"خواهد بود. دستور Mid دارای پارامتر طول از نوع اختیاری است که توسط آن می‌توانیم طول رشته فرعی را تعیین کنیم. اگر مقداری برای این پارامتر مشخص نشود، پس از اجرای فرمان، تمام کاراکترها از نقطه شروع تا پایان رشته انتخاب می‌شوند.
توجه کنید که در تمام این توابع نیز می‌توانید از متغیری که حاوی یک مقدار رشته‌ای است، استفاده کنید:

```
temp = "12Richard"
x = Mid(temp,3)
```

از این دستور می‌توانیم برای استخراج قسمت عددی رشته فوق نیز استفاده کنیم:

```
x = Mid("12Richard",1,2)
```

این کد از کاراکتر اول آغاز شده و دو کاراکتر پس از آن را از رشته جدا کرده و آن‌ها را در متغیر x ذخیره می‌کند. از این پس متغیر x حاوی "12" خواهد بود. به این نکته توجه کنید که این مقدار یک رشته است و یک مقدار عددی محسوب نمی‌شود. البته VBA بسیار فراموش کار است و اگر بخواهید از این مقدار در محاسبات بعدی استفاده کنید نیازی به تبدیل آن به یک مقدار عددی نخواهید داشت.
اگر بخواهید دوباره آن‌را در یکی از سلول‌های صفحه گسترده به کار گیرید لازم است که آن‌را به یک مقدار عددی تبدیل کنید. این کار را با استفاده از تابع Val انجام دهید:

```
Dim iValue as Integer
iValue = Val("12")
```

از این پس متغیر iValue به جای آن‌که یک رشته باشد، یک مقدار عددی خواهد بود.

VBA شامل توابع رشته‌ای Right و Left نیز هست. از تابع Left می‌توان برای جدا کردن عدد 12 استفاده کرد:

```
x = Left("12Richard",2)
```

متغیر x حاوی مقدار 12 خواهد بود.

اگر از تابع Right استفاده کرده بودیم، مقدار rd در x قرار می‌گرفت.

```
x = Right("12Richard",2)
```

توابع Right و Leftهمان‌طور که از نامشان پیداست از دو طرف تابع، جداسازی را انجام می‌دهند.

تغییر ظاهر رشته‌ها

تابع Ucase، تمام کاراکترهای الفبایی موجود در رشته را به صورت حروف بزرگ در می‌آورد:

```
x = UCase("Richard")
```

متغیر x حاوی مقدار "RICHARD" خواهد بود.

تابع Lcase، تمام کاراکترهای الفبایی را به حروف کوچک تبدیل می‌کند:

```
x = LCase("Richard")
```

متغیر x حاوی مقدار "richard" خواهد بود.

در هر دوی این مثال‌ها، تمام کاراکترهای غیر حرفی مثل اعداد بدون تغییر باقی می‌مانند.

جستجوی رشته‌ها

تابعی وجود دارد که برای جستجوی یک رشته کاراکتری در یک رشته بزرگ تر به کار می‌رود. این تابع را Instr می‌نامیم. ساختار دستوری این تابع به خاطر این‌که دو شناسه اختیاری در آن استفاده می‌شود کمی پیچیده است:

```
InStr ([start, ] string1, string2[ , compare])
```

Start، یک پارامتر اختیاری است و نشان می‌دهد عملیات جستجو از کجای رشته باید آغاز شود. اگر مقداری برای آن ذکر نشود، عملیات از کاراکتر اول آغاز می‌شود. Start نباید حاوی مقدار پوچ باشد (null value) و اگر از Start استفاده شود حتماً باید از پارامتر Compare نیز استفاده کنیم.

String1، رشته‌ای است که عملیات جستجو در آن انجام می‌شود (برای مثال "Richard Shepherd") و String2، رشته‌ای است که جستجو می‌شود (برای مثال "shepherd").

Compare، تکنیک به کار گرفته شده برای مقایسه رشته‌هاست. مقادیر احتمالی آن عبارتند از: vbBinaryCompare و vbTextCompare.

به عبارت ساده این مقادیر مشخص می‌کنند که آیا عملیات جستجو نسبت به بزرگ و کوچک بودن حروف حساس باشد یا خیر.BinaryCompare ، از مقدار باینری واقعی استفاده می‌کند پس A معادل A است اما مساوی با a نیست. TextCompare نسبت به بزرگ و کوچک بودن حروف حساس نیست پس A معادل a است. مقدار پیش فرض برای Compare، کنترل در حالت باینری است که به معنای حساس بودن نسبت به بزرگ و کوچک بودن حروف است.

جدول ۱-۵ مقادیر تابع Instr

مقدار برگشتی تابع Instr	مفهوم
0	String1، طول صفر کاراکتر دارد.
Null	String1، پوچ است.
Start Value	String2، طول صفر کاراکتر دارد.
Null	String2، پوچ است.
0	String2 پیدا نشد.
Position	موقعیت String2 در String1
0	start بزرگ‌تر از طول String1 است.

جدول ۱-۵، مقادیری را که تابع Instr می‌سازد فهرست می‌کند. در این‌جا یک مثال ساده می‌آوریم:

```
x = Instr ("Richard Shepherd" , "Shepherd")
```

جواب این کد، عدد 9 است.

توجه کنید که حالت پیش فرض مقایسه، نسبت به بزرگ و کوچک بودن حروف حساس است:

```
x = Instr ("Richard Shepherd" , "shepherd")
```

جواب این کد صفر است چون رشته "shepherd" به خاطر تفاوت شکل حروف پیدا نمی‌شود. جواب تابع زیر، 9 است:

```
MsgBox InStr (1, "Richard Shepherd", "shepherd" , vbTextCompare)
```

مثال بعدی از دو پارامتر اختیاری استفاده می‌کند. به موقعیت نقطه شروع توجه کنید:

```
MsgBox InStr (10, "Richard Shepherd", "shepherd" , vbTextCompare)
```

نتیجه این کد، 0 است (رشته مورد نظر پیدا نشد) چون موقعیت شروع جستجوی رشته پس از جایی است که رشته مورد نظر در آنجا قرار دارد.

بدبختانه تابعی برای جستجوی رو به عقب در رشته وجود ندارد هر چند می‌توانید برای انجام این کار از حلقه For..Next و تابع Mid در کد استفاده کنید.

توابع

در این بخش مروری بر پرکاربردترین توابع VBA خواهیم داشت. توابع VBA بسیار زیاد هستند و ما در این‌جا تنها معروف‌ترین آن‌ها را می‌آوریم.

Len

این تابع، تعداد کاراکترهای موجود در یک رشته را برمی‌گرداند. مثال زیر، مقدار 3 را برمی‌گرداند:

```
MsgBox Len("abc")
```

خروجی مثال زیر، 8 است:

```
MsgBox Len("shepherd")
```

این تابع برای دستورهای جابه‌جایی رشته‌ها بسیار مفید است. برای مثال اگر 4 کاراکتر آخر رشته‌ای را بخواهید که طول آن متغیر است با این تابع می‌توانید طول آن رشته را مشخص کنید.

Abs

این تابع مقدار قدر مطلق را برمی‌گرداند. در هر دو مثال زیر، خروجی برابر 1 است:

```
MsgBox Abs(1)
MsgBox Abs(-1)
```

Int

این تابع، عددی را دریافت می‌کند و آن‌را تا نزدیک‌ترین عدد صحیح، گرد می‌کند. برای مثال، خروجی کد زیر، 1 است:

```
MsgBox Int(1.2)
```

خروجی تابع زیر نیز علیرغم نزدیکی آن به عدد 2، برابر 1 است:

```
MsgBox Int(1.99)
```


Sqr

این تابع، ریشه دوم یک عدد را برمی‌گرداند برای مثال خروجی این کد برابر با 2 است:

```
MsgBox Sqr(4)
```

خروجی کد زیر برابر با 1.732 است:

```
MsgBox Sqr(3)
```

Asc

این تابع، کد اسکی یک کاراکتر را برمی‌گرداند که مقداری بین 0 تا 255 است. خروجی کد زیر برابر با 65 است:

```
MsgBox Asc("A")
```

خروجی کد زیر برابر با 105 است:

```
MsgBox Asc("i")
```

توجه کنید که این تابع تنها روی کاراکتر نخست رشته، عمل می‌کند پس خروجی کد زیر برابر با 114 است که کد اسکی کاراکتر r است:

```
Asc ("richard")
```

Chr

این تابع برعکس تابع Asc عمل می‌کند و کد اسکی را گرفته و کاراکتر معادل با آن را برمی‌گرداند. برای مثال خروجی کد زیر برابر با "A" است:

```
MsgBox Chr(65)
```

خروجی کد زیر، رشته "i" است:

```
MsgBox Chr(105)
```

چون این تابع با مجموعه تمام کاراکترها سروکار دارد پس شامل کاراکترهای قابل چاپ است. برای مثال کد اسکی 13 معادل با کاراکتر برگشت به سر خط (carriage return) است که اگر بخواهیم مثلاً در یک کادر پیغام به سر خط برویم بسیار مفید است:

```
MsgBox "This is " & Chr(13) & "a carriage return"
```

توابع تبدیل

این توابع برای تبدیل یک مقدار از یک قالب به قالب دیگر به کار می‌روند. برای مثال می‌توانیم به تبدیل یک مقدار عددی به مقدار رشته‌ای و تبدیل دوباره آن مقدار رشته‌ای به مقدار عددی اشاره کنیم. این توابع برای تبدیل مقادیر به قالب‌های گوناگون بسیار سودمند هستند. برای مثال ممکن است در یک عدد 4 رقمی، رقم دوم آن را نیاز داشته باشیم. ساده‌ترین راه این است که آن را به یک مقدار رشته‌ای تبدیل کنیم و با استفاده از تابع Mid، آن رقم را جدا کنیم؛ سپس می‌توانیم آن را دوباره برای محاسبات بعدی به یک مقدار عددی تبدیل کنیم.

Cstr

این تابع، یک مقدار عددی را به مقدار رشته‌ای تبدیل می‌کند. خروجی کد زیر، رشته "1234" است:

```
Cstr (1234)
```

CInt

این تابع یک مقدار عددی یا یک رشته را به یک عدد صحیح (2-integer بایتی) تبدیل می‌کند. در خروجی این تابع هیچ رقم اعشاری قرار نمی‌گیرد. خروجی هر دو مثال زیر، مقدار 123 است:

```
CInt(123.45)
CInt("123.45")
```

این تابع مثل تابع Int عمل نمی‌کند بلکه عملیات گرد کردن را به صورت معمول انجام می‌دهد. اگر کاراکتر غیر عددی در عبارت وجود داشته باشد پیغام Type Mismatch (عدم تطبیق نوع داده) ارسال می‌شود.

CLng

این تابع، یک مقدار یا رشته را به عدد صحیح طولانی(4 بایتی long integer) تبدیل می‌کند و البته باز هم قسمت اعشاری حذف می‌شود. خروجی هر دو مثال زیر، 123456789 است:

```
CLng(123456789.45)
CLng("123456789.45")
```

این تابع مثل تابع Int عمل نمی‌کند بلکه عملیات گرد کردن را به صورت معمول انجام می‌دهد. اگر کاراکتر غیر عددی در عبارت وجود داشته باشد پیغام Type Mismatch (عدم تطبیق نوع داده) ارسال می‌شود.

CDbl

این تابع یک مقدار عددی یا رشته‌ای را به عددی اعشاری با دقت مضاعف (double – 8 بایتی) تبدیل می‌کند:

```
CDbl("123.56789")
```

خروجی کد زیر برابر با 123.56789 است.

اگر کاراکتر غیرعددی در عبارت وجود داشته باشد پیغام Type Mismatch (عدم تطبیق نوع داده)ارسال می‌شود.

Val

این تابع، یک مقدار رشته‌ای را به یک مقدار عددی تبدیل می‌کند. این تابع نسبت به توابع CInt و CLng انعطاف پذیری بیشتری دارد و کاراکترهای غیرعددی را هم قبول می‌کند:

```
Val("123")
```

خروجی کد فوق، 123 است. کد زیر نیز خروجی 123.45 را بر می‌گرداند:

```
Val("123.45")
```

خروجی کد زیر هم 12 است:

```
Val("12Richard")
```

خروجی کد زیر برابر صفر است و این به معنای آن است که هیچ کاراکتر عددی وجود ندارد:

```
Val("Richard")
```

تابع Format

تابع Format یکی از پرکاربردترین و پیچیده ترین توابع در VBA است. این تابع به شما اجازه می‌دهد تا اعداد را در قالب خروجی دلخواه خود قالب بندی کنید و این دقیقاً مثل روشی است که Excel یک سلول را قالب بندی می‌کند. در آن‌جا هم شما می‌توانید از بین چند گزینه، چگونگی نمایش عدد در آن سلول را مشخص کنید.

تابع Format دقیقاً شبیه قالب بندی یک عدد یا تاریخ در یک سلول از صفحه گسترده عمل می‌کند به استثنای اینکه این کار از داخل یک کد انجام می‌شود. اگر می‌خواهید عددی را در یک کادر پیغام یا فرم کاربر نمایش دهید، این تابع برای خوانا کردن آن عدد بسیار مفید است بویژه اگر با یک عدد بزرگ سروکار داشته باشید:

```
MsgBox Format (1234567.89 , "#,###.#")
```

خروجی کد فوق، 1234567.89 بود.

در رشته قالب‌بندی، هر کاراکتر # نشان دهنده جای یک رقم است. کلما به عنوان جدا کننده هر سه رقم از یکدیگر به‌کار می‌رود. بعد از علامت اعشار، تنها یک رقم مشخص شده (با کاراکتر #) که نشان می‌دهد عدد تا یک رقم اعشار گرد شده است.

هم‌چنان‌که در جدول ۲-۵ می‌بینید می‌توانیم از نام قالب‌های از پیش تعریف شده نیز به جای رشته قالب بندی استفاده کنیم.

جدول ۲-۵ قالب‌های از پیش تعریف شده

نام قالب	شرح
General Number	نمایش عدد به همان صورتی که هست.
Currency	نمایش عدد با نماد واحد پول، از جداکننده هزارگان استفاده کرده و اعداد منفی را داخل کروشه قرار می‌دهد، تا دو رقم اعشار دارد.
Fixed	در سمت چپ اعشار حداقل یک رقم و در سمت راست آن دو رقم قرار می‌دهد.
Standard	عدد را با جداگر هزارگان نمایش می‌دهد و دو رقم اعشار دارد.
Percent	عدد را تقسیم بر صد می‌کند و پس از آن علامت در صد(٪) قرار می‌دهد، دو رقم اعشار دارد.
Scientific	از نماد علمی استاندارد استفاده می‌کند.
Yes/No	اگر عدد صفر باشد No و در غیر این صورت Yes را نشان می‌دهد.
True/False	اگر عدد صفر باشد False و در غیر این صورت True را نشان می‌دهد.
On/Off	اگر عدد صفر باشد Off و در غیر این صورت On را نشان می‌دهد.

در مثال زیر از قالب"Currency" استفاده کرده‌ایم:

```
MsgBox Format (1234567.89 , "Currency")
```

نتیجه کد فوق بسته به نماد واحد پول (currency) به‌کار گرفته شده در تنظیمات ویندوز، $1,234,567.89 است. دیگر تنظیمات ممکن است علامت £ یا یورو را نشان دهند.

همان‌طور که در جدول ۳-۵ می‌بینید چند کاراکتر وجود دارند که در تعریف قالب‌بندی تعریف شده توسط کاربر به‌کار می‌روند.

جدول ۳-۵ قالب‌های تعریف شده کاربر

کاراکتر	شرح
Null String	بدون قالب بندی
0	معادل با جای یک رقم است. می‌تواند یک رقم یا یک صفر باشد. اگر یک رقم به جای آن قرار گیرد، آن رقم نمایش داده می شود و در غیر این صورت به جای آن صفر قرار می‌گیرد.
#	معادل با جای یک رقم است. به جای آن یا یک رقم می‌گذاریم یا چیزی قرار نمی‌گیرد.
.	نشانه اعشار است و برای هر عدد تنها یک بار از آن می‌توان استفاده کرد.
٪	علامت درصد است. عدد را تقسیم بر صد می‌کند و کاراکتر ٪ را در پایان آن قرار می‌دهد.
,	جدا کننده هزارگان است و در صورتی به کار می‌رود که از 0 یا # استفاده کرده باشیم.
E-E+	قالب بندی علمی است و یک عدد را به صورت نمایی نشان می‌دهد.
:	جدا کننده زمان است و زمان را به ساعت، دقیقه، ثانیه تقسیم می‌کند.
/	جدا کننده تاریخ است. قالب نمایش تاریخ را مشخص می‌کند.
$( )£+-	یک کاراکتر حرفی نمایش می‌دهد. برای نمایش کاراکتری غیر از آن‌ها که در اینجا فهرست شده‌اند قبل از آن کاراکتر (\) قرار دهید.

رشته قالب‌بندی می‌تواند تا چهار بخش داشته باشد که با کاراکتر (;) از هم جدا می‌شوند. این بخش‌ها قالب‌بندی‌های مختلفی دارند که روی مقادیر گوناگون اجرا می‌شوند (مثل اعداد مثبت و منفی). برای مثال ممکن است بخواهید دو طرف یک عدد منفی را علامت پرانتز قرار دهید:

```
MsgBox Format (12345.67 , "$#,##0; ($#.##0)")
```

جدول ۴-۵، جزییات مشروح درباره تعداد بخش‌هایی را که در قالب‌بندی عدد به‌کار می‌روند ارایه می‌کنند:

جدول ۴-۵

بخش	جزییات
فقط یک بخش	روی تمام مقادیر اعمال می‌شود
دو بخش	بخش اول برای مقادیر مثبت، بخش دوم برای مقادیر منفی
سه بخش	بخش اول برای مقادیر مثبت، بخش دوم برای مقادیر منفی و بخش سوم برای صفرها
چهار بخش	بخش اول برای مقادیر مثبت، بخش دوم برای مقادیر منفی، بخش سوم برای صفرها و بخش چهارم برای مقادیر Null

هم‌چنین همان‌طور که در جدول ۵-۵ می‌بینید قالب‌بندی‌های از پیش تعریف شده‌ای برای زمان و تاریخ وجود دارد. آن‌ها توسط تنظیمات Time و Date در بخش Control Panel کنترل می‌شوند.

جدول ۵-۵ قالب‌های از پیش تعریف شده تاریخ و ساعت

نام قالب	شرح
General Date	تاریخ و یا زمان را نشان می‌دهد. برای اعداد از نوع REAL، تاریخ و ساعت را نشان می‌دهد. اعداد Integer تنها ساعت را نشان می‌دهند.
Long Date	هم‌چنان‌که در تنظیمات بین المللی کنترل پنل مشخص شده است، یک تاریخ طولانی را نشان می‌دهد.
Medium Date	تاریخی کوتاه را به صورت تعریف شده در تنظیمات بین المللی تاریخ کوتاه کنترل پنل نشان می‌دهد. البته ماه را با حروف مخفف نشان می‌دهد.
Short Date	تاریخ کوتاهی را به صورت تعریف شده در تنظیمات بین‌المللی تاریخ کوتاه کنترل پنل نشان می‌دهد.
Long Time	یک ساعت طولانی را به صورت تعریف شده در تنظیمات بین‌المللی کنترل پنل نشان می‌دهد.
Medium Time	زمان را در قالب ۱۲ ساعتی با استفاده از ساعت، دقیقه و ثانیه نشان می‌دهد.
Short Time	زمان را در قالب ۲۴ ساعتی نشان می‌دهد.

چند کاراکتر وجود دارند که می‌توانیم از آن‌ها برای ایجاد قالب‌های تاریخ و زمان توسط کاربر استفاده کنیم. شرح آن‌ها را در جدول ۶-۵ مشاهده می‌کنید.

جدول۶ -۵ قالب‌های تاریخ / زمان

کاراکتر	مفهوم
c	تاریخ را به صورت ddddd و زمان را به صورت ttttt نشان می‌دهد.
d	روز را به صورت عددی، بدون صفر مجازی نمایش می‌دهد.
dd	روز را به صورت عددی، همراه با صفر مجازی نمایش می‌دهد.
ddd	نام روز را به صورت مخفف نمایش می‌دهد (sun,sat,...).
dddd	نام کامل هر روز را نشان می‌دهد (Sunday, ...).
ddddd	شماره سریال تاریخ را به صورت یک تاریخ کامل بر طبق Short Date در تنظیمات بین المللی کنترل پنل نشان می‌دهد.
dddddd	شماره سریال تاریخ را به صورت یک تاریخ کامل بر طبق Long Date در تنظیمات بین المللی کنترل پنل نشان می‌دهد.
w	روز هفته را به صورت یک عدد نشان می‌دهد (1=Sunday).
ww	هفته را در سال به صورت یک عدد نمایش می‌دهد (۵۳-۱).
m	ماه را به صورت یک عدد و بدون صفر مجازی نشان می‌دهد.
mm	ماه را به صورت یک عدد و همراه صفر مجازی نشان می‌دهد.
mmm	نام ماه را به صورت مخفف نشان می‌دهد(Jan.,Dec.).
mmmm	نام کامل ماه را نشان می‌دهد (January,December).
q	فصل سال را به صورت یک عدد نشان می‌دهد (۴-۱).
y	روز در سال را به صورت یک عدد نشان می‌دهد (۳۶۶-۱).
yy	سال را به صورت یک عدد دو رقمی نمایش می‌دهد.
yyyy	سال را به صورت یک عدد چهار رقمی نمایش می‌دهد.
h	ساعت را به صورت عددی، بدون صفر مجازی نمایش می‌دهد.
hh	ساعت را به صورت عددی، همراه با صفر مجازی نمایش می‌دهد.
n	دقیقه را به صورت عددی، بدون صفر مجازی نمایش می‌دهد.
nn	دقیقه را به صورت عددی، همراه با صفر مجازی نمایش می‌دهد.

کاراکتر	مفهوم
s	ثانیه را به صورت عددی، بدون صفر مجازی نمایش می‌دهد.
ss	ثانیه را به صورت عددی، همراه با صفر مجازی نمایش می‌دهد.
ttttt	شماره سریال ساعت را به صورت یک ساعت کامل نشان می‌دهد.
AM/PM	زمان را به صورت 12 ساعتی نشان می‌دهد و از AM یا PM برای نمایش قبل و بعدازظهر استفاده می‌کند.
am/pm	زمان را به صورت 12 ساعتی نشان می‌دهد و از am یا pm برای نمایش قبل و بعدازظهر استفاده می‌کند.
A/P	زمان را به صورت 12 ساعتی نشان می‌دهد و از A یا P برای نمایش قبل و بعدازظهر استفاده می‌کند.
a/p	زمان را به صورت 12 ساعتی نشان می‌دهد و از a یا p برای نمایش قبل و بعدازظهر استفاده می‌کند.

در این‌جا مثالی از قالب‌بندی زمان بر حسب ساعت، دقیقه و ثانیه مطرح می‌شود:

```
MsgBox Format(Now( ) , "hh:mm:ss AM/PM")
```

قالب‌بندی‌های تاریخ (DATE) مثل اعداد می‌توانند از چند بخش تشکیل شوند. تمام داده‌های تاریخ حتماً یک بخش دارند؛ داده‌های دو بخشی به معنای آن است که بخش نخست برای تمام داده‌ها به‌کار می‌رود و بخش دوم تنها برای رشته‌های تهی یا آن‌هایی که طولشان برابر صفر است به‌کار می‌رود. برای مثال به جدول ۷-۵ توجه کنید:

جدول ۷-۵

رشته قالب‌بندی	تعریف
mm/dd/yy	01/03/03
dd-mmm-yyyy	01-Mar-2003
hh:mm	A.M./P.M.

هم‌چنین می‌توانید از کاراکترهای خاص در رشته قالب‌بندی استفاده کنید تا قالب‌بندی دلخواه‌تان را ایجاد نمایید (جدول ۸-۵).

جدول ۸-۵ کاراکترهای قالب‌بندی

کاراکتر	تعریف
@	معادل با یک کاراکتر است. یک کاراکتر یا یک فاصله را نشان می‌دهد. اگر به جای آن کاراکتری قرار نگیرد، کاراکتر فاصله را قرار می‌دهد.
&	معادل با یک کاراکتر است. یا یک کاراکتر نشان می‌دهد یا چیزی را نشان نمی‌دهد.
<	کاراکترها را کوچک تایپ می‌کند.
>	کاراکترها را بزرگ تایپ می‌کند.
!	کاری می‌کند که کاراکترها از چپ1 به راست تایپ شوند.

در این‌جا چند مثال از به‌کارگیری تابع Format روی اعداد و رشته‌ها ذکر می‌شود:

```
MsgBox "This is " & Format("1000", "@@@,@@@")
MsgBox "This is " & Format("1000", "&&&,&&&")
MsgBox Format ("richard", ">")
```

توابع date و time

چند تابع ویژه برای کار روی تاریخ و زمان وجود دارند که در این بخش آن‌ها را تعریف می‌کنیم:

تابع Now

این تابع، تاریخ و ساعت فعلی را برمی‌گرداند:

```
MsgBox Now
```

تابع Date

این تابع، تاریخ فعلی را با توجه به تنظیمات ویندوز برمی‌گرداند:

```
MsgBox Date
```

از این تابع می‌توان برای تنظیم تاریخ فعلی سیستم نیز استفاده کرد:

```
Date = "03/31/03"
```

تابع Time

این تابع برای برگرداندن زمان فعلی سیستم به‌کار می‌رود:

```
MsgBox Time
```

از این تابع می‌توان برای تنظیم تاریخ فعلی سیستم نیز استفاده کرد:

```
Time = "13:11:00"
```

کد فوق مثالی از تنظیم ساعت برای 11 دقیقه پس از 13 بعد از ظهر است.

تابع DateAdd

این تابع اجازه می‌دهد تا یک مقدار زمان مشخص را به تاریخ فعلی اضافه کرده یا از آن کسر کنیم. ساختار دستوری آن به شرح زیر است:

```
DateAdd (interval, number, date)
```

Interval رشته‌ای است که مقدار فاصله زمانی مورد نظر ما را مشخص می‌کند.
جدول زیر لیستی از انواع فواصل گوناگون را که می‌توان در این تابع به‌کار گرفت، مشخص می‌کند.

جدول ۹-۵

دوره زمانی	فاصله
سال	yyy
فصل	q
ماه	m
روز سال	y
روز	d
روز هفته	w
هفته	ww
ساعت	h
دقیقه	n
ثانیه	s

Number یک مقدار عددی است که عدد مورد نظر شما را که می‌خواهید به این تابع اضافه شود مشخص می‌کند. استفاده از اعداد منفی نیز مجاز است و باعث کسر شدن از تاریخ مورد نظر می‌شود.
Date تاریخی است که قرار است اضافه شود البته می‌تواند نام متغیری نیز باشد که حاوی یک تاریخ است. مثال زیر یک ماه به ژانویه اضافه می‌کند و خروجی آن 1-Feb-03 خواهد بود:

```
MsgBox DateAdd ("m",1, "1-Jan-03")
```

مثال زیر، دوهفته به تاریخ فعلی اضافه می‌کند و خروجی 16-Jan-03 را برمی‌گرداند:

```
MsgBox DateAdd ("ww",2, "1-Jan-03")
```

مثال زیر نیز دو روز را از تاریخ اول ژانویه 2003 کسر می‌کند و خروجی 30-Dec-02 را برمی‌گرداند:

```
MsgBox DateAdd ("d",-2, "1-Jan-03")
```

دقت کنید که خروجی‌های فوق با تنظیمات ویندوز شما مرتبط هستند.

تابع DateDiff

این تابع، فاصله بین دو تاریخ مشخص شده را برمی‌گرداند:

```
DateDiff (interval, date1, date2)
```

Interval یک عبارت رشته‌ای بر مبنای صفحه بعد است و مشخص کننده نوع فاصله زمانی است و رشته date1 نشان دهنده تاریخ شروع و date2 نشان دهنده تاریخ خاتمه است:

جدول ۱۰-۵

دوره زمانی	فاصله
سال	yyyy
فصل	q
ماه	m
روز سال	y
روز	d
روز هفته	w

جدول ۱۱-۵

هفته	ww
ساعت	h
دقیقه	n
ثانیه	s

مثال زیر، کاربرد این تابع را نشان می‌دهد:

```
MsgBox DateDiff ("m", "1-jan-03", "15-mar-03")
```

خروجی کد فوق، 2 است چون 2 ماه فاصله بین اول ژانویه تا اول مارس وجود دارد. توجه کنید که این تابع، جواب را گرد می‌کند و اگر حتی date2، 30 مارس هم بود نیز جواب 2 برمی‌گشت. تنها وقتی date2، اول آوریل باشد جواب خروجی 3 می‌شود.

تابع DatePart

این تابع یک قسمت خاص از یک تاریخ را برمی‌گرداند:

```
DatePart (interval, date)
```

Interval، مدت زمانی است که بر مبنای جدول ۱۲-۵ مشخص می‌شود و date تاریخی است که می‌خواهید مورد بررسی قرار گیرد:

جدول ۱۲-۵

دوره زمانی	فاصله
سال	yyyy
فصل	q
ماه	m
روز سال	y
روز	d
روز هفته	w
هفته	ww
ساعت	h

دوره زمانی	فاصله
دقیقه	n
ثانیه	s

ساختار دستوری DatePart به صورت زیر است:

```
MsgBox DatePart ("q", "1-mar-03")
```

کد فوق، مقدار 1 را بر می‌گرداند چون اول مارس در فصل ۱ قرار دارد.
خروجی کد زیر، 3 است چون مارس، ماه سوم است:

```
MsgBox DatePart ("m", "1-mar-03")
```

تابع DateSerial

این تابع، شماره سریال یک سال، ماه و روز مشخص را که به صورت اعداد صحیح وارد شده‌اند، برمی‌گرداند. شماره سریال در واقع تعداد روزهای سپری شده از اول ژانویه سال 1900 میلادی تا آن تاریخ است:

```
DateSerial (year, month, day)
```

Year، عددی بین 100 و 9999 ؛ month عددی بین 1 تا 12 و day عددی بین 1 تا 31 است. هر سه آن‌ها می‌توانند یک عبارت یا متغیر عددی هم باشند.
برای مثال خروجی کد زیر مقدار 37686 خواهد بود که نشان دهنده تاریخ 6 مارس 2003 است:

```
MsgBox CDbl (DateSerial (2003, 3, 6))
```

لازم است در این کد، از CDbl (تبدیل به نوع double) استفاده کنیم در غیر این صورت، مقداری که در کادر پیغام به نمایش درمی‌آید برحسب تنظیمات ویندوز خواهد بود.

تابع DateValue

این تابع یک تاریخ را به یک مقدار تبدیل می‌کند. برای مثال کد زیر مقدار 37686 خواهد بود که نشان‌دهنده تاریخ 6 مارس 2003 است:

```
MsgBox CDbl (DateValue ("06-Mar-2003"))
```

لازم است در این کد، از CDbl (تبدیل به نوع double) استفاده کنیم در غیر این صورت مقداری که در کادر پیغام به نمایش درمی‌آید برحسب تنظیمات ویندوز خواهد بود.

تابع day

این تابع یک عدد صحیح بین 1 و 31 را برمی‌گرداند که نشان دهنده روز تاریخ مشخص شده در ماه مربوطه است:

```
Day (dateexpression)
```

Dateexpression می‌تواند یک رشته حاوی تاریخ یا یک عبارت عددی که نشان دهنده یک تاریخ است باشد.

خروجی هر دو کد زیر 6 است که نشان دهنده ششمین روز از ماه مارس است:

```
MsgBox Day(37686)
MsgBox Day("6-Mar-2003")
```

تابع Hour

این تابع یک عدد صحیح بین 0 و 23 را نشان می‌دهد که نشان دهنده ساعتی از روز در تاریخ مشخص شده است:

```
Hour (dateexpression)
```

مثال‌هایی از Dateexpression در ساختار فوق می‌تواند "31-Dec-2002 12:00" یا ساعتی بدون ذکر تاریخ مثل "09:00" باشد:

```
MsgBox Hour ("17:50")
```

خروجی کد فوق مقدار 17 است.

کد زیر مقدار 16 را برمی‌گرداند:

```
MsgBox Hour ("6-Mar-2003 4:30pm")
```

کد زیر مقدار 11 را برمی‌گرداند؛ جواب 11 تقسیم بر 24 مساوی با 4583330 . 0 است که مقدار ساعت برای 11:00 Am می‌باشد:

```
MsgBox Hour(11 / 24)
```


تابع Month

این تابع یک عدد صحیح بین 1 تا 12 را بر حسب Dateexpression برمی‌گرداند:

```
Month (dateexpression)
```

مثالی از Dateexpression می‌تواند "31-Dec-2002 12:00" باشد البته می‌توان از یک ساعت بدون ذکر تاریخ (مثل "09:00") نیز استفاده کرد.

خروجی هر دو کد زیر مقدار 3 است چون هر دو عبارت تاریخ نشان دهنده 6 مارس 2003 هستند:

```
MsgBox Month(37686)
MsgBox Month("6-Mar-2003")
```

تابع Second

این تابع عددی صحیح بین 0 تا 59 را برمی‌گرداند و این مقدار بسته به timeexpression است که نشان دهنده ثانیه‌ها از یک دقیقه خاص می‌باشد:

```
Second (timeexpression)
```

مثالی از timeexpression می‌تواند "31-Dec-2002 12:00" باشد البته می‌توان از یک ساعت بدون ذکر تاریخ (مثل "09:00") نیز استفاده کرد.

خروجی کد زیر مقدار 48 خواهد بود:

```
MsgBox Second("4:35:48pm")
```

تابع Minute

این تابع یک عدد صحیح را برمی‌گرداند که نشان دهنده دقیقه واقعی از یک زمان مشخص است:

```
MsgBox Minute(11.27 / 24)
```

خروجی کد فوق مقدار 16 است چون 11.27 برابر با 11:16:12 AM است. ممکن است این جواب کمی گیج کننده به نظر برسد که به خاطر کار با قسمت اعشاری است.

کدهای زیر دو مثال از به‌کارگیری تابع Minute هستند:

```
MsgBox Minute("4:35pm")
MsgBox Minute(11.25 / 24)
```

تابع Year

این تابع یک عدد صحیح را برمی‌گرداند که نشان دهنده تعداد سال سپری شده واقعی در تاریخ مذکور است:

```
MsgBox Year(37686)
MsgBox Year("6-Mar-2003")
```

تابع Weekday

این تابع یک عدد صحیح بین 1 (معرف یکشنبه) و 7 (معرف شنبه) است که نشان می‌دهد تاریخ مذکور در کدام روز هفته واقع است:

```
Weekday(dateexpression)
MsgBox Weekday("6-Mar-2003")
```

خروجی کد فوق مقدار 5 است که معرف پنجشنبه است.

اگر بخواهیم یک تاریخ همیشه به صورت پیش فرض در یک روز خاص هفته باشد از این تابع استفاده می‌کنیم برای مثال اگر بخواهیم یک تاریخ همیشه هفته‌ای را نشان دهد که با جمعه خاتمه می‌یابد از فرمول زیر استفاده می‌کنیم:

```
MsgBox Now – weekday(Now) + 6
```

تابع Weekday از یکشنبه آغاز می‌شود پس می‌تواند Now را دوباره به آخرین یکشنبه برگرداند و سپس 6 را به آن اضافه کند تا به جمعه برسد. هم‌چنین می‌توانید از این تابع برای محاسبه تعداد روزهای کاری بین دو تاریخ استفاده کنید:

```
For n = DateValue("1-Jan-03") To DateValue("18-Jan-03")

        If Weekday(n) = 1 Or Weekday(n) = 7 Then

Else

        WorkDay = WorkDay + 1

End If

Next n

MsgBox WorkDay
```


در کد فوق WorkDay، مقدار 13 را برمی‌گرداند که تعداد روزهای کاری بین دو تاریخ مذکور است.

دستور SendKeys

دستور SendKeys، دقیقاً یک تابع نیست بلکه یک عبارت دستوری است که به شما اجازه می‌دهد که به‌وسیله ارسال کلیک‌ها (که روی صفحه کلید انجام می‌شوند) به برنامه‌ای دیگر، دستوراتی را برای آن صادر کنید؛ دقیقاً انگار در حال تایپ دستورات در همان برنامه هستید. از این تابع می‌توان برای اتوماسیون سطح پایین برنامه‌هایی که از اتوماسیون OLE پشتیبانی نمی‌کنند، استفاده کرد.

SendKeys، یک یا چند ضربه روی صفحه کلید را به پنجره فعال ارسال می‌کند:

```
SendKeys keytext [ ,wait]
```

در مثال فوق، Keytext رشته‌ای شامل چند کلید است که به پنجره فعال ارسال می‌شوند؛ wait نیز یک مقدار بولین است (True/False). اگر wait، True باشد آن‌گاه کلیدها باید قبل از برگشتن کنترل به رویه، مورد پردازش قرار گیرند. اگر wait، False باشد آن‌گاه کنترل بلافاصله پس از ارسال ضربات روی صفحه کلید، به رویه، بر می‌گردد. اگر wait حذف شده باشد، مقدار آن به صورت پیش فرض False تلقی می‌شود.

در هنگام ارسال کلیدها به یک برنامه دیگر، مقدار wait بسیار مهم است. اگر برنامه به سرعت در حال اجرا باشد ممکن است دستور SendKeys نادیده گرفته شود. مقدار wait را بدین خاطر True تنظیم می‌کنیم که ابتدا ضربات روی صفحه کلید پردازش شوند و برنامه قبل از این کار نتواند به اجرای خود ادامه دهد.

معمولاً در اغلب موارد از خود کلیدهای صفحه کلید در دستور SendKeys استفاده می‌شود. برای مثال کد زیر:

```
SendKeys "A123"
```

عبارت A123 را به پنجره فعال ارسال می‌کند.

البته این کار با این فرض انجام می‌شود که یک برنامه در حال اجرا داشته باشیم و پنجره اجرای آن برنامه پنجره فعال ما باشد.

دو دستور دیگر نیز به انجام این عملیات کمک می‌کنند. اولی تابع Shell است که اجازه می‌دهد یک برنامه کاربردی دیگر راه‌اندازی شود و کنترل به آن برنامه جدید انتقال یابد:

```
Shell (commandstring [ ,windowstyle])
```

commandstring، خط فرمانی برای فراخوانی برنامه‌های کاربردی است. اگر به shortcut های موجود

در صفحه دسک‌تاپ خود نگاه کنید می‌بینید که در آنجا یک کادر متنی با نام Target وجود دارد که نام فایل اجرایی و مسیر رسیدن به برنامه کاربردی به علاوه پارامترهای ضروری را در خود دارد. از این کادر متنی به عنوان commandstring استفاده می‌شود.

پارامتر windowstyle نشان دهنده چگونگی باز شدن برنامه کاربردی است (این‌که به صورت یک پنجره معمولی، آیکن یا به صورت پنهان باز شود).

پیش از آن‌که بتوانید ضربات صفحه کلید را به برنامه کاربردی ارسال کنید لازم است آن‌را باز نمایید. مثال زیر نشان دهنده دستور باز کردن ماشین حساب ویندوز است:

```
x = Shell("calc.exe",1)
```

این کد، برنامه ماشین حساب ویندوز را در یک پنجره استاندارد باز کرده و آن را به صورت پنجره فعال در می‌آورد و از این پس می‌توانیم با استفاده از SendKeys، ضربات روی صفحه کلید را به آن ارسال کنیم.

AppActivate، دستور دیگری است که قبل از ارسال کلیدها باید از آن استفاده کنید. اگر برنامه قبلاً بارگذاری شده باشد نیازی نیست از تابع Shell برای بارگذاری مجدد آن استفاده کنیم بلکه فقط باید پنجره برنامه را به صورت فعال در آوریم تا بتوانیم کلیدها را به آن ارسال کنیم. این دستور اجازه می‌دهد تا برنامه‌ای را که قبلاً بارگذاری شده است، با استفاده از عنوانی که در نوار عنوان برنامه وجود دارد، فعال کنیم:

```
AppActivate "Microsoft Word"
```

در این روش می‌توانیم اتوماسیون ساده دیگر برنامه‌های کاربردی را انجام دهیم:

```
Sub test_sendkeys ( )

x = Shell("calc.exe")

For n = 1 to 10

        SendKeys n & "{+}" , True
Next n

MsgBox "Press OK to close calculator"

AppActivate "calculator"

SendKeys "%{F4}" , True
```



```
End Sub
```

در این مثال، ماشین حساب ویندوز بارگذاری شده است و سپس باکمک یک حلقه For..Next آن‌را وادار به جمع اعداد ۱ تا ۱۰ می‌کنیم.

چون برنامه Excel جایی است که کد از آن‌جا اجرا می‌شود، کادر پیغام در آن‌جا ظاهر می‌شود. آیکن Excel روی نوار ابزار ویندوز روشن می‌شود. Excel را انتخاب کرده و روی OK کلیک کنید آن‌گاه ماشین حساب بسته می‌شود.

کاراکترهای جمع (+)، توان (^)، درصد(٪)، تقریب(~) و پرانتز (()) همگی معانی خاصی برای SendKeys دارند. برای مشخص کردن یکی از این کاراکترها باید آن‌را داخل آکولاد ({}) قرار دهیم مثلاً برای مشخص کردن علامت جمع باید {+} را تایپ کنیم. کاراکترهای کروشه ([]) هیچ معنایی برای SendKeys ندارند اما می‌توانیم آن‌ها را هم در بین آکولاد قرا دهیم.

برای مشخص کردن کاراکترهای ویژه‌ای که در هنگام فشار دادن کلید به نمایش در نمی‌آیند(مثل Enter و TAB) و کلیدهایی که به جای نمایش کاراکتر، یک عملیات اجرا می‌کنند باید از کدهایی که در جدول ۷-۵ نشان داده شده استفاده کنیم.

برای مشخص کردن کلیدهای ترکیبی که همراه با کلیدهای SHIFT، CTRL و ALT کار می‌کنند باید قبل از کد مربوط به این کلیدها، یک یا چند تا از این کدها را قرار دهیم:

جدول ۱۳-۵

کلید	کد
SHIFT	+
CTRL	^
ALT	٪

برای مشخص کردن کلیدهای ترکیبی که همراه با نگه داشتن کلیدهای SHIFT، CTRL و ALT کار می‌کنند باید کد این کلیدها را داخل پرانتز قرار دهیم. برای مثال برای مشخص کردن نگه داشتن SHIFT در هنگام فشردن E و C ، از (EC)+استفاده می‌کنیم. برای مشخص کردن نگه داشتن SHIFT در هنگام فشردن E و سپس فشردن C بدون نگه داشتن SHIFT، از EC+استفاده می‌کنیم.

اگر از SendKeys برای راه اندازی دیگر برنامه‌ها استفاده می‌کنید باید دقت کنید که صفحه کلید هنوز فعال است و اگر در حین اجرای SendKeys، کلیدی فشرده شود نتایج مصیبت باری رخ می‌دهد.

چند سال پیش من یک برنامه SendKeys برای یکی از بانک‌های انگلستان نوشتم که قرار بود با یک برنامه ضبط زمان کار کند. برنامه در طی شب اجرا می‌شد و برگه‌های زمانی برای استفاده در صبح روز جمعه ایجاد می‌کرد. در یکی از صبح‌های جمعه هیچ برگه زمانی ایجاد نشده بود و برنامه به هم ریخته بود. دلیل این امر آن بود که عصر روز قبل یکی از خدمتکاران در هنگام گردگیری صفحه کلید، یکی از دکمه‌های آن را فشار داده بود واین باعث به هم ریختن کل برنامه شده بود.

جدول ۱۴-۵ کلیدهای ویژه‌ای که معمولاً نمایش داده نمی‌شوند.

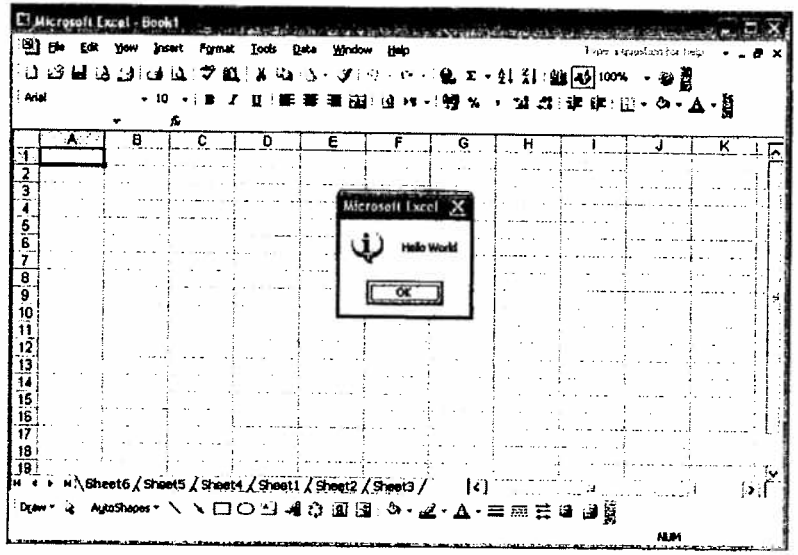
کلید	کد
BACKSPACE	{BACKSPACE},{BS},{BKSP}
BREAK	{BREAK}
CAPSLOCK	{CAPSLOCK}
DEL or DELETE	{DEL} or {DELETE}
DOWN ARROW	{DOWN}
END	{END}
ENTER	{ENTER} or ~
ESC	{ESC}
HELP	{HELP}
HOME	{HOME}
INS or INSERT	{INS} or {INSERT}
LEFT ARROW	{LEFT}
NUM LOCK	{NUMLOCK}
PAGE DOWN	{PGDN}
PAGE UP	{PGUP}
PRINT SCREEN	{PRTSC}
RIGHT ARROW	{RIGHT}
SCROLL LOCK	{SCROLLLOCK}
TAB	{TAB}
UP ARROW	{UP}
F1	{F1}
F2	{F2}
F3	{F3}
F4	{F4}
F5	{F5}
F6	{F6}
F7	{F7}
F8	{F8}
F9	{F9}
F10	{F10}
F11	{F11}
F12	{F12}

کادرهای پیغام

در بسیاری از مثال‌های این کتاب از دستور MsgBox برای ارسال نتایج، به کاربر استفاده کرده‌ایم. شما می‌توانید کدی بنویسید که نتیجه را در یک سلول خاص صفحه گسترده قرار دهد اما کادر پیغام یک روش بسیار ساده برای ارسال داده‌ها به کاربر است. کادر پیغام تنها به یک دستور و متنی که قرار است برای کاربر نمایش داده شود نیاز دارد. تا این‌جا فقط از ساده‌ترین فرم آن استفاده کرده‌ایم:

```
MsgBox "Hello World"
```

تصویر ۱-۵ نتیجه اجرای این کد را نشان می‌دهد که کمی متفاوت از کادرهای پیغامی است که در دیگر برنامه‌های کاربردی دیده‌ایم.



تصویر ۱-۵ یک کادر پیغام ساده

قسمت عنوان (caption) در نوار عنوان، عبارت "Microsoft Excel" را نشان می‌دهد. در این کادر هیچ آیکنی وجود ندارد و تنها دارای یک دکمه انتخاب است.

ثابت	مقدار	شرح
vbDefaultButton2	256	دکمه دوم، دکمه پیش فرض است.
vbDefaultButton3	512	دکمه سوم، دکمه پیش فرض است.
vbDefaultButton4	768	دکمه چهارم، دکمه پیش فرض است.
vbExclamation	48	آیکن Exclamation را نمایش می‌دهد.
vbInformation	64	آیکن Information را نمایش می‌دهد.
vbOKCancel	1	دکمه‌های OK و Cancel را نمایش می‌دهد.
vbOKOnly	0	دکمه OK را نمایش می‌دهد.
vbQuestion	32	آیکن علامت سؤال را نمایش می‌دهد.
vbRetryCancel	5	دکمه‌های Retry و Cancel را نمایش می‌دهد.
vbYesNo	4	دکمه‌های Yes و No را نمایش می‌دهد.
vbYesNoCancel	3	دکمه‌های Yes و No و Cancel را نمایش می‌دهد.

خروجی مثال زیر، یک کادر پیغام با دکمه های Yes و No و یک عنوان تعریف شده خواهد بود:

```
MsgBox "Test message", vbYesNo, "My message"
```

می‌توانیم مقادیر ثابت دکمه و آیکن را با اپراتور Or ترکیب کنیم:

```
x = MsgBox ("Test Message", vbAbortRetryIgnore Or vbCritical)
```

کد فوق یک کادر پیغام با دکمه های Abort، Retry و Ignore و یک آیکن پیغام Critical را نشان میدهد.

نمایش دکمه‌ها کار نسبتاً ساده است اما چطور می‌توانیم تشخیص دهیم که کاربر روی کدام دکمه کلیک کرده است؟ واضح است که برای این کار هم باید کدنویسی کنیم. می‌توانیم پاسخ را در یک متغیر ذخیره کنیم:

```
x = MsgBox ("Test Yes No", vbYesNo, "Test")
```

توجه کنید که در این مثال از x= استفاده کرده‌ایم و در دو طرف پارامترها علامت پرانتز گذاشته‌ایم. بدون استفاده از پرانتزها حتماً با یک پیغام خطا مواجه می‌شویم چون تابعی را فراخوانی می‌کنیم که برای نشان دادن پارامترها به پرانتز نیاز دارد.

این مثال یک کادر پیغام دو دکمه‌ای (Yes و No) را نشان می‌دهد. اگر روی Yes کلیک شود کادر پیغام عدد 6 را نشان می‌دهد(vbYes). اگر روی No کلیک شود کادر پیغام عدد 7 را نشان می‌دهد (vbNo). سپس می‌توانید برای مشخص کردن عملیاتی که قرار است بر طبق انتخاب‌ها انجام شود کدنویسی کنیم:

```
If x = vbYes Then Action1 Else Action2
```

جدول زیر مقادیر برگشتی از یک کادر پیغام را فهرست می‌کند:

جدول ۱۷-۵

ثابت	مقدار	شرح
vbOK	۱	روی دکمه OK کلیک شده است.
vbCancel	۲	روی دکمه Cancel کلیک شده است.
vbAbort	۳	روی دکمه Abort کلیک شده است.
vbRetry	۴	روی دکمه Retry کلیک شده است.
vbIgnore	۵	روی دکمه Ignore کلیک شده است.
vbYes	۶	روی دکمه Yes کلیک شده است.
vbNo	۷	روی دکمه NO کلیک شده است.

فصل ششم

عملگرها

عملگرها، توابع ریاضی، توابع مقایسه‌ای یا عملیات‌های منطقی بین ۲ عدد یا عبارت عددی در برنامه شما را انجام می‌دهند. یک مثال ساده از یک عملگر، عملگر جمع (+) یا تفریق (-) است. حتماً قبلاً در هنگام نوشتن فرمول‌ها در صفحه گسترده از عملگرها استفاده کرده‌اید.

عملگرها دارای تقدم‌هایی هستند که ترکیب انجام محاسبات را مشخص می‌کنند. در دسته‌های مستقل عملگرها (مثل حسابی، مقایسه‌ای و منطقی)، آن‌ها به ترتیب تقدمی که در جدول زیر می‌بینید (از بالا به پایین) ارزیابی می‌شوند:

جدول ۱-۶

عملگر حسابی	عملگر مقایسه‌ای	عملگر منطقی
توان (^)	مساوی (=)	Not
منفی (-)	نامساوی (<>)	And
ضرب و تقسیم (/,*)	کمتر از (<)	Or
تقسیم صحیح (\)	بیشتر از (>)	Xor
باقیمانده صحیح تقسیم (mod)	کمتر یا مساوی (<=)	Eqv
جمع و تفریق (- , +)	بیشتر یا مساوی (=>)	Imp
الحاق رشته‌ای (&)	Like / Is	

در فرمول‌ها، می‌توانیم تقدم را با استفاده از پرانتزها تغییر دهیم (دقیقاً به همان روشی که در Excel انجام دادیم). همیشه فرمول‌هایی که در داخلی‌ترین پرانتز قرار دارند پیش از همه اجرا می‌شوند.

استفاده از پرانتزها، برای تغییر تقدم عملگرها می‌تواند منجر به نتایج متفاوتی نسبت به آن چه ما انتظار داریم بشود. پس در هنگام استفاده از آن‌ها بسیار دقت کنید. کدهای زیر را در یک زیرروال از یک ماژول امتحان کنید:

```
MsgBox (10 + 6) / 3
```

جواب کد فوق، 5.3333 خواهد بود:

```
MsgBox 10 + 6 / 3
```

جواب کد فوق 12 است.

در مثال اول، پرانتزها باعث می‌شوند که قبل از عملیات تقسیم بر 3، عملیات جمع 6+10 انجام شود. در مثال دوم، تقسیم 6/3 نسبت به عملیات جمع با 10، تقدم دارد.

عملگرهای حسابی

این‌ها عملگرهایی هستند که عملیات‌های حسابی را انجام می‌دهند مثل جمع (+)، تفریق (-)، ضرب(*) و تقسیم (/).

عملگر *

این عملگر، دو عدد را در هم ضرب می‌کند:

```
MsgBox 6 * 3
```

جواب کد فوق، 18 است.

به جای اعداد می‌توان از عبارت‌های عددی نیز استفاده کرد. نوع داده نتیجه یک عبارت، وابسته به عبارت‌ها یا اعدادی است که در آن شرکت داشته‌اند.

عملگر+

این عملگر، دو عدد یا عبارت را با یکدیگر جمع می‌کند:

```
MsgBox 4 + 2
```

نتیجه کد فوق، 6 خواهد بود.

این اپراتور هم می‌تواند اعداد را با هم جمع کند و هم می‌تواند رشته‌ها را به یکدیگر الحاق نماید. الحاق رشته‌ها با این عملگر می‌تواند باعث سردرگمی‌شود پس بهتر است برای عملیات الحاق از عملگر & استفاده کنید. مثالی که در پایان این بخش ارایه شده است نشان می‌دهد که الحاق رشته‌ها با عملگر + چطور تحت تأثیر عملگر + قرار می‌گیرد .

اعداد می‌توانند هر عبارت عددی باشند. در این‌جا چند قانون کلی برای جمع و الحاق ذکر می‌شود.

- اگر هردو عملوند عددی باشند؛ با هم جمع می‌شوند.
- اگر هر دو عملوند رشته باشند؛ به هم الحاق می‌شوند.
- اگر یک عملوند عددی و دیگر از نوع Variant باشد، با یکدیگر جمع می‌شوند.
- اگر یک عملوند رشته و دیگری از نوع Variant باشد به یکدیگر الحاق می‌شوند.

اگر یک عملوند عددی و دیگری رشته باشد، پیغام Type Mismatch (عدم تطبیق نوع داده) صادر می‌شود:

```
MsgBox 1 + " Richard"
```

توجه کنید که اگر از اپراتور & الحاق استفاده کرده بودید، چنین امری رخ نمی‌داد:

```
MsgBox 1 & " Richard"
```

عملگر -

این عملگر، یک عدد را از عدد دیگر کم می‌کند پاسخ کد زیر برابر 2 خواهد بود:

```
MsgBox 6 – 4
```

جواب کد زیر،5- خواهد بود:

```
MsgBox -5
```

به جای اعداد می‌توانیم از عبارت‌های عددی هم استفاده کنیم.

عملگر /

این عملگر، دو عدد را بر هم تقسیم می‌کند و نتیجه را به صورت یک عدد اعشاری برمی‌گرداند :

```
MsgBox 6 / 3
```

نتیجه عبارت فوق، 2 است. اگر باقیمانده وجود داشته باشد به صورت عددی اعشاری نمایش داده می‌شود. به جای اعداد می‌توانیم از عبارت‌های ریاضی هم استفاده کنیم.

عملگر \

این عملگر، دو عدد را بر هم تقسیم می‌کند و نتیجه را به صورت یک عدد صحیح برمی‌گرداند:

```
MsgBox 6 \ 4
```

نتیجه کد فوق، 1 است.

به جای اعداد می‌توان از عبارت‌های عددی هم استفاده کرد.

عملگر^

این عملگر، یک عدد را به توان عدد دوم می‌رساند:

```
MsgBox 2 ^ 3
```

پاسخ کد فوق، 8 است (2 به توان 3).

عملگرها می‌توانند هر نوع عبارت عددی باشند. البته عملگر عددی تنها اگر توان (نما) یک عدد صحیح باشد، می‌تواند منفی نیز باشد.

عملگر Mod

این عملگر، دو عدد را بر هم تقسیم کرده و باقیمانده را برمی‌گرداند:

```
MsgBox 6 Mod 4
```

جواب کد فوق 2 است که باقیمانده تقسیم 6 بر 4 است.

این عملگر اغلب در زمانی به کار می‌رود که می‌خواهیم مشخص کنیم یک عدد زوج است یا فرد. اگر پس از تقسیم بر 2، باقیمانده برابر 1 باشد عدد زوج است و در غیر این صورت عدد فرد است.

عملگرهای مقایسه‌ای

این عملگرها، عبارت را با هم مقایسه می‌کنند؛ مانند آن‌چه در فصل ۴، هنگام بررسی تصمیم‌گیری در VBA انجام دادیم:

```
MsgBox 3 > 1
```

چون 3 بزرگ‌تر از 1 است، نتیجه کد فوق True است.

عملگرهای مقایسه‌ای همیشه یک مقدار بولین (False یا True) را برمی‌گردانند به استثنای وقتی که یک مقدار Null در عبارت وجود داشته باشد، که در آن صورت، نتیجه همیشه Null خواهد بود. در این‌جا لیستی از عملگرهای مقایسه‌ای ذکر می‌شوند:

جدول ۲-۶

عملگر	مفهوم
=	مساوی
<>	نامساوی
<	کمتر از
>	بیشتر از
<=	کمتر یا مساوی
>=	بیشتر یا مساوی

اگر هر دو عبارت، عددی باشند آن‌گاه یک مقایسه عددی و اگر هر دو عبارت رشته‌ای باشند یک مقایسه رشته‌ای انجام می‌شود. اما اگر، یکی عددی و دیگری رشته‌ای باشد آن‌گاه خطای Type Mismatch رخ می‌دهد.

عملگر الحاق

این عملگر (&) دو عملوند را به یکدیگر الحاق می‌کند:

```
MsgBox "Richard" & "Shepherd"
```

نتیجه کد فوق، "Richard Shepherd" است.

می‌توانیم با این عملگر،اعداد و رشته‌ها را هم به یکدیگر الحاق کنیم. اما به یاد داشته باشید که نتیجه، یک رشته محسوب می‌شود. نتیجه کد زیر "12 Twelve" است:

```
MsgBox 12 & " Twelve"
```

جواب این کد، 34 است اما یک رشته تلقی می‌شود نه عدد:

```
MsgBox 3 & 4
```

عملگرهای منطقی

این عملگرها روی دو عبارت، نظارت منطقی اعمال می‌نمایند. آن‌ها برای تصمیم‌گیری از ریاضی دودویی ساده استفاده می‌کنند.

عملگر And

این عملگر بر این مبنا کار می‌کند که هردو مقدار یا عبارت، باید مقدار True (غیر صفر) داشته باشند. مقدار True در VBA در واقع 1- است. نتیجه کد زیر، False است چون در هنگام استفاده از عملگر And باید هر دو مقدار، True باشند تا جواب True به‌دست آوریم:

```
MsgBox True And False
```

عملگر Not

این عملگر، یک عملیات Not منطقی را روی دو عدد یا عبارت اجرا می‌کند و در صورت بر قرار نبودن یک شرط، مقدار True برمی‌گرداند:

```
MsgBox Not (2 = 3)
```

نتیجه کد فوق، True است چون مساوی با 2 نیست. پس چون شرط بر قرار نیست، جواب صحیح است.

اپراتور Or

این عملگر بر این مبنا عمل می‌کند که یکی از دو مقدار یا عبارت، True (غیر صفر)باشند. البته اگر هر دوی آن‌ها True باشند هم خروجی این عملگر، True خواهد بود. خروجی کد زیر، True است چون یکی از مقادیر ،True می‌باشد:

```
MsgBox True Or False
```

خروجی کد زیر False است چون هیچ‌کدام از مقادیر، True نیستند:

```
MsgBox False Or False
```

عملگر Xor

این عملگر مانند Or عمل می‌کند به استثنای این‌که اگر هر دو عبارت True باشند جواب False را برمی‌گرداند. تنها اگر یکی از عبارات یا مقادیر True و دیگری False باشد جواب True برمی‌گردد. Xorمخفف " یای انحصاری" است.

خروجی کد زیر،True است:

```
MsgBox True Xor False
```

خروجی کد زیر، False است:

```
MsgBox True Xor True
```

عملگرهای دیگر

عملگر Is

این اپراتور دو متغیر را با هم مقایسه می‌کند تا ببیند آیا آن‌ها مشابه هم هستند یا خیر.

کد زیر مقدار True بر می‌گرداند چون هر دو عبارت مشابه هم هستند (sheet1 مشابه با sheet1 است):

```
MsgBox Worksheets(1) Is Worksheets(1)
```

خروجی کد زیر False است چون دو عبارت مشابه هم نیستند:

```
MsgBox Worksheets(1) Is Worksheets(2)
```

چون sheet1 مشابه با sheet2 نیست.

عملگر Like

این عملگر، دو عبارت رشته‌ای را مقایسه می‌کند تا ببیند که آیا کاملاً با هم تطبیق دارند یا خیر:

```
Option Compare Text
Sub test( )
        MsgBox "RICHARD" Like "richard"
End Sub
```

اگر دستور Option Compare در بخش declarations برابر با Text (متن) تنظیم شده باشد،خروجی کد فوق True است اما اگر برابر با Binary تنظیم شده باشد، خروجی کد فوق False خواهد بود.

می‌توانیم از کاراکترهای عمومی هم استفاده کنیم. کاراکتر؟ دلالت بر یک کاراکتر و کاراکتر * دلالت بر یک رشته کاراکتر دارد. کاربرد کاراکترهای عمومی در برنامه‌نویسی دقیقاً شبیه کاربرد آن‌ها در عملیات جستجوی ویندوز (Search) است. جدولی از کاراکترهای عمومی را در این‌جا ارایه کرده‌ایم:

جدول ۳-۶

کاراکتر	مفهوم
?	یک کاراکتر
*	چند کاراکتر
#	یک رقم (0-9)
[charlist]	یک کاراکتر از charlist
[!charlist]	کاراکتری که در charlist نباشد

خروجی مثال‌های بعد، True است:

```
MsgBox "RICHARD" Like "ri? hard"

MsgBox "RICHARD" Like "ric*"
```

فصل هفتم

اشکال زدایی

وقتی کدی را می‌نویسیم ممکن است خطاهایی به‌وجود آید که باعث خرابی برنامه، وقفه برنامه یا بروز نتایج غیرمنتظره شود. این عرصه‌ای است که برنامه‌نویسان ماهر را از افرادی که تنها کدی تایپ می‌کنند، مجزا می‌کند.

انواع خطاها

در هنگام نوشتن کدها، خطاها به سادگی بروز می‌کنند. این بخش، مثال‌هایی از انواع خطاهایی را که ممکن است با آن‌ها مواجه شوید، ارایه می‌کند.

خطاهای کامپایل یا ترجمه (Compile Errors)

خطاهای ترجمه از کدهایی که درست ساختاربندی نشده‌اند به‌وجود می‌آیند. مثلاً ممکن است از یک ویژگی یا متد استفاده کنید که برای شئ خاصی قابل استفاده نباشد یا این‌که برای حلقه For از Next استفاده نکنیم یا از یک If بدون Endif استفاده نماییم.

وقتی کدی را اجرا می‌کنیم، کامپایلر ابتدا به سراغ کد رفته و این نوع خطاها را کنترل می‌کند. اگر هر کدام از این خطاها را پیدا کند، کد اجرا نشده و یک پیغام خطا به نمایش در می‌آید که به نخستین خطاهای یافت شده در برنامه اشاره می‌کند. توجه کنید که ممکن است در یک رویه چندین خطای کامپایل وجود داشته باشد اما فقط اولین آن‌ها علامت‌گذاری می‌شود. ممکن است خطای نخست را تصحیح کرده و فکر کنید که کد را تصحیح کرده‌اید و سپس به رویه برگردید و باز هم با یک خطای دیگر مواجه شوید. بهترین راه برای عدم مواجه شدن با این وضعیت، تبعیت از قوانین کدنویسی در ابتدای کار است.

در ادامه، انواع خطاهایی را که در هنگام کامپایل کد با آن‌ها مواجه می‌شویم، می‌آوریم. معمولاً این خطاها را خطاهای فرمان کامپایل یا خطاهای زمان طراحی می‌نامیم.

خطاهای زمان اجرا

این‌ها خطاهایی هستند که در هنگام اجرای برنامه رخ می‌دهند. برای مثال می‌توانیم به باز کردن فایلی که وجود ندارد یا تقسیم یک عدد بر صفر اشاره کنیم. این خطاها می‌توانند یک پیغام خطا ایجاد کرده و برنامه را متوقف سازند. این خطاها در فرمان کامپایل کشف نمی‌شوند چون از هیچ قانون برنامه‌نویسی تبعیت نمی‌کنند اما باعث می‌شوند کد اجرا نشود.

خطاهای منطقی(Logic Errors)

این خطاها زمانی بروز می‌کنند که برنامه شما به صورت دلخواه‌تان عمل نکند. کد می‌تواند صحیح باشد و هیچ خطایی هم ایجاد نشود اما جوابی که به‌دست می‌آید و آن‌چه رخ می‌دهد صحیح و مورد قبول ما نیست. برای مثال، فرض کنید کاربر، دو کارپوشه را باز کرده و در هر کدام، یک کاربرگ با نام Sheet1 باز شده است. کد شیئی را در یک ماژول می نویسید که به Sheet1 ارجاع می‌شود اما مشخص نمی‌کنید که Sheet1 در کدام کارپوشه واقع شده است. این مورد یکی از خطاهای منطقی است.
یافتن این خطاها از همه سخت‌تر است و حتی با استفاده از تمام ابزارهای اشکال زدایی مدرن، باز هم یافتن آن‌ها مستلزم صرف زمان زیادی است.

حالت زمان طراحی[1]، حالت زمان اجرا[2] و حالت وقفه[3]

در هنگام کار روی یک برنامه کاربردی در VBA ، ممکن است در سه حالت قرار داشته باشیم:

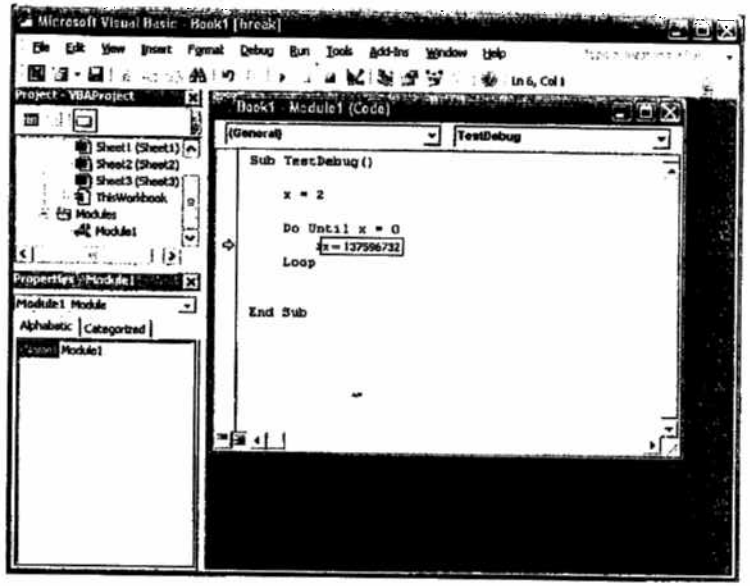
- زمانی طراحی (Design time) : زمانی است که روی کد یک برنامه کاربردی یا طراحی یک فرم کار می‌کنیم.
- زمان اجرا (runtime) : زمانی است که کد یا فرمی را اجرا می‌کنید. روی نوار عنوان صفحه VBA کلمه "running" وجود دارد و در زمان اجرا، هر چند می‌توانیم کد را ببینیم اما امکان تغییر آن وجود ندارد.
- وقفه(Break) : اگر در طی زمان اجرا، کلید CTRL-BREAK را بزنیم، اجرای کد متوقف شده و یک کادر پیغام محاوره‌ای با پیغام خطای "Code execution has been interrupted" (اجرای کد متوقف شده است) به نمایش درمی‌آید. با کلیک روی دکمه Debug می‌توانیم به پنجره کدنویسی برویم.

1- Design Time
2- Runtime
3- Break Mode

با کلیک روی Debug به حالت مرور کد می‌رویم که حالت اشکال زدایی نیز نامیده می‌شود. در این‌جا می‌توانیم کد را ببینیم. قسمتی که باعث توقف اجرای کد شده را هم به رنگ زرد می‌بینید و در واقع همان بخشی است که باعث بروز مشکل شده است. می‌توانیم مکان‌نما را روی هر متغیری که در آن‌جا قرار دارد بگذاریم و بلافاصله مقدار متغیر در آن بخش به نمایش درمی‌آید. هم‌چنین می‌توانیم با تغییر مکان‌نمای کنار خط زرد رنگ، نقطه اجرای برنامه را به جای دیگری منتقل کنیم.
برنامه ساده زیر را امتحان کنید.

```
Sub Test_Debug ()
x = 2
Do Until x = 1
      x = x + 1
Loop
End Sub
```

وقتی این کد را اجرا کنیم اجرای برنامه هرگز متوقف نمی‌شود چون x هرگز مساوی 1 نمی‌شود. کلید CTRL-BREAK را بزنید تا پنجره خطا ظاهر شود. روی Debug کلیک کنید تا به حالت مرور کد در پنجره کدنویسی بروید. مکان‌نما را روی x قرار دهید تا مقدار آن نمایش داده شود (تصویر ۱-۷).
با کلیک روی نماد Run از نوار ابزار (نماد مثلثی شکل) یا فشردن کلید F5 می‌توانید اجرای کد را مجدداً شروع کنید و اجرای کد از جایی‌که متوقف شده بود، دوباره آغاز می‌شود. یک پیکان زرد رنگ در سمت چپ کد ظاهر می‌شود که نقطه اجرای فعلی را نمایش می‌دهد. سعی کنید پیکان زرد رنگ را به یک نقطه جدید (مثلاً خط x = 2) بکشید و سپس مجدداً کد را اجرا کنید. این کار پس از تغییر یک کد به دلیل بروز یک خطا و برای شروع و اجرای مجدد یک حلقه یا شرط If مفید است.



تصویر ۱-۷ مثالی از تماشای لحظه‌ای اجرای کد

نقاط وقفه

می‌توانیم یک نقطه وقفه (Breakpoints) به کد اضافه کنیم تا در صورتی که کد به آن خط رسید، اجرای آن متوقف شود. سپس می‌توانیم از دستورات پله‌ای یا مرحله‌ای استفاده کنیم تا در زمان خاصی، اجرای کد به آن خط انتقال یابد. برای بررسی متغیرها از پنجره مرور کد به موازات رسیدن آن‌ها به آن بخش استفاده می‌کنیم.

می‌توانیم با استفاده از کلید F9 یا انتخاب Debug | Toggle Breakpoint از منو، نقاط وقفه را فعال یا غیر فعال کنیم. این‌کار باعث می‌شود در ستون سمت چپ پنجره کد، یک دایره قهوه‌ای ظاهر شود.

اگر مثال حلقه را اجرا کنید و در دستور Loop، یک نقطه وقفه قرار دهید، خواهید دید که هر بار کلید F5 را می‌فشارید یا روی نماد اجرا کلیک می‌کنید، اجرای کد با رسیدن به دستور Loop متوقف می‌شود. می‌توانیم پنجره Debug را نیز باز کنیم تا عبارت‌ها را ببینیم و کاری که انجام می‌دهند، بررسی نماییم.

استفاده از دستورهای توقف(Stop)

وارد کردن یک دستور Stop در کد، شبیه وارد کردن یک نقطه وقفه است؛ به استثنای این‌که دستور توقف در خود کد قرار می‌گیرد. وقتی VBA با یک دستور Stop مواجه شود اجرای کد، متوقف شده و به حالت وقفه می‌رود. اگر چه دستورات Stop شبیه نقاط وقفه عمل می‌کنند اما فعال و غیر فعال کردن آن‌ها مشابه هم نیست.

اگر نقاط وقفه را با کلید F9 تنظیم کنیم، وقتی از پروژه خارج می‌شویم و دوباره آن‌را بارگذاری می‌کنیم همه نقاط وقفه پاک می‌شوند. اما دستورات Stop بخشی از کد محسوب شده و تنها وقتی پاک می‌شوند که برنامه نویس، آن‌ها را از کد پاک کند یا یک علامت نقل قول (') در ابتدای آن‌ها قرار دهیم تا تبدیل به یک عبارت توضیحی شوند. وقتی کد به درستی کار کرد تمام دستورات Stop را پاک کنید.

اجرای قسمت‌های گزینش‌شده کد

اگر دقیقاً بدانیم دستوری که باعث بروز خطا می‌شود در کجا قرار دارد، یک نقطه وقفه تکی می‌تواند به سادگی قسمت مشکل دار را پیدا کند. با این وجود، این احتمال وجود دارد که به صورت تقریبی بدانیم نقطه بروز خطا در کدام قسمت کد قرار دارد.

می‌توانیم یک نقطه وقفه در جایی‌که می‌خواهیم کنترل کد آغاز شود قرار دهیم و سپس به صورت خط به خط، کد را اجرا کنیم تا ببینیم هر دستور چه کاری انجام می‌دهد. هم‌چنین می‌توانیم یک دستور را نادیده بگیریم یا این‌که اجرا را از نقطه دیگری آغاز نماییم.

مرور تک مرحله‌ای کد

مرور تک مرحله‌ای کد به ما اجازه می‌دهد تا هر بار تنها یک دستور را اجرا کنیم. می‌توانیم مکان‌نما را در هر جای کد روی یک متغیر قرار دهیم تا وضعیت آن متغیر را ببینیم. هم‌چنین می‌توانیم برای مرور مقادیر متغیرها از پنجره Debug استفاده کنیم.

می‌توانیم از طریق منوی Debug | Step Into یادکمه F8، کد را به صورت تک مرحله‌ای مرور کنیم. هم‌چنین می‌توانیم کد را از موقعیت مکان‌نما به بعد اجرا نماییم. با ماوس روی یک خط کد کلیک کرده و سپس دکمه CTRL-F8 را فشار دهیم. کد تنها از محل کلیک ماوس به بعد اجرا می‌شود. توجه کنید که ماوس باید روی یک خط قابل اجرای کد قرار داشته باشد و روی خط خالی کد نباشد. اگر چند دستور در یک خط قرار داشته باشند و با کاراکتر : از هم جدا شده باشند نیز می‌توانید آن‌ها را به صورت یک به یک مرور کنید:

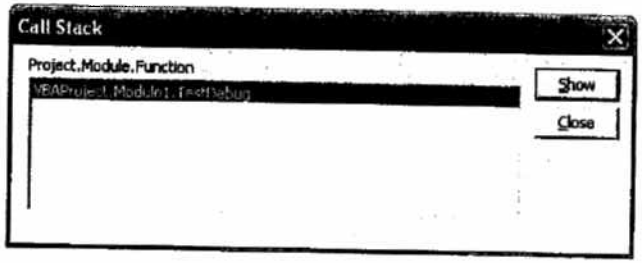
```
Temp = 4: If  temp = 3 Then Exit Sub
```

مرحله‌ای کردن رویه

اگر زیرروال یا تابعی در کد وجود داشته باشد که فراخوانده شده باشد ممکن است نخواهیم کل رویه را خط به خط مرور کنیم. ممکن است قبلاً آن را آزمایش کرده و از عملکرد آن راضی باشیم. اگر از کلید F8 برای مرحله‌ای کردن کد استفاده کنیم ، هر بار یک مرحله از کد را مرور می‌کنیم. این‌کار برای کدی که از قبل می‌دانیم جواب می‌دهد زمان زیادی تلف می‌کند. اگر از کلید SHIFT-F8 استفاده کنیم آن‌گاه زیرروال به عنوان یک دستور تلقی شده و به صورت مرحله به مرحله و خط به خط مرور نمی‌شود.

کادر محاوره Call Stack

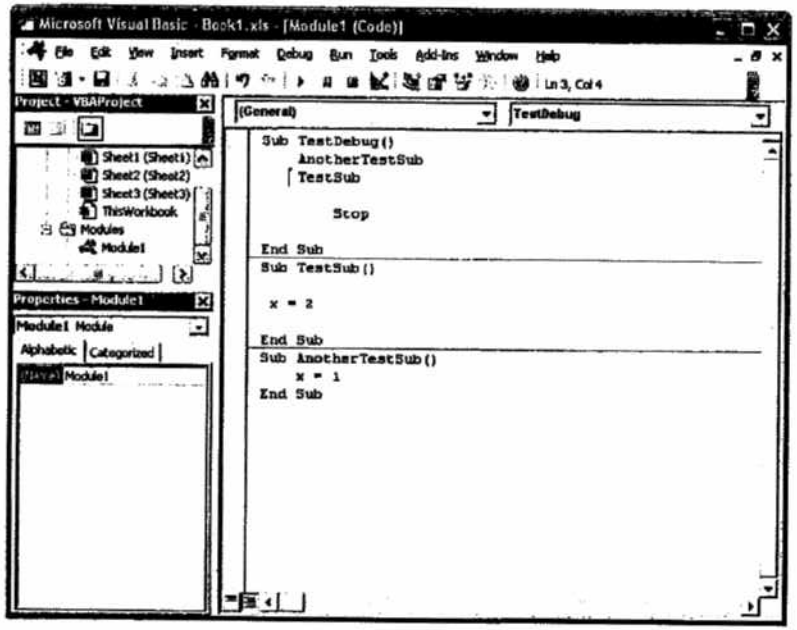
این کادر که در تصویر زیر نمایش داده شده است، لیستی از فراخوان‌های رویه فعال را نشان می‌دهد (یعنی فراخوان‌هایی که آغاز شده‌اند اما هنوز تکمیل نشده‌اند):



تصویر ۲-۷

می‌توانیم این کادر محاوره‌ای را با استفاده از کلید CTRL-L نمایش دهیم. این کادر محاوره‌ای به ما کمک می‌کند تا عملیات فراخوانی رویه‌ها را ردگیری کنیم؛ به‌خصوص اگر آن‌ها به صورت تو در تو قرار داشته باشند و یک رویه، رویه دیگری را فرا بخواند.

اولین فراخوان رویه فعال در بالای لیست قرار دارد و فراخوان‌های رویه بعدی به ادامه آن اضافه می‌شوند (تصویر ۳-۷). با انتخاب یک رویه در تصویر و کلیک روی Show، دستور فراخوانی آن رویه به نمایش درمی‌آید. این رویه با یک پیکان سبز رنگ مشخص می‌شود.

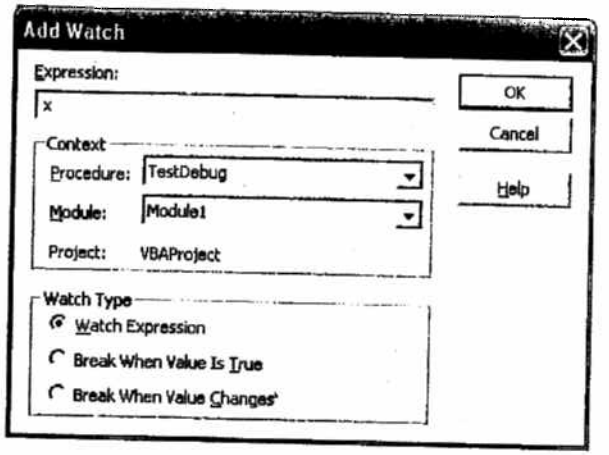


تصویر ۳-۷ چگونگی فراخوانی یک زیر روال و نمایش رویه‌ای که آن را فرا خوانده است.

پنجره Debug

این پنجره اجازه می‌دهد تا نگاهی به متغیرها و ویژگی‌های خاص موجود در کد بیندازیم تا ببینیم آن‌ها در هنگام اجرای برنامه، حاوی چه مقادیری هستند. این کار با اجازه دادن به شما برای تحلیل یک متغیر یا ویژگی کد که دارای مقدار نادرست است و تعیین منشأ بروز مشکل، در رفع اشکال‌های بعدی به ما کمک می‌کند.

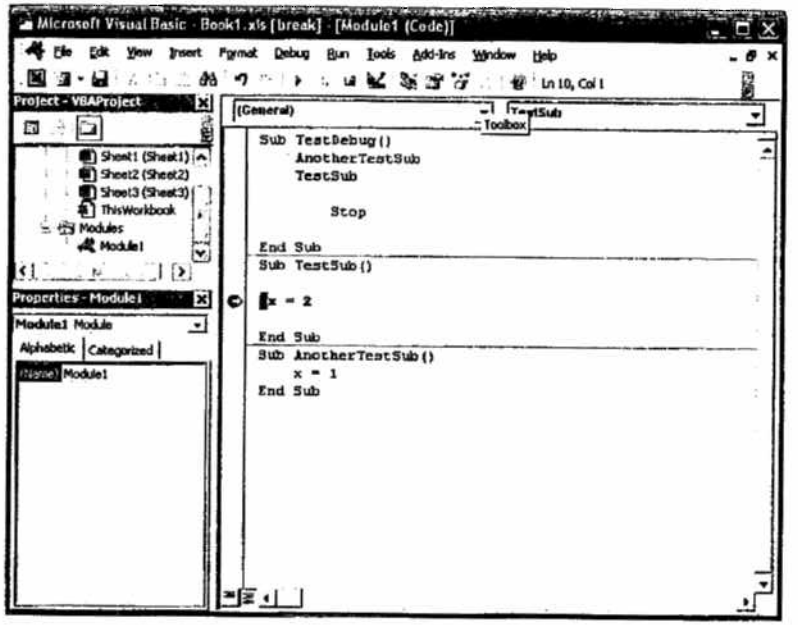
با انتخاب Debug | Add Watch که در تصویر ۴-۷ نمایش داده شده است می‌توانیم پنجره Debug را تنظیم کنیم.



تصویر ۴-۷ اضافه کردن watch به یک متغیر

متغیر یا عبارتی را که می‌خواهیم کنترل شود در کادر محاوره‌ای Expression وارد می‌کنیم. برای مثال اگر یک متغیر x داریم و می‌خواهیم مسیر مقدار آن را حفظ کنیم، x را در کادر وارد می‌کنیم. جزییات به صورت خودکار درج می‌شوند که البته قابل ویرایش و تغییر هم هستند و نیز می‌توانیم مشخص کنیم که آیا وقتی مقدار متغیر True (غیر صفر) است یا وقتی مقدار آن تغییر می‌کند مایل به ایجاد وقفه در کد هستیم یا خیر.

روی OK کلیک کنید تا پنجره Debug همراه با جزییات متغیر انتخاب شما ظاهر شود (تصویر ۵-۷).



تصویر ۵-۷ کنترل مقدار یک متغیر از طریق پنجره watch

وقتی برنامه به نقطه توقف می‌رسد، مقدار موجود در پنجره Debug به روز در آورده می‌شود. به موازات این‌که هر مرحله را به پایان می رسانیم، این پنجره هم به روز در آورده می‌شود. می‌توانیم یک عبارت را مشخص کنیم (highlight)، روی آن کلیک کرده و سپس دکمه DELETE را بزنیم تا حذف شود. البته به جای زدن این دکمه می‌توانیم روی آن کلیک راست کرده و گزینه DELETE را انتخاب نماییم.

می‌توانیم یک عبارت را انتخاب کرده و با کلیک راست روی آن و انتخاب گزینه Edit، آن‌را ویرایش کنیم.

وقایعی‌که می‌توانند درهنگام اشکال‌زدایی، مشکلاتی را موجب شوند

وقایع خاصی وجود دارند که می‌توانند مشکلاتی برای برنامه نویس و فردی که عملیات رفع اشکال را اجرا می‌کند به‌وجود آورند. این وقایع باعث پیچیده شدن و سردرگمی فرآیند اشکال زدایی می‌شوند. این وقایع را در بخش‌های بعدی بررسی خواهیم کرد:

Mouse Down (فشردن دکمه ماوس)

Mouse Down، رویدادی است که وقتی کاربر یکی از دکمه‌های ماوس را فشار می‌دهد راه‌اندازی می‌شود. این رویداد قبل از آن‌که دکمه به جای خود بازگردد، رخ می‌دهد. اگر در این نقطه کد را قطع کنید، رویداد mouse up (برگشت ماوس به سر جای اولش) را در اختیار نخواهید داشت. رویداد mouse up تنها زمانی رخ می‌دهد که یک دکمه ماوس را فشار داده و آن را رها کنیم.

Key Down

رویدادی است که وقتی کلیدی از صفحه کلید را می‌زنیم راه اندازی می‌شود. این رویداد قبل از آن‌که دکمه به جای خود بازگردد، رخ می‌دهد. اگر در این نقطه کد را قطع کنید، رویداد Key up (برگشت دکمه به سر جای اولش) را در اختیار نخواهید داشت. رویداد Key up تنها زمانی رخ می‌دهد که یک دکمه صفحه کلید را فشار داده و آن‌را رها کنید.

Got Focus/Lost Focus (تمرکز/ عدم تمرکز)

این رویداد زمانی رخ می‌دهد که کاربر روی فرم یا یک کنترل خاص روی فرم کلیک می‌کند تا روی آن کنترل متمرکز شود. اگر کد را در این نقطه قطع کنید، ممکن است نتایج غیر مترقبه‌ای به‌دست آورید.

استفاده از کادرهای پیغام در اشکال زدایی

به غیر از روش‌هایی که تا این‌جای کار بررسی شدند، روش‌های دیگری هم وجود دارند که از ورود خطاها به کد ما جلوگیری می‌کنند. هرچند آن‌ها ممکن است زیاد دلچسب و خوشایند نباشند اما حتماً جواب می‌دهند. برای مثال، کادرهای پیغام همیشه برای این‌که شما را از آن‌چه رخ می‌دهد آگاه سازند مفید هستند. آن‌ها می‌توانند مقدار یک متغیر را نمایش دهند یا حتی مقدار چندین متغیر را با الحاق

آن‌ها به یکدیگر نشان دهند. وقتی یک رویه بزرگ داریم و نمی‌دانیم اشکال‌ها در کجا قرار دارند استفاده از کادرهای پیغام بسیار مفید است. برای مثال، فرض کنید که یک قسمت بزرگ کد به ظاهر اجرا می‌شود اما ما نمی‌دانیم که اشکال در کدام منطقه رخ می‌دهد و این قسمت از کد در حین اجرا باعث وقفه (hang) تمام کد می‌شود و حتی به CTRL-BREAK هم جواب نمی‌دهد. شاید تا این‌جای کار، کد به درستی جواب داده است اما با موارد خاصی برخورد کرده که باعث وقفه در اجرای آن شده است.

سؤال این است که تا قبل از این جریان، کد چگونه عمل می‌کرد؟ می‌توانیم تمام کد را به صورت مرحله به مرحله کنترل کنیم اما اگر رویه ما بزرگ باشد، این کار به زمان بسیار زیادی نیاز دارد. قرار دادن کادرهای پیغام در نقاط حساس کد می‌تواند به ما کمک کند تا متوجه شویم که وقفه در کجای کد رخ داده است. باید مطمئن شویم که تمام کادرهای پیغام، مطالب با معنایی نمایش می‌دهند (مانند "OK1"، "OK2"و"OK3") سپس می‌توانیم با کمک این پیغام‌ها کشف کنیم که وقفه در کدام مرحله از اجرای کد رخ داده است. پس از یافتن اشکال‌های برنامه و اطمینان از صحت عملکرد کد، حتماً باید پیش از ارایه کد به کاربران، این پیغام‌های اضافی از صفحه اصلی حذف شوند. هم‌چنین می‌توانیم متغیرها را نیز به یکدیگر الحاق کنیم:

```
Sub test_for()
For n = 1 To 4
    For m = 2 To 8

        MsgBox m & " xxxxx " & n

    Next m
Next n

End Sub
```

این کادر پیغام، مقادیر متغیرهای m و n را نمایش می‌دهد که توسط چند کاراکتر "x" از هم جدا شده‌اند. هدف از به‌کارگیری کاراکترهای x این است که اگر مقادیر m یا n ، یا کاراکتر فاصله یا مقدار تهی بودند، کاربر متوجه این مورد بشود.

اگر بخواهیم در هنگام اجرای برنامه به صورت مستمر، مقدار متغیرها خوانده شود، روش‌هایی برای انجام آن وجود دارد. اشکال زدایی امکان این کار را مهیا نمی‌کند اما با کمی ابتکار می‌توانیم راه حلی برای آن پیدا کنیم. می‌توانیم با استفاده از کدنویسی، خصوصیات caption (قسمت عنوان) را هم روی شئ Application (صفحه گسترده)و هم شئ UserForm تغییر دهیم. در این حالت فرض می‌کنیم از قبل، UserForm تعریف شده است:

```
Sub test_for()

For n = 1 To 4

    For m = 2 To 8

        Application.Caption = n & " xxxx " & m

        UserForm.Caption = n & " xxxx " & m

    Next m

Next n
End Sub
```

این کد به موازات اجرای برنامه، مقدار متغیرها را در قسمت عنوان پنجره (caption) نمایش می‌دهد. بسته به میزان سرعت تغییر متغیرها و میزان طولانی بودن رویه، قادر خواهیم بود تا الگوی عملیات در حال اجرا را مشاهده کنیم و ببینیم در هر مرحله چه اتفاقی رخ می‌دهد.

پس از آن می‌توانیم با مساوی قراردادن نوار عنوان برنامه با یک رشته تهی، برنامه کاربردی را مجدداً راه‌اندازی کنیم:

```
Application.Caption = " "
```

پرهیز از بروز اشکال و خطا

طراحی و برنامه‌ریزی دقیق یک برنامه کاربردی از اهمیت زیاد برخوردار است و به کاهش بروز اشکال‌ها و خطاها کمک می‌کند. در بیشتر موارد، بهتر است که یک برنامه بزرگ را به بخش‌های کوچک‌تری تقسیم کنیم تا هم کدنویسی و هم آزمایش آن‌ها ساده‌تر انجام شود. همیشه مطمئن شوید که توضیحات موجود در کد در بین کاراکترهای (') قرار گرفته‌اند. دقت کنید که چنین توضیحاتی همیشه در کد به رنگ سبز نمایش داده می‌شوند. اگر برنامه نوشته شده، پیچیده باشد اغلب، حتی پس از چند روز از خاتمه کدنویسی هم بسیار مشکل خواهد بود که به سراغ آن برویم و تمام مراحل کار هنوز در حافظه ما باقی مانده باشد و بتوانیم تعیین کنیم که کد، چه عملی انجام می‌دهد. حتی برنامه‌نویسان حرفه‌ای اعتقاد دارند که اگر به سراغ برنامه‌ای که چند وقت قبل نوشته‌اند بروند، در درک آن‌چه کد انجام می‌دهد با مشکل مواجه می‌شوند. در این مواقع، توضیحات (Comments) هستند که کمک مورد نیاز را ارایه می‌کنند.

دانستن این‌که تمام متغیرها چه مقداری نمایش می‌دهند و این‌که هر تابع چه کاری انجام می‌دهد بسیار مهم است. وقتی برنامه رشد کرده و پیچیده‌تر می‌شود، ارایه توضیحات نیز اهمیت بیشتری پیدا می‌کند. البته اگر به حافظه خود ایمان دارید می‌توانید چنین مرحله‌ای را حذف کنید. اما این‌کار را توصیه نمی‌کنیم.

فصل هشتم

خطاها و تابع Error

خطاهای زمان اجرا (Runtime Error) به سادگی در برنامه نفوذ می‌کنند و برنامه نویس نیز دخالتی در این امر ندارد. کاربر کاری خارج از حوزه برنامه انجام می‌دهد و باعث ایجاد پیغام خطایی می‌شود که موجب توقف اجرای کد می‌گردد. این امر ممکن است زمانی رخ دهد که به یک منبع داده خارجی مثل یک پایگاه داده دسترسی پیدا می‌کنیم. هم‌چنین ممکن است کاربر کاری انجام دهد که برنامه‌نویس پیش بینی آن را نکرده است.

هر چقدر در هنگام نوشتن برنامه دقت کنیم تا تمام شرایط را در نظر بگیریم باز هم همیشه کاربری وجود دارد تا کاری انجام دهد که هرگز تصور آن را هم نکرده‌ایم و خارج از مجموعه تمام مواردی است که در برنامه در نظر گرفته‌ایم. برای مثال ممکن است کاربر در هنگام وارد کردن نام از علامت آپوستروف استفاده کند (O'Brien). چنین کاراکتری در یک رشته پرس و جوی SQL باعث بروز مشکل می‌شود. مثال دیگر، خواندن یک فایل از یک دیسک است. برنامه نویس ممکن است به کاربر اجازه دهد تا برای خواندن یک فایل، درایوی مشخص کند. فرض کنیم درایوهای شبکه را به عنوان مرجع معرفی می‌کنیم اما کاربر، فلاپی را به عنوان مرجع مشخص می‌کند. این امر باعث بروز خطایی می‌شود که اجرای برنامه را متوقف می‌کند. چنین قضیه‌ای کاربر را بسیار ناراحت کرده و اعتماد او را به برنامه سلب می‌نماید. پس لازم است خطاها را پیداکرده و آن‌ها را اصلاح کنیم.

روی فرم‌های محاوره‌ای متداول (به فصل ۱۰ مراجعه شود) از عملیات بررسی خطا استفاده می‌کنیم تا نشان دهیم که دکمه Cancel کلیک شده است. برای این‌که آزمایش کنیم آیا کاربر روی دکمه Cancel کلیک کرده است باید مقدار خصوصیت CancelError را True و سپس در یک دستور On Error قرار دهیم تا کد متوجه شود در صورت بروز خطا به کجا برود.

هیچ دیسکی در فلاپی نگذارید و کد ساده زیر را با فشار دادن کلید F5 اجرا کنید:

```
Sub Test_Error( )

        temp = Dir("a:\*.*")

End Sub
```

اجرای این کد باعث ایجاد پیغام خطایی می‌شود که اعلام می‌دارد درایو A آماده نیست. به عبارت ساده، برنامه شما خراب شده و تا زمانی که مشکل برطرف نشود جواب نخواهد داد و این اصلاً از نقطه نظر کاربر جالب نیست.

می‌توانیم کد فوق را به سادگی با تابع خطایاب ساده زیر اصلاح کنیم:

```
Sub Test_Error( )

        On Error GoTo err_handler
                temp = Dir("a:\*.*")
        Exit Sub
        err_handler:
         MsgBox "The A drive is not ready" & "  " & Err.Description
End Sub
```

خط نخست کد فوق با استفاده از دستور On Error، تابعی ایجاد می‌کند که در هنگام بروز خطا آن‌را نادیده گرفته و به err_handler اشاره می‌کند که برچسبی در زیر خط کد Exit Sub است و در صورت بروز خطا فعال می‌شود. هدف از برچسب، تعریف بخشی از کد است که می‌توانیم با استفاده از دستور GoTo به آن‌جا بپریم.

خط کد خواندن درایو A مثل قبل است و تنها پس از آن Exit Sub را قرار داده‌ایم چون اگر همه چیز مرتب باشد مطمئناً نمی‌خواهیم کد، تابع err_handler را اجرا کند.

اگر در هر نقطه پس از خط کد On Error، خطایی رخ دهد اجرای کد به خط کد تابع err_handler رفته و کادر پیغامی به نمایش در می‌آید که می‌گوید درایو A آماده نیست. با وجود این ممکن است متوجه شده باشید که در صورت بروز هر نوع خطایی (نه فقط آماده نبودن درایوA) اجرای کد به خط تابع err_handler می‌پرد. مثلاً اگر خطا به دلیل غلط تایپ کردن دستوری رخ دهد جالب نیست که تابع err_handler و پیغام مربوط به آن به نمایش درآید.

خوشبختانه این امکان وجود دارد که خطا را بررسی کنیم تا متوجه شویم چه مشکلی رخ داده است. هم‌چنین می‌توانیم از شئ Err استفاده کنیم تا شرحی از خطا به دست آوریم و این شرح را به پیغام خود وصل نماییم تا کاربر بهتر در جریان امور قرارگیرد.این کار را با استفاده از تابع Err انجام می‌دهیم. این تابع عددی را بر می‌گرداند که با خطای زمان اجرایی که رخ داده، مرتبط است. می‌توانیم به صورت زیر، مثال فوق را اصلاح کنیم:

```
Sub Test_Error( )

        On Error GoTo err_handler
         temp = Dir("a:\*.*")
        Exit Sub
```



```
        err_handler
        If Err.Number = 71 Then
                MsgBox "The A drive is not ready"
        Else
                MsgBox "An error occurred"
        End if
End Sub
```

در مثال اول دیدیم که عدد 71 به عنوان عدد مربوط به پیغام "Drive not ready" قرار می‌گیرد. برنامه نگاهی به Err (یک متغیر سیستمی که آخرین شماره خطا را در خود نگه می‌دارد) انداخته و کنترل می‌کند که آیا مقدار آن 71 است یا خیر. اگر چنین بود کادر پیغام "Drive not ready" به نمایش در می‌آید و در غیر این صورت کادر پیغام "An error occurred" نشان داده می‌شود.

دستور Resume

با افزودن این دستور را می‌توان به کد، اجرای کد به جایی برمی‌گردد که خطا در آنجا رخ داده است. این دستور فرصتی در اختیار کاربر قرار می‌دهد تا در عملیات مداخله کند (مثلاً یک فلاپی در درایو A قرار دهد و سپس کد بتواند محتویات فلاپی را مرور کند). شما می‌توانید این دستور را در روال رفع خطای کد خود قرار دهید به طوری‌که به دنبال انجام عملیاتی از جانب کاربر، اجرای کد از محلی که خطا رخ داده است مجدداً از سر گرفته شود:

```
Sub Test_Error( )

        On Error GoTo err_handler
        temp = Dir("a:\*.*")
        Exit Sub
        err_handler
        If Err.Number = 71 Then
          MsgBox "The A drive is not ready"
          Else
                MsgBox "An error occurred"

        End if
        Resume
End Sub
```

می‌توانیم دستور Next To Resume را هم به‌کار ببریم. این دستور از روی خط کد که خطایی را ایجاد می‌کند می‌پرد بنابراین می‌توان گفت که خطا را نادیده می‌گیرد:

```
Sub Test_Error( )
```

```
    On Error Resume Next
    temp = Dir("a:\*.*")
End Sub
```

در کد فوق اگر دیسکی در درایو A وجود نداشته باشد باز هم برنامه به صورت صحیح ادامه می‌یابد و این به خاطر On Error Resume Next است که خط کد ایجاد کننده خطا را نادیده می‌گیرد.
Resume Next برای برخورد با خطاها در هنگام وقوع‌شان مفید است اما باعث می‌شود اشکال زدایی کد بسیار مشکل شود. در ادامه برنامه ممکن است داده‌های نادرست یا مجهولی به خاطر نادیده گرفتن قسمت‌های قبلی کد (به دلیل وقوع خطا) به‌وجود بیایند. ممکن است برنامه ظاهراً درست خاتمه پذیرد اما در حقیقت چنین نباشد چون خطاهای پنهان یا داده‌های نادرستی در برنامه وجود دارد. این امر بدان خاطر است که هر بار خطایی رخ می‌دهد اجرای برنامه از آن مرحله می‌پرد. پس اگر از دستور On Error Resume Next استفاده می‌کنید مطمئن شوید برای اطمینان از صحت عملکرد کد، تمام ورودی‌ها و خروجی‌ها به آن را کنترل کرده‌اید.

نکات احتیاطی در ردگیری خطاها

وقتی از دستور On Error در کد خود استفاده می‌کنید، این تله در سراسر رویه به صورت فعال باقی می‌ماند مگر آن‌که خودمان آن‌را غیر فعال کنیم. همان‌طور که قبلاً دیدیم می‌توان روالی برای کنترل این‌که آیا درایو A: دیسکی دارد یا خیر، ایجاد کنیم تا بر طبق وضعیت موجود، اعمالی را انجام دهد. اما این نکته مهم است که وقتی از قسمت مشکوک کد گذشتیم، آن روال را غیر فعال کنیم. اگر چنین کاری انجام نشود تمام خطاهای بعدی در آن رویه از همان روال برای بر طرف کردن خطاها استفاده می‌کنند. این امر باعث می‌شود نتایج گیج کننده‌ای برای کاربر ارسال شود چون احتمالاً پیغام خطایی که در رابطه با یک خطا ایجاد می‌شود، اصلاً ربطی به آن ندارد. مثلاً ممکن است پیغام بعدی درباره تقسیم یک عدد بر صفر باشد اما هنوز پیغام "Drive not ready" به نمایش در بیاید.
اگر On Error Resume Next به‌کار گرفته شود و غیر فعال هم نشود هر نوع خطایی ممکن است رخ دهد بدون آن‌که کاربر متوجه آن‌ها شود. می‌توانیم روال ردگیری خطا را به صورت زیر غیر فعال کنیم:

```
On Error Resume Next
On Error GoTo 0
```

دستور On Error Resume Next که قبلاً دیدیم تمام خطاها را نادیده می‌گیرد. دستور

On Error GoTo 0، روال ردگیری خطا را لغو می‌کند و اجازه می‌دهد تمام خطاها مثل روال معمولی خود به نمایش در بیایند. این دستور، On Error Resume Next را لغو می‌کند و همه چیز را به حالت نرمال برمی‌گرداند.

ایجاد خطاهای اختصاصی

چرا ممکن است بخواهیم خطاهای اختصاصی ایجاد کنیم؟ مهم‌تر از همه این است که می‌خواهیم کد بدون خطا داشته باشیم. چنین خطاهایی گاهی اوقات در هنگام آزمایش برنامه‌های کاربردی ما یا زمانی که می‌خواهیم شرط خاصی را که معادل با خطایی در زمان اجراست، کنترل کنیم سودمند هستند.
با دستور Error می‌توانیم خطایی را در کد خود ایجاد کنیم:

```
Sub Test_Error( )
    Error 71
End Sub
```

این کد، خطای "Drive not ready" را شبیه سازی می‌کند. می‌توانیم از Err.Raise(71) هم برای این کار استفاده کنیم. به علاوه با استفاده از کد زیر می‌توانید خطای آخر را مجدداً ایجاد کنید:

```
Error Err
```

فصل نهم

محاوره‌ها (Dialogs)

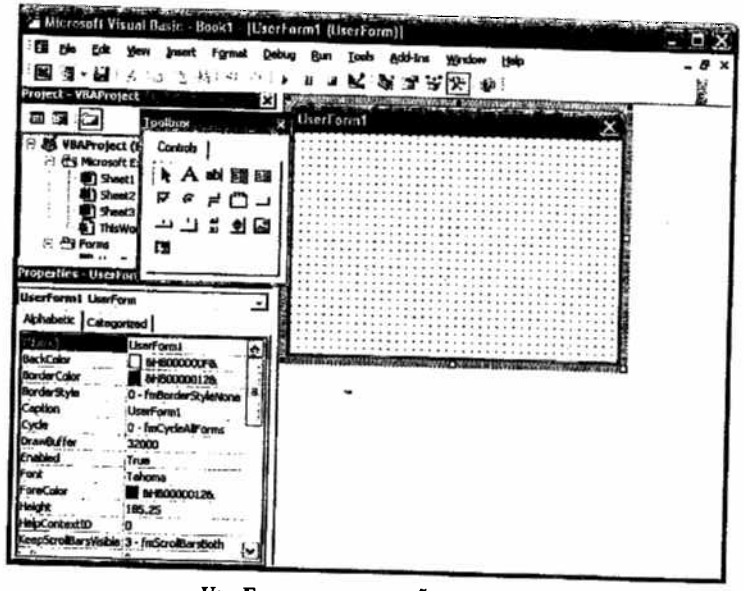
در فصل ۷ چگونگی استفاده از کادرهای پیغام را که نوع ساده‌ای از کادرهای محاوره‌ای محسوب می‌شوند، بررسی کردیم. استفاده از آن‌ها بسیار ساده است و روش مناسب و راحتی برای برقراری ارتباط با کاربر ارایه می‌کنند. آن‌ها می‌توانند برای جلب توجه کاربر، اجرای کد را متوقف کرده و به کاربر اجازه دهند تا انتخاب‌های ضروری را انجام دهد.

با وجود این اگر بخواهیم کارهای پیچیده‌تری، مانند درخواست از کاربر برای انتخاب یک کاربر یا تعیین مقدار برای چند پارامتر را انجام دهیم، تکلیف چه می‌شود؟ این کار با یک UserForm انجام می‌شود به طوری‌که کار با فرمی خاتمه می‌یابد که نمایی شبیه یک فرم استاندارد ویندوز دارد.

برای استفاده از UserForm باید ابتدا یکی از آن‌ها را در پروژه خود درج کنیم. می‌توانیم از منوی پنجره کد، با انتخاب Insert | UserForm، یک UserForm خالی را در پروژه وارد نماییم. در این حالت، پنجره صفحه نمایش باید شبیه تصویر ۱-۹ باشد.

فرم، یک صفحه خالی است که می‌توانیم کنترل‌ها را روی آن قرار دهیم. توجه کنید که در این حالت، یک پنجره جعبه ابزار (toolbox) نیز باز شده است که به ما اجازه می‌دهد کنترل‌هایی را انتخاب کنیم و روی فرم قرار دهیم (مانند کادرهای متنی، کادرهای انتخاب و ...).

هم‌چنین می‌توانیم در پنجره Properties، خصوصیات فرم را ببینیم. اگر این پنجره باز نبود می‌توانیم منوی View | PropertiesWindow, VBE را انتخاب کنیم یا دکمه F4 را فشار دهیم. UserForm مشابه با فرم‌های Visual Basic است و خصوصیات، رویدادها و متدهایی شبیه به آن دارد. UserForm، یک شئ محسوب می‌شود و در ساختار درختی VBA Project محلی را به خود اختصاص داده است.



تصویر ۱-۹ آماده سازی یک UserForm

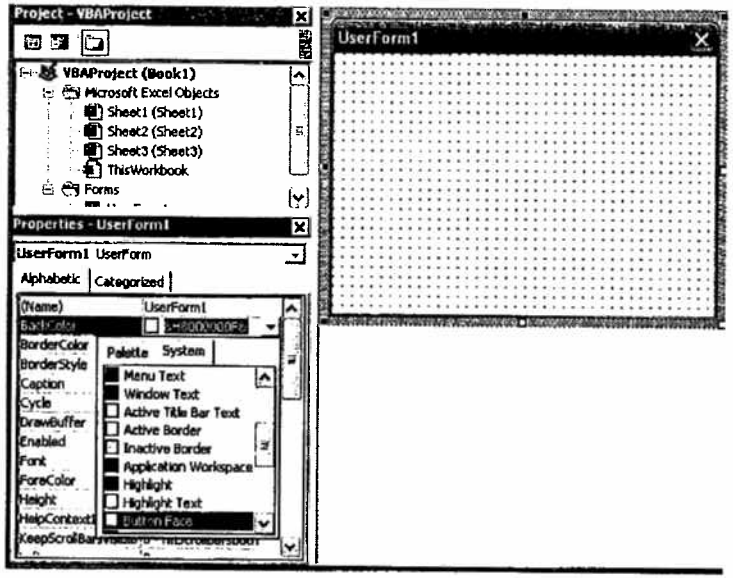
مواردی که در جدول ۱-۹ می‌آیند، خصوصیات اصلی فرم هستند که باید آن‌ها را بشناسیم:

جدول ۱-۹

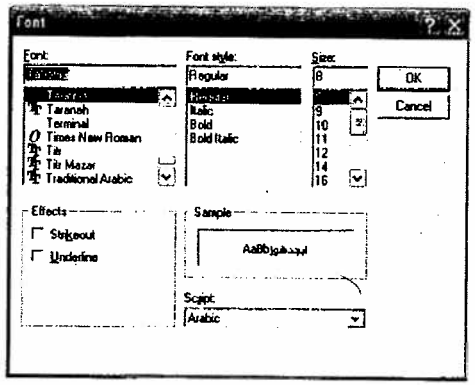
خصوصیت	شرح
Name	نامی که به‌وسیله آن، در کد به فرم ارجاع می‌کنیم، در حالت پیش‌فرض UserForm است که بهتر است به نام بهتر و با معناتری تغییر یابد.
BackColor	رنگ پس زمینه فرم شما را نشان می‌دهد. خاکستری (groy)، رنگ پیش‌فرض است که با نمای کلی دیگر اشیا نیز تطبیق دارد اما می‌توانید رنگ دلخواه خود را نیز انتخاب کنید. برای رنگ‌هایی به غیر از طیف سفید تا مشکی می‌توانید روی Pallete کلیک کرده و رنگ دلخواه خود را انتخاب کنید (تصویر ۲-۹).

خصوصیت	شرح
BorderColor	مانند BackColor است اما رنگ لبه‌های فرم را مشخص می‌کند.
BorderStyle	مشخص می‌کند که آیا فرم لبه دارد یا خیر.
Caption	عنوان فرم را که در نوار عنوان بالای فرم ظاهر می‌شود، مشخص می‌کند.
Enabled	معمولاً مقدار آن‌را True تنظیم می‌کنیم تا کاربران بتوانند با کنترل‌های روی فرم ارتباط برقرار کنند. اگر مقدار آن، False تنظیم شده باشد، همه چیز روی فرم به رنگ خاکستری (غیرفعال یا disable) در می‌آید و نمی‌توانیم از فرم استفاده کنیم. این خصوصیت زمانی مفید واقع می‌شود که پردازش در یک رویه، طول بکشد و نخواهیم کاربر در هنگام پردازش،کار دیگری روی فرم انجام دهد.
Font	خصوصیات Font (قلم) برای شئ form را تغییر می‌دهد (تصویر ۳-۹). به خاطر داشته باشید که این خصوصیت تنها روی نوع قلم متن چاپ شده روی خود شئ form تأثیر می‌گذارد و ربطی به اشیایی که روی فرم قرار می‌گیرند (مانند کادرهای متنی، کادرهای گزینه ها و ...) ندارد. البته آن اشیا به صورت مستقل دارای این خصوصیت هستند که با کلیک روی آن‌ها می‌توانیم از پنجره Properties ، این خصوصت را برای آن‌ها تغییر دهیم.
ForeColor	مثل خصوصیت BackColor عمل می‌کند و رنگ متنی را که روی فرم چاپ خواهد شد، تعیین می‌کند.
Height	می‌توانید مقدار آن‌را به کلیک کردن و نگه داشتن گیره‌هایی که روی لبه فرم قرار دارند و کشیدن آن‌ها تا رسیدن به اندازه دلخواه، تنظیم کنید.
Left	فاصله فرم را از سمت چپ نسبت به لبه پنجره Excel که فرم در آن‌جا ظاهر می‌شود نشان می‌دهد.
Locked	مشخص می‌کند که آیا امکان وارد کردن ورودی توسط کاربر در یک کنترل وجود دارد یا خیر. اگر فعال (enable) نباشد، کنترل به رنگ خاکستری در می‌آید. Value یا مقدار انتخابی برای این

خصوصیت	شرح
	خصوصیت،True و False است.
Picture	می‌توانید یک تصویر را به عنوان پس زمینه فرم مشخص کنید. روی کلمه (None) کلیک کنید،کادری با کاراکتر (...) در سمت راست ظاهر خواهد شد. می‌توانیم روی این کادر کلیک کنیم تا به کادر محاوره‌ای انتخاب فایل گرافیکی دست پیدا کنیم. فایل گرافیکی دلخواه را انتخاب کرده و روی OK کلیک کنید. برای حذف یک تصویر می‌توانید محتویات این خصوصیت را پاک کنید تا دوباره کلمه None در آن ظاهر شود.
PictureAlignment	یک منوی باز شو را نشان می‌دهد که می‌توانید تراز بندی تصویر را به دلخواه خود تنظیم کنید (مثلاً چپ چین،وسط چین و ...).
PictureSizeMode	یک منوی بازشو در اختیار شما می‌گذارد که ۳ گزینه درمورد چگونگی نمایش تصویر دارد: 0-Mode Clip : مقدار پیش‌فرض است. در این حالت، تصویر به اندازه نرمال به نمایش درمی‌آید و در صورت بزرگ بودن، برای آن‌که در فرم جا شود، لبه‌های آن بریده می‌شوند. 1-Mode Stretch : تصویر به صورت افقی و عمودی کشیده می‌شود تا در فرم جای گیرد. این‌کار ممکن است باعث ناموزون شدن تصویر شود. 3 – Mode Zoom: تصویر بدون خراب شدن کیفیت، آنقدر بزرگ می‌شود تا در فرم جا بگیرد.
Width	این خصوصیت را می‌توانیم با کلیک روی گیره‌های موجود روی لبه‌های فرم و سپس کشیدن آن‌ها تا رسیدن به اندازه دلخواه، تنظیم کنیم.



تصویر ۲-۹ جعبه رنگ برای خصوصیت Back Color

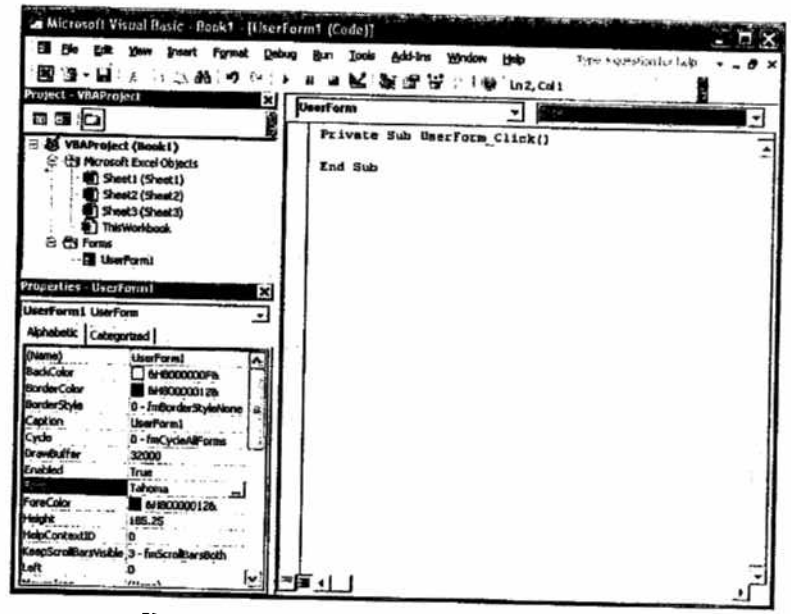


تصویر ۳-۹ خصوصیت Font

مشاهده فرم

در هنگام طراحی فرم ممکن است بخواهیم ببینیم که این فرم در هنگام اجرا به چه صورت است. وقتی در design mode هستیم، می‌توانیم با انتخاب Run | Sub/UserForm یا فشار دادن F5، این کار را انجام دهیم.

همان‌طورکه در رابطه با اشیا ویژوال بیسیک می‌دانید، هر فرم دارای ماژول خاص خود است که با رویدادهای موجود روی فرم سروکار دارد (تصویر ۴-۹). برای دستیابی به ماژول، دوبار روی فرم طراحی کلیک می‌کنیم یا از منو، View | Code را انتخاب می‌کنیم یا دکمه F7 را فشار می‌دهیم.



تصویر ۴-۹ پنجره کد نویسی برای رویدادهای UserForm

در سمت راست ماژول می‌توانیم یک لیست بازشو ببینیم که رویدادهای مرتبط با فرمی را که برای آن کدنویسی می‌کنیم، فهرست می‌کند. برای مثال یکی از این رویدادها، Initialize است که در هنگام

فراخوانی فرم فعال می‌شود. اگر روی Initialize کلیک کنیم بلافاصله سر صفحه و پا صفحه کدنویسی برای آن رویداد ظاهر خواهد شد. یک کادر پیغام به صورت زیر را به آن اضافه کنید:

```
Private Sub UserForm_Initialize

    MsgBox "This is my form"

End Sub
```

حال دکمه F5 را فشار دهید تا فرم اجرا شود. ابتدا کادر پیغام ظاهر شده و بعد از کلیک روی OK در کادر پیغام، فرم ظاهر خواهد شد. با مرور لیست رویدادها خواهید دید که در تعدادی از آن‌ها می‌توانیم کدنویسی کنیم. رویداد MouseMove را امتحان کنید. روی MouseMove کلیک کنید تا سرصفحه و پاصفحه کدنویسی مربوط به‌آن ظاهر شود و سپس کد زیر را در آنجا بنویسید:

```
Private Sub UserForm_MouseMove(ByVal Button AS Integer, ByVal Shift As Integer,
ByVal X As Single, ByVal Y As Single)

If UserForm1.Caption = "UserForm1" Then

    UserForm1.Caption = "You moved the mouse"

Else

    UserForm1.Caption = "UserForm1"

End If
End Sub
```

پس از اجرای فرم متوجه خواهید شد که با حرکت ماوس روی فرم، عنوان (Caption) موجود در نوار ابزار فرم، تغییر می‌کند.

توجه کنید که رویداد MouseMove پارامترهایی برای تعیین موقعیت X و Y ماوس روی فرم و همچنین برای تعیین فشرده شدن دکمه‌های ماوس با کلید SHIFT نیز دارد. رویدادهایی برای KeyDown ، KeyUp، MouseDown و MouseUp را خواهید دید:

- KeyDown: وقتی کلیدی توسط کاربر فشرده شود، راه‌اندازی می‌شود.
- KeyUp: وقتی کلید فشرده شده توسط کاربر رها شود، راه‌اندازی می‌شود.
- MouseDown: وقتی کاربر روی یکی از دکمه‌های ماوس کلیک کند راه‌اندازی می‌شود.
- MouseUp: وقتی کلید فشرده شده ماوس رها شود راه‌اندازی می‌شود.

- KeyPress: وقتی کلیدی روی صفحه کلید فشرده شده و سپس رها شود راه‌اندازی می‌شود. پارامتر موجود در این رویداد، مقدار کلید فشرده شده را مشخص می‌کند. این رویداد، ترکیبی از KeyDown و KeyUp است.
- Click: وقتی ماوس روی یک کنترل مانند دکمه فرمان یا کادر انتخابی کلیک کند فعال می‌شود.

نمایش فرم در کد

می‌توانیم با فشار دادن کلید F5 نتایج فرم را ببینیم اما باید بتوانیم فرم را به کد پیوند بزنیم. با متد Show می‌توان این کار را انجام داد:

```
UserForm1.Show
```

وقتی این دستور اجرا شود، فرمان به پنجره UserForm انتقال می‌یابد. هر کدی که در رویداد Initialize باشد اجرا شده و منتظر عکس العمل کاربر می‌ماند. عکس العمل کاربر ممکن است بستن پنجره یا کلیک روی OK در فرم باشد.

وقتی کارمان با فرم انجام شد می‌توانیم آن‌را پنهان کنیم. برای این‌کار از متد Hide استفاده می‌کنیم:

```
UserForm1.Hide
```

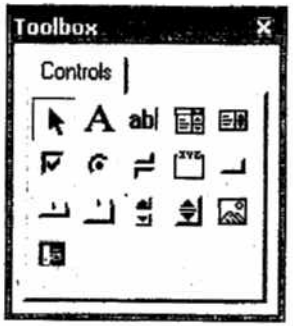
تنها تصمیم دیگری که باید اتخاذ شود، از جایی است که فرم را می‌بندید. لازم است که آن‌را به یک رویداد کاربری مانند وقتی که یک برگه جدید به کارپوشه اضافه می‌شود، متصل کنیم. به خاطر دارید که در فصل۱ چطور یک کادر پیغام با محتوای "Hello Word" را به رویداد NewSheet اضافه کردیم؟ می‌توان همین کار را برای نمایش یک فرم هم انجام داد. روی ThisWorkBook در ساختار درختی Project، دوبار کلیک کنید تا ماژول WorkBook باز شود. از منوی باز شو در سمت چپ پنجره ماژول، WorkBook و از منوی بازشوی سمت راست، NewSheet را انتخاب کنید.سپس کد زیر را وارد نمایید:

```
Private Sub WorkBook_NewSheet(ByVal Sh AS Object)
    UserForm1.Show
End Sub
```

حال به صفحه گسترده رفته و یک برگه جدید درج کنید. فرم شما ظاهر خواهد شد.

پرکردن فرم

اکنون فرم شما خوب کار می‌کند اما کار خاصی انجام نمی‌دهد. لازم است چند کنترل روی آن قرار دهید تا با کاربر ارتباط برقرار کند. این کنترل‌ها می‌توانند کادرهای کمبو، کادرهای متن یا دکمه‌های فرمان باشند. ابتدا لازم است کد رویداد ماوس را که در بخش قبلی اضافه شده حذف کنیم، در غیر این صورت در هنگام اجرای فرم، هر بار که ماوس را حرکت دهید، کادرهای پیغام ارسال خواهند شد. کد موجود در پنجره جعبه ابزار را حذف کنید. این پنجره معمولاً در سمت چپ فرم ظاهر می‌شود و حاوی متداول‌ترین کنترل‌هاست. اگر این پنجره را نمی‌بینید روی UserForm کلیک کنید تا پنجره جعبه ابزار ظاهر شود (تصویر ۵-۹). جعبه ابزار، آیکن مربوط به کنترل‌هایی که می‌توانید استفاده کنید، نمایش می‌دهد.



تصویر ۵-۹ پنجره Toolbox

برای قرار دادن یک کنترل روی فرم کافی است فقط روی آیکن مربوط به آن در جعبه ابزار کلیک کنید و آن‌را به محل دلخواه‌تان روی فرم بکشید. البته بعداً هم می‌توانید آن‌را به محل جدیدی روی فرم انتقال دهید یا با کشیدن گیره‌هایی که روی لبه‌های کنترل وجود دارند آن‌را به اندازه دلخواه در آورید.

کنترل‌های پیش‌فرض جعبه ابزار

بخش‌های صفحه بعد به بررسی کنترل‌هایی می‌پردازد که از ابتدا و به صورت پیش‌فرض در جعبه ابزار وجود داشته‌اند:

Label (برچسب)

کنترل Label، متنی را روی فرم نمایش می‌دهد. می‌توانیم متن را با وارد کردن آن در خصوصیت Caption در پنجره Properties در گوشه پایین- سمت چپ صفحه نمایش، وارد کنیم.
خصوصیاتی مانند BackColor, TextAlgin, ForeColor, Font وWordWrap را هم می‌توانیم برای آن تنظیم کنیم. البته این، لیست کامل خصوصیات نیست و چندین خصوصیت دیگر هم وجود دارند. می‌توانیم در پنجره Properties، دیگر خصوصیات آن کنترل را به‌طور کامل ببینیم.

TextBox (کادر متنی)

کنترل TextBox مشابه با کنترل Label است اما به کاربر اجازه می‌دهد تا در هنگام اجرا، متنی را وارد کند. هم‌چنین می‌توانیم آن‌را به یک منبع دیگر ارجاع کنیم (مثلاً سلولی از یک صفحه گسترده مانند Sheet1!a1). اگر این‌کار را انجام دهید نه تنها کادر متنی، مقدار آن سلول را به خود می‌گیرد بلکه هر چیزی که در کادر متنی تایپ شود نیز در آن سلول نوشته می‌شود و سلول به صورت مؤثر به عنوان متغیری در ذخیره کردن داده‌ها عمل می‌کند.
از دیگر خصوصیاتی که برای این کنترل می‌توانیم تنظیم کنیم می‌توان به Enable, BackColor, ForeColor, Font, Locked, TextAlign, MaxLength, Mulitiline, PasswordCharacter وWordWrap اشاره کرد.

نکته: Enable به معنای آماده بودن کنترل برای استفاده است. اگر مقدار آن False باشد کنترل به رنگ خاکستری درآمده و غیر فعال می‌شود. Locked که مقدار پیش‌فرض آن False می‌باشد به معنای آن است، در صورتی‌که مقدارش True باشد، قابل مشاهده است (خاکستری نیست) اما کاربر نمی‌تواند متنی را در آن وارد کند.

ComboBox

کنترل ComboBox کادر افقی آشنایی است که وقتی روی پیکان رو به پایین آن کلیک می‌کنیم لیستی از مقادیر باز می‌شود. خصوصیات آن، به غیر از کاراکتر Password، مشابه با کادر متن است. این کنترل چند خصوصیت جدید هم دارد:

جدول ۲-۹

خصوصیت	شرح
ControlSource	این خصوصیت به یک سلول (مثل Sheet1!a1) ارجاع کرده و مانند ControlSource برای کادرهای متنی عمل می‌کند.
MatchRequired	این خصوصیت به معنی آن است که مقداری که کاربر تایپ می‌کند باید با مقداری در لیست تطبیق یابد و این روش برای اطمینان از این است که تنها از مقادیر موجود در لیست استفاده کرده‌ایم.
RowSource	این خصوصیت می‌تواند به سلول‌هایی در صفحه گسترده ارجاع کند و به عنوان منبعی برای لیست بازشو عمل نماید. این خصوصیت می‌تواند به چند سلول هم ارجاع کند (مانند Sheet 1!a1.a10).

ListBox (کادر فهرست)

کنترل ListBox، لیستی از مقادیر قابل انتخاب را نمایش می‌دهد و اجزا آن‌را به صورت دایمی می‌توان دید. این کنترل خصوصیات مشابهی با Combo Box دارد اما دارای خصوصیات دیگری هم هست:

جدول ۳-۹

خصوصیت	شرح
BoundColumn	کادر فهرست به شما اجازه می‌دهد که بیش از یک ستون داده را به عنوان منبع داشته باشید. خصوصیت BoundColumn نشان می‌دهد که اگر داده‌ای انتخاب شود، کدام ستون داده‌ها را در منبع کنترل قرار خواهد داد.
ColumnCount	نشان می‌دهد که چه تعداد ستون به کار گرفته می‌شود. این خصوصیت به خصوصیات BoundColumn و RowSource متصل می‌شود. اگر شماره ستون (Column count) بیش از یک باشد آن‌گاه RowSource می‌تواند به سراغ بیش از یک ستون در صفحه گسترده رود.
ColumnHeads	اگر مقدار آن True باشد، سلول‌هایی که دقیقاً بالای RowSource هستند به‌کار گرفته می‌شوند.
ColumnWidths	می‌توانیم پهنای آن‌را با استفاده از نقطه ویرگول (؛) تنظیم کنیم تا اعداد

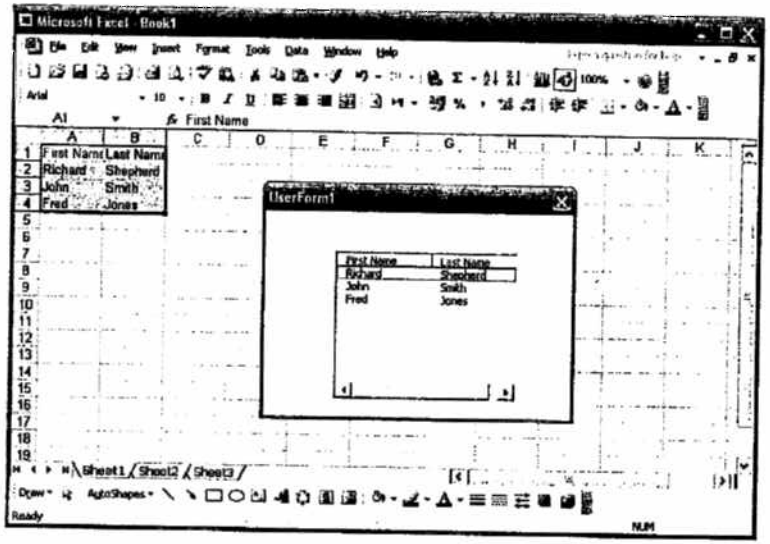
خصوصیت	شرح
	را از هم جدا کند (مثلاً ۲۰،۳۰).
MultiSelect	می‌توانیم مشخص کنیم که آیا کاربر می‌تواند بیش از یک گزینه را از لیست انتخاب کند یا خیر. اگر اجازه چندین انتخاب به کاربر دهیم، نمی‌توانیم با استفاده از خصوصیت ControlSource این را بخوانیم. در عوض باید از کدنویسی استفاده کنیم و آن‌را روی ماژول فرم قرار دهیم. مقادیر این خصوصیت می‌توانند fmMultiSelectSingle, fmMultiSelectMulti و fmMultiSelectExtended باشند.

در مثال تصویر ۶-۹، مقدار خصوصیت RowSource را برابر Sheet1!a2..b4، مقدار خصوصیت ColumnCount را برابر ۲ و مقدار خصوصیت ColumnHeads را True انتخاب کرده‌ایم.

برای خواندن چندین انتخاب، روی کادر فهرست دابل‌کلیک کنید تا ماژول فعال شود و سپس روی رویداد Exit در منوی بازشوی سمت راست کلیک کرده، کد زیر را وارد نمایید:

```
Private Sub ListBox1_Exit (ByVal Cancel AS MSForms.ReturnBoolean)
    For n = 0 To UserForm1.ListBox1.ListCount – 1
      If UserForm1.ListBox1.Selected(n) = True Then
          MsgBox UserForm1.ListBox1.List(n, 0)

  End If
Next n
End Sub
```

وقتی فرم بسته می‌شود رویداد Exit روی کادر فهرست فعال می‌شود که در مرحله بعدی مقادیر موجود در لیست را مرور می‌کند تا ببیند که آیا انتخاب شده‌اند یا خیر. توجه کنید که شاخص آن از صفر آغاز می‌شود. سپس لازم است برای صحت عملیات در حلقه For..Next، یک واحد از خصوصیت ListCount کسر کنیم. پس این کد، تمام ردیف‌های گزینش شده را نمایش خواهد داد. پارامتر دوم (0)، اختیاری است و بر ستون‌ها دلالت دارد. توجه کنید که شاخص آن از صفر (نه یک) آغاز می‌شود. چون مقدار خصوصیت Selected را True انتخاب کرده‌ایم، از آنچه انتخاب شده است، مطلع می‌شویم.



تصویر ۶-۹ یک کادر فهرست روی یک UserForm

CheckBox (کادر انتخاب)

این کنترل به کاربر اجازه می‌دهد تا یک کادر را انتخاب کند یا از حالت انتخاب خارج کند. با دابل کلیک کردن روی کنترل یا با تنظیم خصوصیت Caption می‌توانیم متنی برای آن تنظیم کنیم. با تنظیم خصوصیت ControlSource می‌توانیم آن را به یک سلول وصل کنیم تا به سلولی مانند Sheet1!a1 اشاره کند. اگر سلول مذکور مقداری نداشته باشد، کادر انتخاب به صورت پیش‌فرض حاوی مقدار True خواهد بود. توجه کنید که سلول منبع کنترل، مقدار True یا False را نمایش می‌دهد تا وضعیت کادر انتخابی را مشخص کند.

نکته: خصوصیات Locked و Enable برای کادرهای انتخاب دقیقاً مانند کادرهای متنی عمل می‌کنند. کدی که برای تفسیر کادر انتخاب به‌کار می‌رود دقیقاً مانند کد کنترل ListBox است اما رویداد Change برای کادرهای انتخاب اضافه شده است:

```
Private Sub CheckBox1_Change()
    MsgBoxUserForm1.CheckBox1.Value
End Sub
```

هر بار کادر انتخاب گزینش شده یا از حالت انتخاب خارج شود، یک کادر پیغام به نمایش درمی‌آید که مقدار آن را نشان می‌دهد.

OptionButton (دکمه گزینش)

دکمه‌های گزینش را گاهی دکمه رادیویی (radio button) نیز می‌نامند و این زمانی است که مجموعه‌ای از آن‌ها را در اختیار داریم چون آن‌ها مانند دکمه‌های فشاری روی یک رادیوی قدیمی عمل می‌کنند.

برای آن‌که این دکمه‌ها به درستی عمل کنند لازم است حداقل دوتا از آن‌ها را روی فرم داشته باشیم چون وقتی یکی از آن‌ها فعال می‌شود بقیه به صورت خودکار غیرفعال می‌شوند. به همین دلیل نمی‌توانیم از یکی از آن‌ها به تنهایی استفاده کنیم.

با استفاده از خصوصیت ControlSource می‌توانیم این دکمه‌ها را به یک سلول وصل کنیم (مثل Sheet1!a1). بسته به این‌که آیا دکمه مذکور انتخاب شده است یا خیر، سلول مربوطه حاوی مقدار True یا False خواهد بود.

برای تفسیر مقدار هر دکمه رادیویی می‌توانیم کدنویسی کنیم. کد زیر را روی رویداد Change یک دکمه گزینشی امتحان کنید:

```
Private Sub OptionButton1_Change( )
    MsgBox UserForm1.OptionButton1.Value
End Sub
```

Frame (قاب)

قاب‌ها به شما اجازه می‌دهند تا یک مجموعه کنترل مشابه را برای تشریح عملکرد آن‌ها در یک مجموعه کنار هم قرار دهید. فقط می‌توانیم عنوان قاب را با تنظیم خصوصیت Caption تنظیم کنیم.

یکی از مشکلات در مورد کنترل Frame این است که حتی در زمان اجرا، روی کنترل‌های قبلی موجود روی فرم را می‌پوشاند. وقتی از کنترل Frame استفاده می‌کنیم ابتدا باید کنترل Frame را تعریف کرده و سپس دیگر کنترل‌ها را روی آن قرار دهیم.

CommandButton (دکمه فرمان)

این کنترل، یک کنترل قوی است که زیاد روی فرم‌ها مورد استفاده قرار می‌گیرد. می‌توانیم عنوان دکمه فرمان را باداابل کلیک روی‌آن یا تنظیم خصوصیت Caption تغییر دهیم. همچنین می‌توانیم با تنظیم مقدار خصوصیت Default برابر با True،یک دکمه را به عنوان دکمه پیش‌فرض درآوریم. این به

معنای آن است که در هنگام بارگذاری فرم، آن دکمه به صورت فعال در آمده و تمرکز روی آن است (یعنی اگر کاربر Enter را فشار دهد، کد مربوط به آن دکمه اجرا می‌شود).

برای این‌که دکمه‌ای کاری انجام دهد باید در یک رویداد برای آن، کدنویسی کنیم. برای وارد شدن به ماژول، روی فرم دابل کلیک کرده و از منوی بازشویی که در قسمت بالا و سمت چپ پنجره قرار دارد،کنترل دکمه فرمان را انتخاب کنید. رویداد Click را انتخاب کرده و کد زیر را وارد نمایید:

```
Private Sub CommandButton1_Click ()
MsgBox "You pressed the button"
    End Sub
```

TabStrip (نوار دکمه‌ای)

این کنترل، به شما اجازه می‌دهد تا یک نوار دکمه‌ای را روی فرم قرار دهید. از کنترل TabStrip می‌توانیم برای مرور مجموعه اطلاعات مختلف کنترل‌های مرتبط با هم، استفاده کنیم. منطقه سرویس‌دهی کنترل TabStrip، یک فرم مجزا محسوب نمی‌شود. در عوض، این منطقه بخشی از فرم است که حاوی کنترل TabStrip می‌باشد.

می‌توانیم در زمان طراحی، با کلیک روی دکمه و نگه داشتن کلید SHIFT، چند دکمه را با هم انتخاب کنیم. حتی می‌توان با کلیک راست روی یک دکمه(Tab)، صفحاتی را اضافه کرد،تغییر نام داد و یا حذف نمود. برای تفسیر عملیات‌های کاربری روی نوار دکمه‌ای، لازم است حتماً کدنویسی کنیم. برای دیدن ماژول، روی فرم دابل کلیک کنید و سپس از منوی بازشویی که در قسمت بالا و سمت چپ قرار دارد، TabStrip را انتخاب کنید. از منوی بازشوی بالا- سمت راست رویداد Change را انتخاب کرده و کد زیر را بنویسید:

```
Private Sub TabStrip1_Change()
        MsgBox TabStrip1.SelectedItem.Index

        MsgBox TabStrip1.SelectedItem.Caption

End Sub
```

وقتی فرم را اجرا کنیم و روی دکمه‌ها کلیک کنیم، شاخص دکمه انتخاب شده (که مقدار آن برای نخستین دکمه برابر با 0 است) و عنوان آن (caption) به نمایش در می‌آید.

با قرار دادن کد زیر در قسمت رویداد Change، می‌توانیم بهتر تأثیر کنترل TabStrip را مشاهده کنیم:

```
Private Sub TabStrip1_Change()
    Select Case TabStrip1.SelectedItem.Index
      Case 0
                    TabStrip1.ForeColor = QBColor(12)
      Case 1
```

```
        TabStrip1.ForeColor = QBColor(15)
    End Select
End Sub
```

هر بار روی یک دکمه کلیک می‌کنیم، رنگ متن روی دکمه‌ها تغییر می‌کند. توجه کنید که هر دو دکمه هم رنگ می‌شوند.

MultiPage (چند صفحه‌ای)

کنترل TabStrip روی یک فرم قرار دارد. کنترل MultiPage، فرم‌های مختلف انتخاب شده توسط دکمه‌ها است و به دلایل زیادی سودمند است. در زمان طراحی می‌توانیم با کلیک روی هر دکمه، آن را انتخاب کرده و روی آن راست کلیک کنیم تا صفحاتی را درج نماییم، تغییر نام دهیم یا حذف کنیم. می‌توانیم کنترل‌ها را روی هر صفحه مستقل دکمه بکشیم. سعی کنید روی صفحه نخست، یک کادر متنی و سپس روی صفحه دوم یک دکمه فرمان قرار دهید.

وقتی فرم را اجرا می‌کنیم، هر صفحه مانند یک فرم مجزا عمل خواهد کرد و تنها کنترل‌های مربوط به خود را نمایش می‌دهد. آن‌ها طوری رفتار می‌کنند که انگار روی فرم‌های مجزایی قرار دارند. توجه کنید که در زمانی طراحی، وقتی روی هر صفحه کلیک می‌کنید، پنجره Properties متفاوتی برای هر صفحه باز می‌شود که در مورد کنترل TabStrip چنین نبود.

می‌توانیم برای تفسیر عملیات‌های کاربر، مانند TabStrip، کدنویسی کنیم. روی صفحه دکمه دوبار کلیک کنید تا وارد ماژول کنترل MultiPage شوید در حالی‌که رویداد Change فعال است، کد زیر را وارد نمایید:

```
Private Sub MultiPage1_Change()
    MsgBox MultiPage1.SelectedItem.Caption
End Sub
```

ScrollBar (نوار پیمایش)

این کنترل، یک نوار پیمایش عمودی روی فرم قرار می‌دهد که مشابه با نمونه‌هایی است که در بسیاری از برنامه‌های کاربردی مایکروسافت دیده‌ایم. خصوصیاتی برای مقادیر حداکثر و حداقل و برای تغییرات کوچک و بزرگ در این نوارها وجود دارد که SmallChange و BigChange نامیده می‌شوند. تغییر کوچک، زمانی رخ می‌دهد که روی یکی از پیکان‌های نوار پیمایش کلیک کنیم و تغییر رنگ زمانی رخ می‌دهد که در منطقه بین پیکان و مکان‌نما روی نوار پیمایش کلیک نماییم.

مقدار SmallChange و BigChange به صورت پیش‌فرض 1، مقدار حداکثر برابر 32767 و مقدار حداقل برابر صفر تنظیم شده است. این به معنای آن است که کلیک روی پیکان‌ها یا فضای خالی بین پیکان و مکان‌نما، چیزی را زیاد جابه‌جا نمی‌کند: اگر مقدار حداکثر برابر 32767 تنظیم شده باشد، لازم است BigChange حدود 1000 و SmallChange حدود 100 تنظیم شده باشند.

می‌توانیم با استفاده از خصوصیت ControlSource، این کنترل را به یک سلول صفحه گسترده (مانند Sheet1!a.a10) متصل کنیم. مقدار نوار پیمایش در سلول ظاهر می‌شود اما تنها زمانی به روز درآورده می‌شود که کنترل دیگری غیر از نوار پیمایش فعال شده باشد.

هم‌چنین می‌توانیم برای خواندن مقدار نوار پیمایش، کدنویسی کنیم. کد زیر را وارد کنید:

```
Private Sub ScrollBar1_Change()
    MsgBox ScrollBar1.Value
End Sub
```

هر بار که نوار پیمایش جابجا می‌گردد، یک پیغام جدید ظاهر شده و مقدار جدیدی را نمایش می‌دهد. می‌توانیم ببینیم که بر طبق مقداری که برای SmallChange وBigChange تنظیم کرده‌ایم، چه تغییراتی رخ می‌دهد.

SpinButton

این کنترل، معمولاً به کنترل دیگری (مانند کادر متنی) ارتباط پیدا می‌کند تا مقداری را نمایش دهد.

می‌توانیم این کنترل را با استفاده از خصوصیت ControlSource به یک سلول صفحه گسترده مرتبط کنیم (مانند Sheet1!a.a10). مقدار این کنترل در سلول ظاهر می‌شود اما تنها وقتی به روز در آورده می‌شود (update) که کنترل دیگری را انتخاب کنیم.

خصوصیت SmallChange، مقدار تغییر را تنظیم می‌کند. خصوصیت Orientation مشخص می‌کند که کنترل به طور عمودی تقسیم شده است یا افقی.

هم‌چنین می‌توانیم کدی بنویسیم تا مقدار این کنترل را بخواند. اگر روی آن دابل کلیک کنیم به رویداد Change این کنترل وارد می‌شود. کد زیر را وارد کنید:

```
Private Sub SpinButton1_Change( )
MsgBox SpinButton1.Value
End Sub
```

Image (تصویر)

این کنترل، کنترلی است که یک تصویر یا عکس را در خود نگه می‌دارد. برای درج یک عکس، روی خصوصیت Picture کلیک کرده و سپس روی کادر (...) کلیک کنید تا وارد پنجره‌ای شوید که می‌توانید نام و مسیر فایل تصویری دلخواه‌تان را مشخص کنید. وقتی روی OK کلیک کنیم، تصویر مربوطه در کنترل درج خواهد شد.

برای حذف تصویر کافی است مقداری را که در کادر خصوصیت Picture قرار دارد پاک کنید تا مقدار آن مجدداً (None) شود.

فصل دهم

کنترل Common Dialog

در زبان برنامه‌نویسی ویژوال بیسیک کنترلی با نام Common Dialog وجود دارد که به شما اجازه می‌دهد برای باز کردن فایل‌ها، ذخیره کردن فایل‌ها، چاپ، انتخاب رنگ و انتخاب قلم دلخواه به کادرهای محاوره استاندارد مایکروسافت دسترسی پیدا کنید.

در این کنترل، یک رابط کامل وجود دارد که به شما اجازه می‌دهد کادر محاوره دلخواه‌تان را انتخاب کرده و بتوانید برای تغییر ظاهر کادر محاوره‌ای به شکلی که منطبق با هدف برنامه شما باشد، پارامترهای گوناگونی تنظیم کنید. از این بهتر آن است که می‌توانید پارامترهای انتخاب شده کاربر را بازیابی کرده و بر طبق آن‌ها کدنویسی نمایید. برای مثال اگر کاربر فایلی را برای باز شدن انتخاب کرده باشد می‌توانیم نام فایل و مسیر آن‌را به سادگی مشخص کرده و بر طبق این اطلاعات کدنویسی کنیم.

یک نکته مهم این است که این کادرها تنها کادرهای محاوره‌ای هستند که یک رابط مشترک با کاربر ارایه می‌کنند. البته برای اجرای عملیات با توجه به گزینش‌ها یا رویدادهایی که کاربر انتخاب می‌کند نیز باید کد نویسی کنیم وگرنه هیچ اتفاقی به صورت خودکار نمی‌افتد. برای مثال اگر کادر محاوره‌ای Save را نمایش دهیم و کاربر نام فایل را تایپ کند و روی OK کلیک کند هیچ اتفاقی نمی‌افتد مگر آن‌که کدی نوشته باشیم که نام فایل انتخاب شده را گرفته و عملیات‌هایی بر طبق آن انجام دهد.

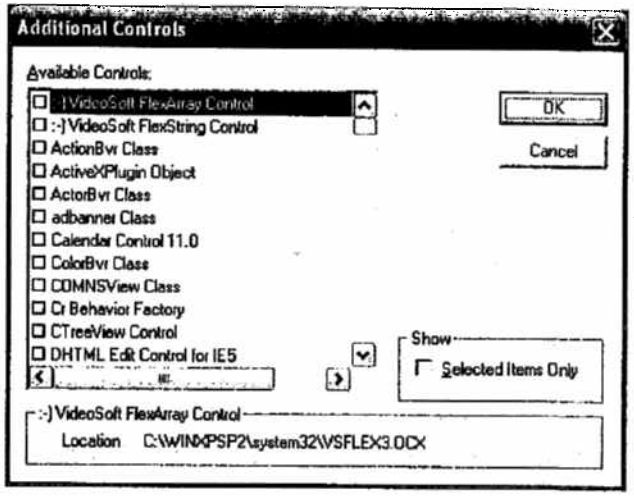
کادر محاوره‌ای Common User Interface دقیقاً مثل هر برنامه دیگر مایکروسافت عمل می‌کند بجز این‌که خصوصیات را تغییر داده و از مقادیر ایجاد شده توسط آن می‌توانیم در کد خود استفاده کنیم.

استفاده از کنترل Common Dialog

این کنترل به صورت پیش‌فرض در جعبه ابزار Forms وجود ندارد پس لازم است که آن‌را اضافه کنیم. برای انجام این کار، لازم است که در یک پنجره فرم باشیم و پنجره جعبه ابزار هم قابل مشاهده باشد. اگر هنوز فرم کاربری اضافه نشده است می‌توانیم از منوی کد، User Form | Insert را انتخاب کنیم. سپس باید فرم و پنجره جعبه ابزار را ببینیم.

از منوی VBE باید Tools | Additional Controls را انتخاب کنیم مدت زمانی طول می‌کشد تا VBA در کامپیوتر شما جستجو کند و کنترل‌های موجود را پیدا کند. حال صفحه نمایش شما باید مثل

تصویر ۱-۱۰ باشد. این صفحه،لیستی از کنترل‌های اضافی موجود را نشان می‌دهد و لیست جامعی است. محتویات این لیست وابسته به این است که چه برنامه‌های کاربردی در کامپیوتر شما بارگذاری شده‌اند.



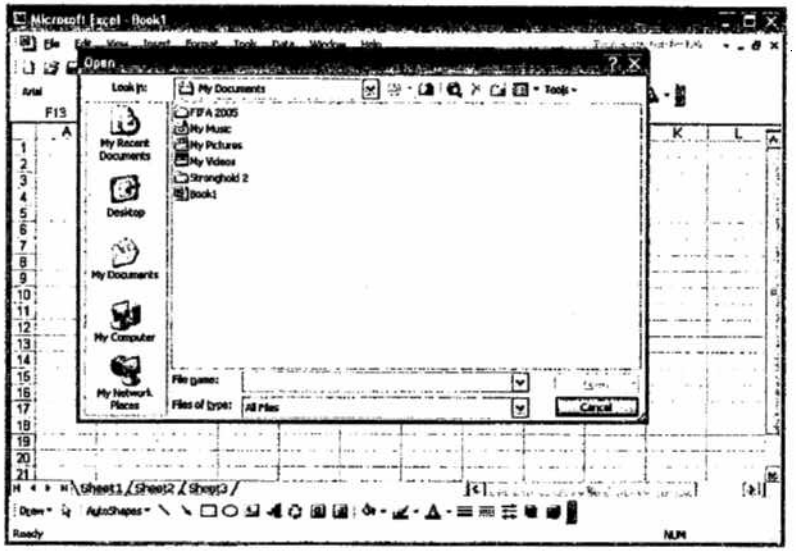
تصویر ۱ - ۱۰ اضافه کردن کنترل Common Dialog

در این لیست جستجو کنید تا به کنترل Microsoft Common Dialog برسید. در کادر کوچک کنار آن تیک زده و OK را انتخاب کنید. حال در جعبه ابزار باید شاهد مریی شدن کنترل Common Dialog باشیم. اگر آیکن آن را نمی‌شناسید، ماوس را یک به یک روی آیکن‌های موجود برده و نوشته‌ای را که در زیر آن‌ها ظاهر می‌شود بخوانید تا به آیکن مذکور برسید.

روی کنترل کلیک کرده و آن را روی یک فرم کاربری بکشید. مهم نیست کنترل را روی کدام فرم و در کدام قسمت قرار می‌دهید چون در زمان اجرا، قابل مشاهده نیست. کنترل مورد استفاده کد VBA قرار می‌گیرد و فرم تنها به عنوان نگه‌دارنده کنترل عمل می‌کند.

کادر محاوره‌ای Open

این کادر به کاربر اجازه می‌دهد تا یک درایو، یک فهرست، پسوند نام فایل و نام یک فایل را انتخاب کند (تصویر۲-۱۰).



تصویر ۲-۱۰ کادر محاوره‌ای Open

برای نمایش دادن یک کادر محاوره‌ای، خصوصیات فیلترها و سپس خصوصیت Action (عملیات) را تنظیم کنید. کد زیر را در یک ماژول تایپ کنید و سپس آن‌را اجرا نمایید:

```
Sub Show_Open ( )
        UserForm1.CommonDialog1.CancelError = True
On Error GoTo errhandler
UserForm1.CommonDialog1.Filter = _
"All Files (*.*) |*.*| Text files (*.txt)"
UserForm1.CommonDialog1.Action = 1
MsgBox UserForm1.CommonDialog1.FileName
errhandler:
Exit Sub
End sub
```

خصوصیت CancelError ابتدا True تنظیم می‌شود. روی یک کادر محاوره‌ای Open File، یک دکمه OK و یک دکمه Cancel وجود دارد. اگر خصوصیت CancelEroor را True تنظیم کنید پس از فشردن دکمهCancel یک پیغام خطا ارسال می‌شود. بد متأسفانه این تنها راهی است که می‌توانیم متوجه شویم که کاربر دکمه OK یا دکمه Cancel را فشرده است.

این کار باعث می‌شود به خط بعدی کد بروید که عملیاتی را راه‌اندازی می‌کند که در صورت بروز خطا اجرا می‌شود. سپس اجرای کد به یک برچسب کد با نام errhandler که در ادامه کد معرفی می‌شود می‌پرد. یک برچسب کد با استفاده از یک نام منحصر به فرد معرفی شده است و این مرحله باعث ایجاد یک نقطه ارجاع می‌شود که کار آن شبیه دستور GoTo است.

خط بعدی کد، فیلتری برای نوع فایل تنظیم می‌کند. در این مثال، فیلتر برای تمام فایل‌ها و تنها برای فایل‌های متنی است. این فیلتر نشان دهنده آن است که در File Type ComboBox چه چیزی ظاهر می‌شود. با استفاده از کلید SHIFT و ۱، نوار عمودی که برای جدا سازی به کار گرفته شده است حاصل می‌شود. این مثال، دو فیلتر را به کار می‌گیرد :

یکی برای تمام فایل‌ها (*.*) و دیگری برای فایل‌های متنی (*.txt)

اگر تنها فایل‌های Word یا Excel را بخواهید می‌توانید به ترتیب از فیلترهای *.doc و *.xls استفاده کنید.

نهایتاً خصوصیت Action را برابر 1 تنظیم می‌کنید که به معنای نمایش کادر محاوره‌ای Open File است. کاربر یک انتخاب انجام می‌دهد و روی OK کلیک می‌کند. کادر پیغام، نام فایل انتخاب شده را نمایش می‌دهد. اگر کاربر روی Cancel کلیک کند یک پیغام خطا صادر شده و اجرای کد به errhandler پریده و خارج می‌شود.

می‌توانید با تنظیم خصوصیت DialogTitle، عنوان موجود در نوار عنوان را تغییر دهید:

```
UserForm1.CommonDialog1.DialogTitle = "Open"
```

اگر کادر انتخابی Read-Only را نخواهید می‌توانید خصوصیت Flags را تنظیم کنید:

```
UserForm1.CommonDialog1.Flags = 4
```

با استفاده از دستور Or، می‌توانید بیش از دو پارامتر را در خصوصیت Flags تنظیم کنیم (به فصل ۶ رجوع شود). تمام خصوصیاتی که می‌خواهیم برای کادر محاوره‌ای به‌کارگیریم باید پیش از تنظیم خصوصیت Action تنظیم شده باشند در غیر این صورت، تأثیری نخواهند داشت.

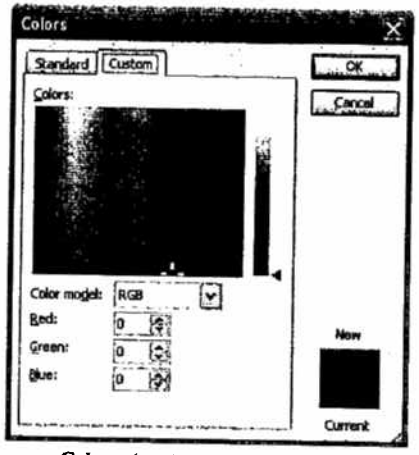
کادر محاوره‌ای Save As

کادر محاوره‌ای Save As دقیقاً مثل کادر محاوره‌ای Open عمل می‌کند با این تفاوت که خصوصیت Action در آن برابر با 2 تنظیم شده است:

```
Sub Show_Save ( )
        UserForm1.CommonDialog1.CancelError = True
        On Error GoTo errhandler
        UserForm1.CommonDialog1.Filter = _
        "All Files (*.*) |*.*| Text files (*.txt)"
        UserForm1.CommonDialog1.Action = 2
        MsgBox UserForm1.CommonDialog1.FileName
        errhandler:
        Exit Sub
End sub
```

کادر محاوره‌ای Colors

کادر محاوره‌ای Colors به کاربر اجازه می‌دهد تا از یک مجموعه رنگ یکی را انتخاب کند یا این‌که رنگ دلخواه خود را ایجاد نماید (تصویر ۳-۱۰).



تصویر ۳-۱۰ کادر محاوره‌ای Color

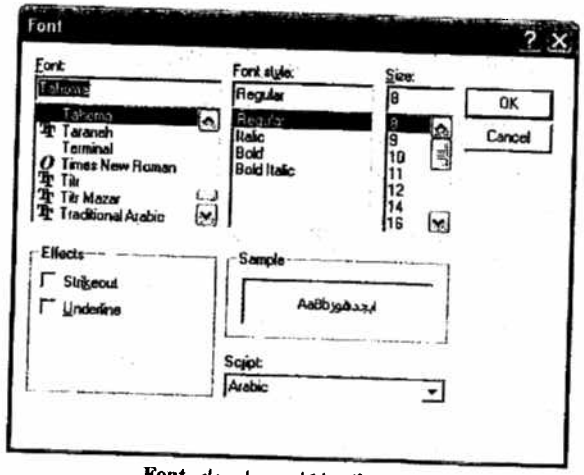
کدنویسی برای آن مثل کادر محاوره Open عمل می‌کند به استثنای این‌که مقدار خصوصیت Action برابر با 3 تنظیم شده و مقدار برگشتی آن، مقدار معادل با یک رنگ است:

```
Sub Show_Color ( )

        UserForm1.CommonDialog1.CancelError = True

        On Error GoTo errhandler

        UserForm1.CommonDialog1.Action = 3
        MsgBox UserForm1.CommonDialog1.Color
        errhandler:
        Exit Sub
End sub
```

کادر پیغام،تعدادی از رنگ های انتخاب شده توسط کاربر را نمایش می‌دهد.

کادر محاوره‌ای Font

این کادر به کاربر اجازه می‌دهد تا با مشخص کردن نوع قلم، اندازه آن، رنگ و سبک نگارش، قلمی را انتخاب کند (تصویر ۴-۱۰).



تصویر ۴-۱۰ کادر محاوره‌ای Font

وقتی کاربر در این کادر محاوره‌ای،گزینشی انجام می‌دهد خصوصیات زیر حاوی اطلاعاتی درباره انتخاب کاربر خواهند بود:

جدول ۱-۱۰

خصوصیت	عملکرد
color	رنگ انتخاب شده
FontBold	آیا حالت ضخیم انتخاب شده است.
FontItalic	آیا حالت مورب انتخاب شده است.
FontStrikeThru	آیا حالت وسط خطدار انتخاب شده است.
FontUnderline	آیا حالت زیر خطدار انتخاب شده است.
FontName	نام قلم انتخاب شده
FontSize	اندازه قلم انتخاب شده

برای انتخاب Color، Strikethru و Underline باید خصوصیت Flags را برابر با &H100& قرار دهیم. با استفاده از عملگر Or می‌توانیم این مقدار را با مقادیر دیگر Flag ترکیب کنیم.

کدی که برای نمایش دادن این کادر به کار می‌رود مثل کادر محاوره‌ای Open عمل می‌کند اما خصوصیت Action در این‌جا برابر 4 بوده و لازم است خصوصیت Flags را به صورت زیر تنظیم کنیم:

جدول ۲-۱۰

نمایش	مقدار خصوصیت
Screen Font	۱
Printer Font	۲
Both	۳

توجه کنید که باید پیش از نمایش کادر محاوره‌ای Font، خصوصیت Flags را تنظیم کنید در غیر این صورت، پیغام خطای "No Fonts Exist" به نمایش در می‌آید:

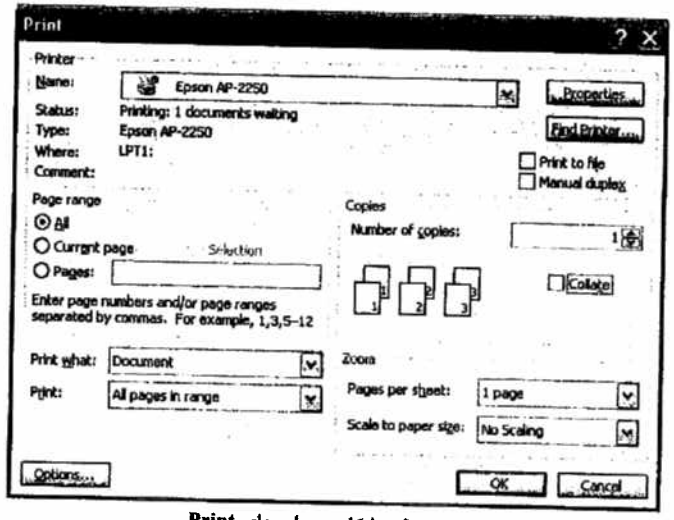
```
Sub Show_Fonts ( )
        UserForm1.CommonDialog1.CancelError = True
        On Error GoTo errhandler
        UserForm1.CommonDialog1.Flags = 3 or &H100&
        UserForm1.CommonDialog1.Action = 4
        MsgBox "FontBold is " & UserForm1.CommonDialog1.FontBold
        MsgBox "FontItalic is " & UserForm1.CommonDialog1.FontItalic
```

```
        MsgBox "FontName is " & UserForm1.CommonDialog1.FontName
        MsgBox "Font Size is " & UserForm1.CommonDialog1.FontSize
        MsgBox "Font Color is " & UserForm1.CommonDialog1.FontColor
        MsgBox "Font StrikeThru is " & UserForm1.CommonDialog1.FontStrikeThru
        MsgBox "Font Underline is " & UserForm1.CommonDialog1.FontUnderline
        errhandler:
        Exit Sub
End sub
```

توجه کنید که خصوصیت Color، شماره رنگ را برمی‌گرداند (مثل کادر محاوره‌ای Color)؛ اما خصوصیات FontName و FontSize، مقادیر واقعی را برمی‌گردانند. بقیه کد هم بر مبنای انتخابی که در کادر محاوره‌ای انجام شده است مقادیر True و False را برمی‌گردانند.

کادر محاوره‌ای Print

این کادر محاوره‌ای به کاربر اجازه می‌دهد تا مشخص کند خروجی چگونه چاپ شود. کاربر می‌تواند محدوده چاپ و تعداد صفحاتی را که باید چاپ شوند، مشخص کند (تصویر ۵-۱۰). این کادر هم‌چنین به کاربر اجازه می‌دهد تا چاپگر را نیز انتخاب کرده و پیکربندی کند. به یاد داشته باشید هر تغییری در پیکر بندی چاپگر دهید در دیگر برنامه‌ها نیز اعمال می‌شود.



تصویر ۵-۱۰ کادر محاوره‌ای Print

این کادر محاوره‌ای، داده‌ای برای چاپگر ارسال نمی‌کند و برای انجام این کار باید کدنویسی کنیم.
وقتی کاربر در زمان اجرا، گزینشی انجام می‌دهد، خصوصیات زیر اطلاعات آن گزینش را نگه می‌دارد:

جدول ۳-۱۰

خصوصیت	عملکرد
Copies	تعداد کپی‌هایی که از یک برگه باید چاپ شود
Orientation	جهت‌گیری چاپگر را مشخص می‌کند: ۱ = عمودی و ۲ = افقی

در Excel، گزینه انتخابی Pages در دسترس نیست.
کدی که برای نمایش دادن این کادر به‌کار می‌رود دقیقاً مثل کادر Open عمل می‌کند به استثنای این‌که مقدار خصوصیت Action برابر با 5 تنظیم شده است:

```
Sub Show_PrintDialog ( )
        UserForm1.CommonDialog1.CancelError = True
        On Error GoTo errhandler
        UserForm1.CommonDialog1.Action = 5
        MsgBox "Copies = " & UserForm1.CommonDialog1.Copies
        MsgBox "Orientation = " & UserForm1.CommonDialog1.Orientation
        errhandler:
        Exit Sub
End sub
```

کادرهای محاوره‌ای پیش‌فرض

یک ابزار کمکی دیگر که مایکروسافت در Excel تعبیه کرده است، کادرهای محاوره‌ای پیش‌فرض هستند. هر بار از یک صفحه گسترده Excel استفاده می‌کنیم، کادرهای محاوره‌ای را برای وارد کردن پارامترها (مثلاً برای قالب‌بندی یک سلول یا به‌کارگیری یک فیلتر) فرا می‌خوانیم. وقتی یکی از این عملیات‌ها را اجرا می‌کنیم یک کادر محاوره‌ای ظاهر می‌شود تا به کاربر اجازه دهد انتخاب‌های دلخواه خود را انجام دهد. اگر عملیاتی را از طریق منوی Format | Cells در Excel اجرا کنیم، یک کادر محاوره‌ای ظاهر می‌شود. این یکی از چندین کادر محاوره‌ای تعبیه شده در Excel است که کد شما می‌تواند به آن دسترسی داشته باشد.
در VBA، یک مجموعه Dialogs وجود دارد که می‌توانید برای نمایش هر کدام از کادرهای محاوره‌ای Excel در کد خود از آن‌ها استفاده کنید. نقطه ضعف این قضیه آن است که نمی‌توانید ظاهر کادرهای

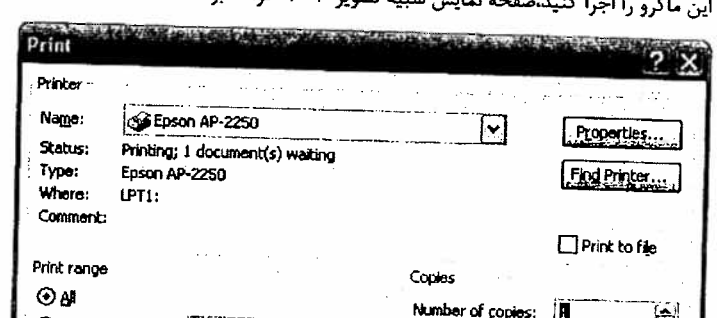
محاوره‌ای را مثل کنترل Common Dialog تغییر دهید و نمی‌توانید مستقیماً به پارامترهای برگشتی دسترسی داشته باشید.

البته این کادرهای محاوره‌ای هنوز هم نقش مهمی در کد ما ایفا می‌کنند مخصوصاً زمانی که نیازی به تفسیر عملیات کاربر نیست. تمام کدهای موجود در این کادرهای محاوره‌ای پیش‌فرض از قبل نوشته شده‌اند تا عملیات کاربر را مورد توجه قرار دهند و همه کاری که ما باید انجام دهیم این است که کادر محاوره‌ای را نمایش دهیم. سپس کد مربوطه به طور خودکار اجرا می‌شود به طوری‌که به نظر می‌رسد کاربر آن‌را از خود منو انتخاب کرده است.

در بخش قبلی، به کارگیری کنترل Common Dialog برای نمایش کادر محاوره‌ای چاپگر را بررسی کردیم اما هنوز هم باید برای انجام عملیات چاپ، کدنویسی کنیم. یک راه میانبر استفاده از کادر محاوره‌ای Excel برای انجام عملیات چاپ با استفاده از شئ Application و متدهای Show است:

```
Sub Printer_Dialog ( )
        Application.Dialogs(xlDialogPrinter).Show
End sub
```

وقتی این ماکرو را اجرا کنید،صفحه نمایش شبیه تصویر ۶-۱۰ خواهد بود.



تصویر ۶-۱۰ استفاده از کادر محاوره‌ای پیش فرض Excel برای Print

این برای Excel بسیار مناسب‌تر است چون تمام عملیات را بر طبق انتخاب‌های کاربر انجام خواهد داد.

برای به‌دست آوردن لیست کاملی از کادرهای محاوره‌ای موجود می‌توانید Application.dialogs را در پنجره‌های کدنویسی تایپ کنید. در این روش تقریباً هر کادر محاوره‌ای که می‌توانید از منوی معمولی Excel می‌توانید به‌دست آورید لیست می‌شود.

فصل یازدهم

Command Bars و Command Buttons

در فصل‌های قبلی دیدید که چگونه کدی را در VBA بنویسید و چگونه آن‌را با فشار دادن کلید F5، یا کلیک روی آیکن Run روی نوار ابزار اجرا کنید. البته در بعضی موارد ممکن است بخواهید خود کاربر، کد را اجرا کند و امکان این هم وجود نداشته باشد که آن‌ها به پنجره کدنویسی رفته و کلید F5 را فشار دهند. البته اگر این امکان هم وجود داشته باشد باز هم بسیاری از کاربران نمی‌دانند چطور این کار را انجام دهند.

در هنگام نوشتن یک برنامه حرفه‌ای حتماً باید به یاد داشته باشید که هر چه کاربر کمتر با برنامه سرو کله بزند بهتر است. کاربران تمایل دارند همه چیز سهل و ساده در اختیارشان قرار گیرد و حوصله یادگیری دستورالعمل‌های طولانی درباره عملکرد ماکرو شما را ندارند. البته ممکن است خودتان نخواهید آن‌ها به منبع کد دسترسی داشته باشند.

برای تحقق این موارد می‌توانید کد را از رویدادی مثل Workbook_Open فرا خوانید و ماکروی شما تنها زمانی که کارپوشه باز می‌شود اجرا شده و به کاربر اجازه نمی‌دهد از تمام رویه‌های جالبی که نوشته‌اید استفاده کند. در عوض می‌توانید از شئ Command Bars برای ایجاد یک منو استفاده کنید یا این‌که یک دکمه فرمان (Command Button) را روی خود صفحه گسترده قرار دهید. با استفاده از شئ Command Bars، می‌توانید منوهای جدید را به عنوان بخشی از ساختار منوهای خود Excel ایجاد کنید. برای مثال وقتی کاربر گزینه Tools را از منو انتخاب کند، می‌توانید یک گزینه منوی اضافی نیز در اختیار او بگذارید تا او را به بخش کدنویسی برنامه هدایت کند. این کار باعث می‌شود برنامه شما یک نمای حرفه‌ای داشته باشد و به نظر برسد بخشی از خود Excel است.

Command Bars

شئ Command Bars نشان دهنده منوهای صفحه گسترده Excel است و به شما اجازه می‌دهد دستورات منوی خود را به دستورات استاندارد Excel اضافه کنید. برای مثال می‌توانید طوری آن‌را تنظیم کنید که وقتی کاربری از منوی صفحه گسترده، Tools را انتخاب کرد، یک گزینه در منو با نام MyCode موجود باشد که همراه با دیگر گزینه‌های انتخابی به نمایش در آید. این کار باعث می‌شود برنامه شما یک نمای حرفه‌ای داشته باشد و بتوانید ابزارهای جدیدی برای راحتی کاربر اضافه کنید.

در این‌جا مثال ساده‌ای می‌آوریم که در آن، یک گزینه منوی جدید به بخش Tools اضافه شده و برای آن هم کدنویسی می‌کنیم. وارد یک ماژول شده و زیرروال زیر را به آن اضافه نمایید:

```
Sub Test_Menu ( )
    MsgBox "You pressed my menu item"
End Sub
```

وقتی این کد اجرا شود، یک کادر پیغام به مضمون "You pressed my menu item." به نمایش در می‌آید.

حال کد زیر را اضافه کنید. توجه کنید که در دوخط از کد، از کاراکتر زیر متن (Underscore) استفاده کرده‌ایم. این کاراکتر اجازه می‌دهد ادامه خطوط طولانی کد را در خط بعدی نوشته و در هنگام اجرا، هر دوی آن‌ها با هم اجرا شوند:

```
Sub MenuCommand ( )
    CommandBars ("Worksheet Menu Bar").Controls _
      ("Tools").Controls.Add _
    (Type:=msoControlButton).Caption = "MyMenu"
    CommandBars("Worksheet Menu Bar").Controls _
    ("Tools").Controls("MyMenu").OnAction = "Test_Menu"
End Sub
```

کد را تنها یک بار اجرا کنید و سپس به صفحه گسترده بروید. اگر منوی Tools را انتخاب کنید. گزینه‌ای اضافی در انتهای آن خواهید دید که MyMenu نامیده می‌شود. در صورتی که آن‌را انتخاب کنیم، کادر پیغام فوق به نمایش در می‌آید.

خط نخست کد فوق، نوار منوی MyMenu را به منوی Tools اضافه می‌کند. خط دوم کد، مشخص می‌کند که وقتی کاربر این کار را انجام داد، چه عملیاتی اجرا شود. خصوصیت OnAction طوری تنظیم شده است که به زیرروال Test_Menu اشاره داشته باشد.

اگر از Excel خارج شده و دوباره برنامه را اجرا کنیم حتی بدون باز کردن فایل نیز هنوز گزینه انتخابی ما در منو وجود دارد. به نظر می‌رسد که عملیات الحاق تأثیر دایمی دارد. پس برای حذف یک گزینه از منو چه کاری باید انجام داد؟ با استفاده از متد Delete می‌توانیم این کار را انجام دهیم:

```
Sub MenuCommand_Remove ( )
    CommandBars ("Worksheet Menu Bar").Controls _
    ("Tools").Controls.("MyMenu").Delete
End Sub
```

اگر این کد را اجرا کنیم گزینه MyMenu از منوی Toolsحذف می‌شود. توجه کنید اگر این کد را

دوباره اجرا کنیم یک پیغام خطا صادر می‌شود چون چیزی برای حذف کردن وجود ندارد. این یکی از مواردی است که به‌کارگیری دستور On Error Resume Next در کد برای مواردی که دوبار فراخوانی انجام می‌شود، مفید است.

نکته: منوهای موجود Excel را نمی‌توانیم حذف کنیم.

ممکن است متوجه شده باشید که با استفاده از نوار افقی، منوها به چندین بخش تقسیم شده‌اند. برای مثال اگر منوی File را باز کنید می‌بینید که بین Close و Save یک نوار افقی وجود دارد که آن‌ها را به گروههای جداگانه تقسیم می‌کند. با استفاده از متد BeginGroup می‌توانیم چنین نواری ایجاد کنیم:

```
Sub MenuCommand ( )
    CommandBars ("Worksheet Menu Bar").Controls _
     ("Tools").Controls.Add _(Type:=msoControlButton).Caption_
     = "MyMenu"
    CommandBars("Worksheet Menu Bar").Controls _
    ("Tools").Controls("MyMenu").OnAction = "Test_Menu"
    CommandBars("Worksheet Menu Bar").Controls _
    ("Tools").Controls("MyMenu").BeginGroup = True
End Sub
```

با اجرای این کد می‌بینید که گزینه MyMenu در یک گروه مستقل قرار می‌گیرد. باید مقدار خصوصیت BeginGroup را True تنظیم کنید تا خطی بین گزینه‌های موجود کشیده شود.

هم‌چنین می‌توانیم مشخص کنیم که گزینه‌های جدید در کدام قسمت لیست منو ظاهر شوند. مقدار پیش‌فرض این است که آن‌ها همیشه در انتهای لیست ظاهر شوند اما پارامتر before به ما اجازه می‌دهد مشخص کنیم، گزینه‌های جدید در کدام بخش منو قرار می‌گیرند:

```
Sub MenuCommand ( )
    CommandBars ("Worksheet Menu Bar").Controls _
     ("Tools").Controls.Add (Type:=msoControlButton, before:=7) _
     .Caption = "MyMenu"
    CommandBars("Worksheet Menu Bar").Controls _
    ("Tools").Controls("MyMenu").OnAction = "Test_Menu"
End Sub
```

در کد فوق، مقدار پارامتر before برابر با 7 است که باعث می‌شود گزینه‌های جدید به عنوان هفتمین گزینه قرار گیرند. گزینه‌های بعدی در نوار منو به پله‌های بعدی انتقال می‌یابند تا گزینه‌ای که در محل هفتم قرار داشت، حذف نشود. از این پس گزینه هفتم، گزینه هشتم منو خواهد بود.

با تنظیم خصوصیت Enabled می‌توانیم گزینه‌های موجود در منو را فعال یا غیر فعال کنیم:

```
CommandBars ("Worksheet Menu Bar"). _
Controls("Tools").Controls("MyMenu").Enabled = False
```

این کد باعث می‌شود گزینه منوی شما به رنگ خاکستری (غیر فعال) درآید. چنان‌چه مقدار آن‌را True تنظیم کنیم، دوباره فعال می‌شود.

با تنظیم خصوصیت Visible می‌توانیم یکی از گزینه‌های موجود در لیست را مرئی یا غیر مرئی کنیم:

```
CommandBars ("Worksheet Menu Bar"). _
Controls("Tools").Controls("MyMenu").Visible = False
```

خصوصیات Enabled و Visible را می‌توانیم برای تمام گزینه‌های منوی موجود در Excel تنظیم نماییم. هر چند نمی‌توان این گزینه‌ها را حذف کرد اما کنیم آن‌ها را با ساختار منوی دلخواه‌مان تعویض کنیم. البته در هنگام انجام چنین کاری باید دقت زیادی کنیم تا کارآیی Excel صدمه نبیند. متد Reset می‌تواند برای تنظیم مجدد یک منو به ساختار منوی پیش‌فرض به کار رود:

```
CommandBars ("Worksheet Menu Bar").Controls("Tools").Reset
```

هم‌چنین ممکن است بخواهید یک زیر منو به گزینه جدید منویی که خود ساخته‌اید اضافه کنید تا وقتی کاربر آن گزینه را در منو انتخاب می‌کند، یک منوی دیگر هم ظاهر شود (مانند زمانی که Tools | Macro را انتخاب می‌کنیم). می‌توانید با اضافه کردن گزینه منوی غیر استاندارد (Pop-up)، یک زیر منو ایجاد کنید:

```
Sub Test_Popup ( )
Dim newsubitem As Object
CommandBars ("Worksheet menu bar").Controls("Tools"). _
Controls.Add.(Type:=msoControlPopup).Caption = "MyPopup"
Set newsubitem = CommandBars("Worksheet Menu Bar"). _
 Controls.("Tools").Controls("MyPopup")
With newsubitem
        .Controls.Add(Type:=msoControlButton).Caption = "Option1"
        .Controls("option1").OnAction = "Test_Menu"

        .Controls.Add(Type:=msoControlButton).Caption = "Option2"
        .Controls("Option2").OnAction = "Test_Menu"
         End With
End Sub
```

در خط نخست کد فوق، شیئی با نام newsubitem ایجاد شده است تا شئ منوی Pop-up را نگه دارد. این کار ضروری نیست اما باعث می‌شود خواندن و دنبال کردن کد ساده‌تر باشد. سپس مثل یک گزینه

منوی استاندارد، گزینه منوی MyPopup ایجاد شده است اما این‌بار نوع آن‌را msoControlPopup تعیین کرده‌ایم.

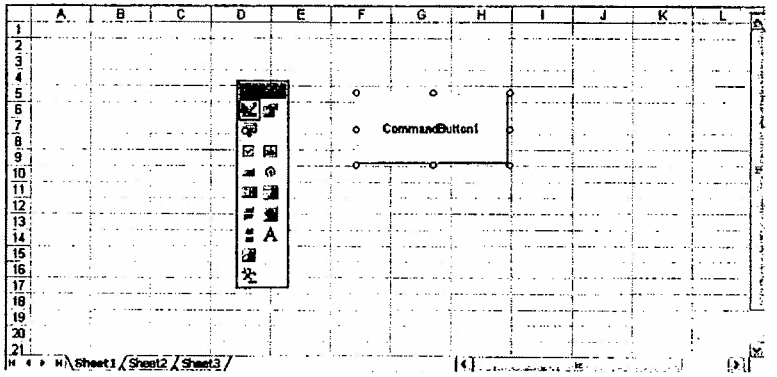
شئ newsubitem برای گزینه منوی غیراستانداردی که قبلاً ایجاد کرده‌ایم، تنظیم شده است. گزینه‌های منو را به همان روش قبل به زیر منوی جدید اضافه کنید.

برای حذف زیر منو به روش قبل عمل می‌کنیم:

```
CommandBars("Worksheet menu bar").Controls("Tools")._
Controls("MyPopup").Delete
```

دکمه‌های فرمان یا Command Button

می‌توانیم کنترل‌ها را مستقیماً با استفاده از جعبه ابزار Control روی خود صفحه گسترده قرار دهیم. هر کنترلی را می‌توان به‌کار گرفت(مانند لیست‌های باز شو یا یک دکمه فرمان). از منوی صفحه گسترده، View | Toolbars | Control Toolbox را انتخاب کنید. جعبه ابزار Control ظاهر خواهد شد. آیکن Command Button را با کلیک روی آن و کشیدن روی صفحه گسترده انتخاب کنید (تصویر ۱-۱۱):



تصویر ۱-۱۱ قرار دادن مستقیم کنترل‌ها بر روی یک صفحه گسترده

روی دکمه فرمان کلیک راست کرده و View Code را انتخاب کنید. این کار باعث می‌شود به پنجره کدنویسی و زیرروال CommandButton1_click( ) بروید. سپس می‌توانید کد خود را در این‌جا بنویسید و یا زیرروال دیگری را فرا بخوانید. برای مثال می‌توانید یک کادر پیغام را روی کد قرار دهید.

می‌توانیم متن روی دکمه را با کلیک راست کردن و انتخاب Properties تغییر دهیم. خصوصیت Caption را به دلخواه ویرایش کنیم. این خصوصیت به صورت پیش فرض، Command Button1 است. می‌توانیم این مقدار را به MyButton یا هر عنوان دیگری تغییر دهیم.

وقتی کار روی دکمه تمام شد، با کلیک روی آیکن موجود در قسمت بالا سمت چپ پنجره از حالت طراحی در جعبه ابزار Control خارج شوید. حال وقتی روی دکمه کلیک کنید به جای این‌که وارد حالت طراحی شود، کد خود را اجرا می‌کند.

ممکن است بخواهید موارد دیگری را به دکمه اضافه کنید یا حتی در صورتی که از آن راضی نباشید، آن‌را پاک نمایید. تنها مشکلی که در حال حاضر وجود دارد این است که هر بار روی آن کلیک می‌کنید کد خود را اجرا کرده و چون VBA، کلیک راست ماوس را روی یک دکمه فرمان به عنوان یک رویداد قبول نمی‌کند، با راست کلیک کردن روی دکمه هیچ اتفاقی نمی‌افتد.

برای ویرایش کردن مجدد دکمه کافی است دوباره Control Toolbox | Toolbars | View را انتخاب کنید تا جعبه ابزار Control ظاهر شود. با کلیک روی آیکن Design Mode در قسمت بالا سمت چپ پنجره جعبه ابزار مجدداً به حالت طراحی وارد می‌شویم. می‌توانیم دکمه را انتخاب کنیم، آن‌را تغییر اندازه دهیم، حذف کنیم و ... .

کد دکمه را می‌توانیم بدون اجبار به رفتن در حالت طراحی تغییر دهیم. کد کاملاً مجزا از صفحه گسترده است. کد برای یک کاربرگ خاص، روی ماژول شئ همان کاربرگ ظاهر می‌شود می‌توانیم به سادگی آن‌را ویرایش کنیم.

بخش دوم

مدل‌های شئ

بخش همه چیز درباره مدل شئ Excel و چگونگی استفاده از آن در خواهید آموخت. مدل شئ Excel، مهم‌ترین بخش در برنامه‌نویسی برای Excel است. مدل شئ، شما را قادر می‌سازد تا هر کاری را که ن منوی Excel می‌توانید انجام دهید، با کدنویسی اجرا کنید.

ن یاد می‌گیرید که چگونه از مدل شئ Excel استفاده کنید تا با نامه‌های مایکروسافت ارتباط برقرار کنید. برای مثال خواهید آموخت ـه نامه‌های الکترونیکی اختصاصی را از طریق Microsoft Ou ارسال کنید و یا یک سند اختصاصی Word ایجاد

فصل دوازدهم

مدل شئ Excel

مدل شئ Excel، مهم‌ترین بخش در به‌کارگیری VBA در Excel است که با ارایه دستورات اضافی برای دستیابی به کاربرگ‌ها و کارپوشه‌ها و با ارایه کارآیی کاملی که کاربر معمولاً از ساختار منو به‌دست می‌آورد، برنامه‌نویسی در Excel را از برنامه‌نویسی در دیگر برنامه‌های کاربردی VBA متمایز می‌کند. برای مثال مدل شئ در Microsoft Access حاوی دستورات و اشیا ویژه‌ای در ارتباط با پایگاه داده است و ابزاری برای دست‌کاری جدول‌ها، رشته‌های پرس و جو، فرم‌ها و گزارش‌ها ارایه می‌کند. در Excel، کل برنامه روی ساختار کارپوشه‌ها و صفحه گسترده‌ها متمرکز است پس مدل شئ نیز در رابطه با این موارد نوشته شده است. Excel یک برنامه سه لایه‌ای است:

لایه خدمات سرویس گیرنده، مدل شئ و لایه خدمات داده‌ای.

رابط معمول صفحه گسترده را که می بینیم، لایه خدمات سرویس گیرنده است و همان لایه‌ای است که معمولاً با کاربر در ارتباط است. در زیر این لایه، مدل شئ Excel قرار گرفته است. هر بار کاری روی صفحه گسترده انجام می‌دهیم در واقع دستورات را از طریق مدل شئ Excel صادر می‌کنیم. برای مثال اگر یک کارپوشه را باز کنیم، کدی که زیر دستور File | Open اجرا می‌شود دقیقاً همان کارآیی دستور Workbooks.Open را برای باز کردن کارپوشه و اضافه کردن آن به شئ مجموعه کارپوشه‌ها ایفا می‌کند. به طور مشابه اگر یک مجموعه محاسباتی (Calculation) در کادر محاوره‌ای Options داشته باشیم و از کلید F9 برای محاسبه مجدد آن استفاده کنیم، کارآیی مشابه با دستور Application.Calculate در هر بار زدن این کلید مورد استفاده قرار می‌گیرد. با استفاده از مدل شئ Excel و یک زبان برنامه‌نویسی مانند ویژوال بیسیک، توسعه Excel front end ما با کارآیی دقیقاً مشابه با Excel front end مایکروسافت کار سختی نخواهد بود. هر دستور منو و کلید تابعی روی Excel front end مایکروسافت در مدل شئ Excel نیز نشان داده شده است. البته این به معنای آن نیست که آن‌ها دقیقاً همان اشیایی هستند که خود Excel از آن‌ها استفاده می‌کند بلکه مایکروسافت شما را با تمام اشیا و متدها تقویت می‌کند تا بتوانید هرکاری را که از طریق منوی Excel انجام می‌دهید، از طریق کدنویسی هم اجرا کنید.

اگر بخواهید front end مخصوص خود را بنویسید به دلیل آن‌که کارآیی کامل در مدل شئ وجود دارد، به کدنویسی کمی احتیاج دارید.

مدل شئ Excel حاوی لایه خدمات داده‌ای است. این لایه حاوی داده‌های صفحات گسترده است و توسط دستوراتی که از جانب مدل شئ Excel صادر می‌شوند، اصلاح می‌شود.

مدل شئ Excel حاوی تعداد زیادی شئ است که برای مثال می‌توان به Workbooks، Worksheets، Ranges، Charts، Pivot tables و Comments اشاره کرد. این اشیا Excel در واقع واحدهایی هستند که کارآیی‌های گوناگونی در تحلیل داده‌ها ارایه می‌کنند. از همه مهم‌تر این است که آن‌ها را می‌توان از طریق کد نویسی تحت کنترل قرار داد.

در هنگام برنامه‌نویسی در Excel با استفاده از VBA، از دستورات و توابع استاندارد VBA مثل If..Then..Else ، For..Next و MsgBox استفاده می‌کنیم اما از مدل شئ برای برقراری ارتباط با برنامه کاربردی Excel استفاده می‌نماییم.

یک شئ، قسمتی از برنامه Excel محسوب می‌شود. اشیا به صورت سلسله مراتبی مرتب می‌شوند. برای مثال در بالای مدل شئ، شئ Application قرار دارد که در واقع خود Excel است. زیر شئ Application، شئ Workbook قرار دارد و در شئ Workbook، اشیا Worksheet قرار دارند. در هر شئ Worksheet، اشیا Ranges قرار دارند و ... .

هر شئ می‌تواند حاوی تنظیمات (که خصوصیت یا Property نامیده می‌شود) و عملیاتی باشد که روی شئ اجرا می‌شوند (متد). برای مثال اگر بخواهیم داده‌هایی را با استفاده از کدنویسی در یک سلول وارد کنیم به خصوصیت Range از کارپوشه ارجاع خواهیم داد. ما یک سلول یا مجموعه‌ای از سلول‌ها را در خصوصیت Range مشخص می‌کنیم و سپس از خصوصیت text و Value برای قرار دادن داده‌ها در سلول‌ها استفاده می‌کنیم. به مثال زیر توجه کنید:

```
Worksheets ("sheet1") .Range("a1").Value = "MyData"
```

این کد،متن "MyData" را در سلول A1 از sheet1 درج می‌کند.

خصوصیات و متدها

تمام اشیا در مدل شئ Excel دارای خصوصیات، متدها یا هر دو هستند. بسته به پیچیدگی شئ، تعداد خصوصیات و متدها ممکن است کم یا زیاد باشد. خصوصیت، یک ویژگی اسکالر است که پارامترهای گوناگون یک شئ را مشخص می‌کند. برای مثال می‌توان به خصوصیت Visible کاربرگ اشاره کرد که می‌تواند مقدار xlSheetVisible(-1)، xlSheetHidden(0) و xlSheetVeryHidden(2) را داشته باشد و

نشان می‌دهد که آیا کاربرگ برای کاربر مرئی است یا پنهان. این کار در این‌جا با استفاده از ثابت‌های داخلی (مقادیر عددی داخل پرانتزها) انجام می‌شود. یا مثلاً شئ Workbooks دارای خصوصیت Count است که تعداد کارپوشه بارگذاری شده در برنامه Excel را مشخص می‌کند.

خصوصیات، حاوی پارامترهایی هستند که شئ را تعریف می‌کنند و بسیار وابسته به اشیایی که جزیی از آن‌ها می‌باشند، هستند. مقادیر خصوصیات می‌تواند متنی یا عددی باشد.

متدها، روش‌های اجرای عملیات‌ها بر مبنای یک شئ خاص هستند. آن‌ها یک راه میانبر برای انجام کاری روی یک شئ خاص ارایه می‌کنند. برای مثال ممکن است بخواهیم یک کاربرگ را از یک کارپوشه پاک کنیم. برای این کار متد Delete روی مجموعه کاربرگ‌ها، کاربرگی را که قرار است پاک شود مشخص می‌کند. همین‌طور متد Open، یک کارپوشه را باز می‌کند.

برای مثال کامپیوتر خود را در نظر بگیرید. کامپیوتر، یک شئ محسوب می‌شود که می‌توانیم خصوصیات و متدهایی برای آن تعریف کنیم. خصوصیت یک جنبه قابل سنجش شئ محسوب می‌شود پس برای کامپیوتر می‌توانیم خصوصیات زیر را داشته باشیم:

جدول ۱-۱۲

خصوصیت	مقدار
Make	Compaq
Year	2002
RAM	128 M
Hard Disk	20 GB
Processor	Intel

توجه کنید که تمام آن‌ها مقادیر عددی ندارند.

متدها،کلماتی هستندکه عملیاتی را که توسط یک شئ قابل انجام هستند، نشان می‌دهند. این کلمات درباره کامپیوتر شما می‌توانند موارد زیر باشند:

- Cold boot
- Warm boot
- Shut down
- Set up

اگر این مثال را به Excel منتقل کنیم و شئ Workbook را به عنوان مثال در نظر بگیریم، دارای خصوصیات بعد خواهیم بود:

جدول ۲-۱۲

خصوصیت	شرح
Author	فردی که کارپوشه را ساخته است.
FullName	نام کامل کارپوشه
HasPassword	مقدار True یا False را دارد که بسته به داشتن یا نداشتن کلمه عبور است.
Path	مسیر بار گذاری کارپوشه

البته خصوصیات دیگری نیز برای یک کارپوشه وجود دارند که ذکر آن‌ها در این‌جا ضروری نیست. موارد زیر را می‌توان به عنوان چند مثال برای متدهای شئ Workbook در نظر گرفت:

جدول ۳-۱۲

متد	شرح
Activate	باعث می‌شود این کارپوشه، کارپوشه فعال باشد.
Close	کارپوشه را می بندد.
NewWindow	یک پنجره جدید برای کارپوشه فعال ایجاد می‌کند.
PrintOut	کارپوشه را چاپ می‌کند.

خصوصیات، می‌توانند فقط خواندنی (read only) یا خواندنی/ نوشتنی باشند. فقط خواندنی یعنی می‌توانیم به مقدار یک خصوصیت (تنظیم آن) دسترسی داشته باشیم اما نمی‌توانیم آن‌را تغییر دهیم. خواندنی/ نوشتنی یعنی هم به مقدار یک خصوصیت دسترسی داریم و هم می‌توانیم آن را تغییر دهیم. این امر تأثیر مهمی در برنامه شما دارد چون ممکن است شما کدی برای نوشتن داده در خصوصیاتی بنویسید که فقط خواندنی هستند. برای مثال کارپوشه، دارای خصوصیتی با نام HasPassword است و بسته به محافظت از کارپوشه با یک کلمه عبور، مقدار True یا False می‌گیرد. ممکن است فکر کنید اگر کلمه عبور را ندانید می‌توانید با False قرار دادن مقدار این خصوصیت، کارپوشه را باز کنید اما مایکروسافت فکر این‌جا را کرده و این خصوصیت را به صورت فقط خواندنی درآورده است. برای حفظ یکپارچگی خصوصیات خاص با دیگر مدل‌ها لازم است که آن‌ها حتماً به صورت فقط خواندنی باشند. برای مثال خصوصیت Count از هر مجموعه شئ، همیشه فقط خواندنی است چون تغییر خصوصیت Count هر کاربرگی در یک کارپوشه می‌تواند نتایج غیر قابل پیش بینی به‌وجود آورد. برای مثال

کارپوشه ممکن است شامل ۴ کاربرگ باشد. اگر بتوانیم خصوصیت Count را به ۲ تغییر دهیم آن‌گاه دو کاربرگ ما حذف خواهند شد و این اصلاً قابل قبول نیست.

متدها به طور مؤثر مانند زیرروال‌ها یا راه‌های میانبر برای انجام عملیاتی هستند که می‌توان آن‌ها را از داخل کد فراخواند تا کارهای خاصی را انجام دهند (مانند باز کردن یک کارپوشه یا اضافه کردن یک کاربرگ جدید). نوشتن کدی برای باز کردن کارپوشه، غیر ممکن خواهد بود چون شما نکات ریزتر درباره قالب بندی فایل و این‌که هر بایت چه چیزی را نشان می‌دهد، نمی‌دانید. بدون VBA لازم خواهد بود تا ساختار زبان برنامه نویسی C را بدانید و یک کد منبع داشته باشید تا کارپوشه را باز کنید. البته مایکروسافت تمام این کارهای سخت را بر عهده ما گذاشته است.

دستکاری خصوصیات

اگر یک خصوصیت از نوع خواندنی/ نوشتنی باشد می‌توانیم مقدار آن‌را دستکاری کنیم. این به معنای آن است که بسته به شئ و خصوصیت آن می‌توانیم مقادیر دیگری را در خصوصیات آن شئ جایگزین کنیم تا تأثیرات متفاوتی به‌دست آوریم. برای مثال ممکن است بخواهیم از کدنویسی برای تغییر متن موجود در یک سلول روی یک کارپوشه استفاده کنیم. می‌توانیم این کار را با نوشتن متن جدید به عنوان مقدار خصوصیت Value برای آن سلول خاص و ارجاع به آن از طریق شئ Worksheet انجام دهیم.

خصوصیات عموماً با استفاده از کد در زمان اجرا تغییر می‌کنند (یعنی زمانی که برنامه اجرا می‌شود). البته امکان دسترسی به بعضی از خصوصیات در زمان طراحی هم وجود دارد و می‌توانیم آن‌ها را با استفاده از پنجره خصوصیات VBE تغییر دهیم. زمان طراحی وقتی است که پنجره کدنویسی را می‌بینیم و طراحی و تغییرات در کد قابل اجراست . برای دیدن مثال در این زمینه روی شئ ThisWorkbook در پنجره VBA کلیک کنید. روی خصوصیت Saved کلیک کنید تا به شما اجازه دهد مقدار آن‌را از True به False تغییر دهید. البته معمولاً این خصوصیات را با استفاده از کد و در جواب عملیاتی که کاربر انجام می‌دهد، تغییر می‌دهیم.

در این‌جا دو مثال از ساختار دستوری برای خواندن خصوصیات می‌آوریم:

```
MsgBox Workbooks("book1").Saved
MsgBox ActiveWorkbook.Saved
```

تمام مجموعه‌ها شاخص‌هایی دارند که اشیا مستقل در آن مجموعه را تعریف می‌کنند. عنوان "book1" که در پرانتز نشان داده شده است، تعریف می‌کند که آن، book1 است و در مجموعه Workbooks قرار

دارد و کد نیز به این مجموعه استناد می‌کند. ممکن است در یک لحظه چندین کارپوشه بارگذاری شده باشند و بر عهده VBA است که بین آن‌ها فرق بگذارد.

برخی اشیا، در اشیا یا مجموعه‌های دیگر گروه‌بندی شده‌اند. برای مثال Excel می‌تواند چندین کارپوشه را به صورت باز نگه دارد. هر کارپوشه مستقل، یک شئ محسوب می‌شود. تمام کارپوشه‌های که در یک لحظه در برنامه Excel باز هستند در شئ یا مجموعه Workbooks گروه‌بندی شده‌اند. دستیابی به یک عضو مستقل در یک مجموعه، مستلزم این است که یا موقعیت عددی آن‌را در مجموعه مشخص کنیم یا از طریق نامش به آن دسترسی پیدا کنیم. مثال قبلی در مجموعه Workbooks به یک کارپوشه با نام book1 دسترسی پیدا می‌کرد .

خط اول در کد قبلی، وضعیت Saved از کارپوشه book1 (True یا False) را مشخص می‌کرد و این وضعیت را در یک کادر پیغام نمایش می‌داد. خط دوم کد قبلی، وضعیت Saved از کارپوشه فعال را در یک کادر پیغام نشان می‌داد. اگر book1 تنها کارپوشه بارگذاری شده در Excel باشد هر دو کد فوق یک نتیجه را نشان خواهند داد. اگر بیش از یک کارپوشه کاری بارگذاری شده باشد کد دوم وضعیت Saved کارپوشه‌ای را که کاربر در حال حاضر روی آن کار می‌کند، نشان می‌دهد. همان‌طور که قبلاً گفته شد، کارپوشه یک شئ است و Saved یک خصوصیت برای آن محسوب می‌شود. اگر کار پوشه ذخیره شده و تغییر جدیدی روی آن اعمال نشده باشد، کادر پیغام مقدار True را نشان خواهد داد. اما اگر کارپوشه تغییری کرده باشد و هنوز ذخیره نشده باشد کادر پیغام مقدار False را برخواهد گرداند.

توجه کنید که از کاراکتر نقطه به عنوان جدا کننده شئ و خصوصیت استفاده شده است. چون اشیا می‌توانند حاوی اشیا فرعی (subobject) و خصوصیات می‌توانند شامل خصوصیات فرعی (subproperties) باشند می‌توانیم بیش از یک جدا کننده نقطه داشته باشیم. برای مثال یک کارپوشه مجموعه‌ای از چندین کاربرگ است. پس یکی از خصوصیات شئ Workbook، مجموعه Worksheets است. اگر بخواهیم به یک کاربرگ در خارج از مجموعه ارجاع کنیم از کدی مانند کد زیر استفاده می‌کنیم:

```
MsgBox Workbooks("book1").Worksheets("sheet1").ProtectContents
```

این کد بسته به این‌که آیا محتویات book1 یا sheet1 با کلمه عبور محافظت شده باشند مقدار True یا False را برمی‌گرداند. خصوصیت ProtectContents بر طبق این‌که آیا کاربرگ کلمه عبور داشته باشد یا نه مقدار True یا False را برمی‌گرداند.

می‌توانیم در صورتی‌که خصوصیات از نوع فقط خواندنی نباشند مقدارشان را تغییر دهیم. برای مثال، برای تنظیم خصوصیت Saved در یک کارپوشه به مقدار True (بدون توجه به این‌که آیا ذخیره شده است یا خیر) می‌توانیم از کد زیر استفاده کنیم:

```
Workbooks("book1").Saved = True
```

نتیجه این است که اگر کارپوشه را ببندیم، دیگر اعلان ذخیره کردن آن‌را دریافت نخواهیم کرد با این مثال متوجه می‌شوید که چرا تأکید داریم هنگام کار روی خصوصیات، حتماً از نتیجه کارتان آگاه باشید در غیر این صورت ممکن است مثل این مثال نتایج غیر مترقبه‌ای به‌دست آورید.

فراخوانی متدها

همان‌طور که قبلاً شرح داده شد، متدها در اصل زیرروال‌هایی هستند که بر مبنای اشیایی ساخته می‌شود که عملیات خاصی را انجام می‌دهند. متدها گاهی اوقات وابسته به پارامترهایی هستند که به آن‌ها انتقال می‌یابند. متد در اصل راه میانبر برای انجام یک عملیات است اما ممکن است لازم باشد پارامترهایی را مشخص کنیم تا برای VBA مشخص کند که دقیقاً باید چه عملیاتی را انجام دهند.

یک مثال از این قضیه، باز کردن یک کارپوشه از یک فایل است. برای انجام این کار از متد Open روی مجموعه Workbooks استفاده می‌کنیم البته باید پارامترهایی مانند نام فایل و مسیر را هم انتقال دهیم تا VBA دقیقاً بداند چه چیزی را باز کند. برای مثال، در کد زیر:

```
Workbooks.Open ("c:\MyFile.xls")
```

c:\MyFile.xls، محل فایلی را که قرار است باز شود، تعریف می‌کند که البته یک پارامتر اجباری برای این متد است. پارامترهای اختیاری دیگری هم وجود دارند که می‌توانند انتقال یابند، مانند کلمه عبور (در صورت نیاز) و پارامتر فقط خواندنی بودن (read_only) فایل.

گاهی اوقات، مانند وقتی که کارپوشه را با متد Save در محل اصلی آن ذخیره می‌کنیم، نیازی نیست متد آرگومان داشته باشد:

```
Workbooks("book1") .Save
```

این کد، عملیات ذخیره کردن را روی book1 انجام می‌دهد، با این فرض که این فایل قبلاً ذخیره شده است. این کد مشابه دستور File | Save از منوی Excel عمل می‌کند. با وجود این اگر فایل قبلاً ذخیره نشده باشد، نام فایل به صورت پیش فرض، book1.xls خواهد بود و در مسیر پیش فرض تنظیم

شده توسط Excel یا آخرین پوشه‌ای که از طریق عملیات Open یا Save انتخاب شده است، قرار می‌گیرد.
هم‌چنین می‌توانیم از متد SaveAs نیز استفاده کنیم که پارامترهایی برای ذخیره کردن یک کارپوشه با نام و محل متفاوت دارد:

```
Workbooks("book1") .SaveAs ("newfile.xls", ,"apple")
```

این کد، پارامتر "newfile.xls" را به عنوان نام فایل انتقال می‌دهد. این عملیات را انتقال ترتیبی می‌نامیم چون پارامترها به ترتیبی که در تابع تعریف شده‌اند، انتقال می‌یابند. دقت کنید که هر پارامتر با کاراکتر کاما از پارامتر بعدی جدا شده است. در این مورد، نام فایل newfile.xls را انتقال می‌دهیم و پارامتر قالب فایل (که اختیاری است) را نادیده گرفته و مقدار پارامتر password را برابر "apple" در نظر می‌گیریم.
بعضی از متدها مانند SaveAs پارامترهای بیشتری دارند که البته بسیاری از آن‌ها هم اختیاری هستند. پارامترهای اختیاری داخل [ ] نشان داده می‌شوند. انتقال پارامترها به صورت ترتیبی در هنگام کار با پارامترهای اختیاری، پیچیده می‌شود چون ممکن است از تابعی استفاده کنیم که ۱۰ پارامتر اختیاری داشته باشد و ما فقط بخواهیم از دوتای آن‌ها استفاده کنیم. مثال زیر را برای باز کردن یک کارپوشه بررسی کنید:

```
Workbooks.Open "c:\MyFile.xls",  , True,  , "apple"
```

می‌بینید که سه پارامتر برای باز کردن فایل، انتقال می‌یابند و حداقل دو پارامتر به کار گرفته نشده و با مقادیرNull که بین دو کاما قرار دارند، این دو پارامتر مشخص شده‌اند. این کاماها به خاطر آن است که با توجه به ترتیب پیش فرض پارامترها، در آن محل باید پارامتری قرار گیرد و چون این پارامتر مورد نیاز نیست، مقداری هم به آن اختصاص نیافته است. برای مثال، تمام فایل‌ها کلمه عبور ندارند پس وقتی یک کارپوشه در این روش باز می‌شود، پارامتر کلمه عبور (password) کاملاً اختیاری است و تقسیم پارامترها به اجبار و اختیاری، وابسته به کاری که متد انجام می‌دهد و چگونگی کدنویسی در Excel Object Library است. همیشه دو طرف پارامترهای اختیاری، [ ] قرار می‌گیرد.
نام فایل، اجباری است، مقدار پارامتر read_only ، True تنظیم می‌شود و کلمه عبور هم "apple" است. این کمی گیج کننده است و هر کس کد را می‌خواند نمی‌تواند بلافاصله تفسیر کند که کد چه عملی انجام می‌دهد و معنای پارامترها چیست.
انتقال با نام (passing by name)، روش دیگر انتقال پارامتر است که باعث سر درگمی کمتری می‌شود و نام پارامترهایی را که انتقال می‌یابند نشان می‌دهد. انتقال با نام، ما را قادر می‌سازد تا آرگومان‌ها را

به صورت گزینشی انتقال دهیم بدون آن‌که مجبور باشیم برای آرگومان‌هایی که نمی‌خواهیم استفاده کنیم از مقادیر Null استفاده کنیم. این شیوه هم‌چنین درک آن‌چه را که به متد انتقال می‌یابد، ساده‌تر می‌کند. اگر پارامترها با نامشان انتقال یابند، می‌توانیم مثال قبل را به صورت زیر بازنویسی کنیم:

```
Workbooks.Open FileName:="c:\myfile.xls", ReadOnly:=True, Password:= "apple"
```

می‌توانیم هر پارامتر را با نام‌گذاری پارامتر و قرار دادن کاراکترهای (=:) پس از آن تعریف کنیم. در هنگام انتقال پارامترها با نامشان می‌توانیم پارامترها را به هر ترتیبی انتقال دهیم و این بر عکس شیوه ترتیبی است که باید در آن، ترتیب انتقال دقیقاً رعایت می‌شد.
هم‌چنین می‌توانیم این فایل را به صورت زیر، با نام دیگری ذخیره کنیم:

```
Workbooks("book1") .SaveAs FileName:="newfile.xls"
```

اگر این کد را اجرا کرده و فایل را با نام newfile.xls ذخیره کنیم، در کدهای بعدی باید با نام جدید به کارپوشه ارجاع کنیم و در غیر این صورت پیغام "Subscript out of range" صادر خواهد شد که به معنای آن است که نام فایل قبلی(book1) دیگر یافت نخواهد شد. مثال زیر دقیقاً به فایل تازه ذخیره شده استناد می‌کند:

```
Workbooks("newfile.xls") .Worksheets("sheet1")
```

مجموعه‌ها (Collections)

در برنامه‌های شئ‌گرا، درک مفهوم مجموعه‌ها از اهمیت برخوردار است. Collections، اشیایی هستند که حاوی گروهی از اشیا مشابه هستند. یک مثال، مجموعه Worksheets است که حاوی تمام اشیا کاربرگ برای یک کارپوشه مشخص است. تمام کاربرگ‌ها، اشیا مشابه هستند چون آن‌ها دارای متدها و خصوصیات مشابهی هستند. یک شئ مانند Chart دارای خصوصیات و متدهای متفاوتی است پس نمی‌تواند بخشی از مجموعه Worksheets باشد اما می‌تواند در مجموعه Charts قرار گیرد.
در Excel، تمام اشیا یا اشیا منفرد هستند که با نام به آن‌ها ارجاع می‌شود یا اشیایی در مجموعه‌ها هستند که با شماره شاخص یا نام به آن‌ها ارجاع می‌شود. مجموعه‌ها، غیر از خصوصیات و متدهای اشیا درونشان، خصوصیات و متدهای خاص خود را دارند. برای مثال، خصوصیت Count نشان دهنده تعداد اشیا موجود در یک مجموعه است و مجموعه‌ها برای اضافه کردن یک شئ جدید هم دارای متد Add هستند.
مجموعه‌ها، خصوصیات و متدهای خاص خود را دارند که باعث می‌شود کاملاً از اشیا درونشان متمایز شوند. اشیا نیز دارای خصوصیات و متدهای خاص خود هستند و ممکن است حاوی مجموعه‌های

دیگری از اشیا نیز باشند. یک مثال، مجموعه Workbooks است حاوی مجموعه اشیا Workbook است و تمام کارپوشه‌ها اخیراً بارگذاری شده را در Excel نشان می‌دهد. همان‌طور که قبلاً گفتیم، این مجموعه دارای خصوصیت Count است تا تعداد کارپوشه‌ها را بشمارد و هم‌چنین برای بارگذاری یک کارپوشه دیگر متد Open را دارد. هر کارپوشه خصوصیاتی مانند HasPassword و متدهایی مانند Save و SaveAs دارد. با وجود این در هر کارپوشه، مجموعه‌ای از کاربرگ‌ها هم وجود دارد که آن‌ها هم دارای خصوصیات، متدها و مجموعه‌های خاص خود هستند.

در Excel می‌توانیم در داخل یک کارپوشه، مجموعه‌ای از کاربرگ‌ها را داشته باشیم که مجموعه Worksheets نامیده می‌شوند و هر کاربرگ داخل این مجموعه یک شماره شاخص و نامی برای شناسایی آن دارد. مجموعه‌های دیگر مانند Windows، ChartObject و Borders نیز وجود دارند اما مجموعه‌های Workbooks و Worksheets، اشیا اصلی برای ارجاع به سلول‌ها هستند پس در این فصل تنها روی آن‌ها متمرکز می‌شویم.

مجموعه‌ها را می‌توان در یک چرخه مرور کرد. مرور در چرخه، بهترین اصطلاح برای تشریح آن‌چه در یک حلقه For Each..Next رخ می‌دهد است. حلقه‌های For Each..Next را در فصل ۴ بررسی کردیم. برای این حلقه‌ها از ساختار دستوری زیر استفاده می‌کنیم:

```
For Each Object within Collection, Next.
```

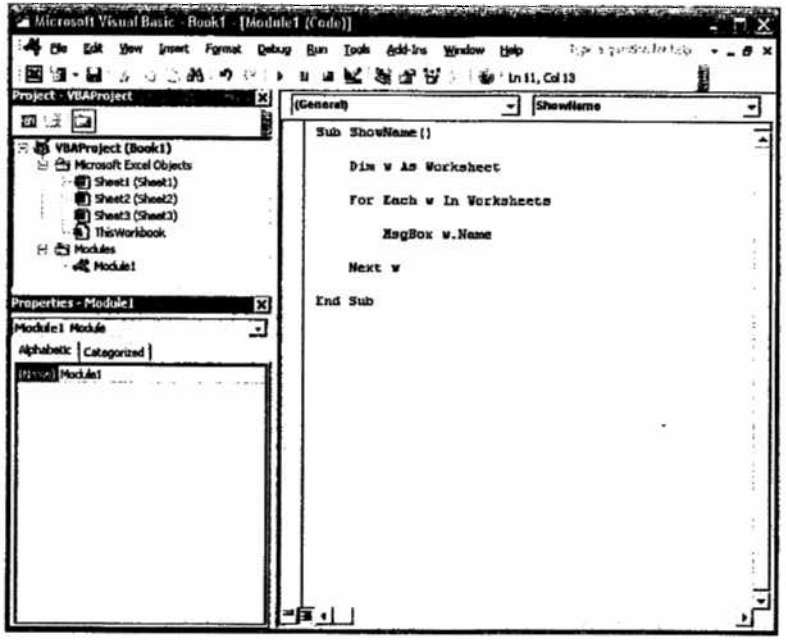
این دستور، هر شئ در مجموعه را در یک چرخه قرار داده و این امکان را به وجود می‌آورد تا خصوصیات آن شئ را بررسی یا دستکاری کنیم یا با فراخوانی یک متد، از آن استفاده کنیم (برای مثال، Workbooks("book1").Save).

کد زیر را در یک ماژول VBA قرار دهید. اگر ماژول در اختیار ندارید از منوی، | Insert | Module Visual Basic Editor را انتخاب کنید. دکمه F5 را بزنید تا کد اجرا شود و خواهید دید که مانند تصویر 1-12، نام هر برگه به نمایش در می‌آید:

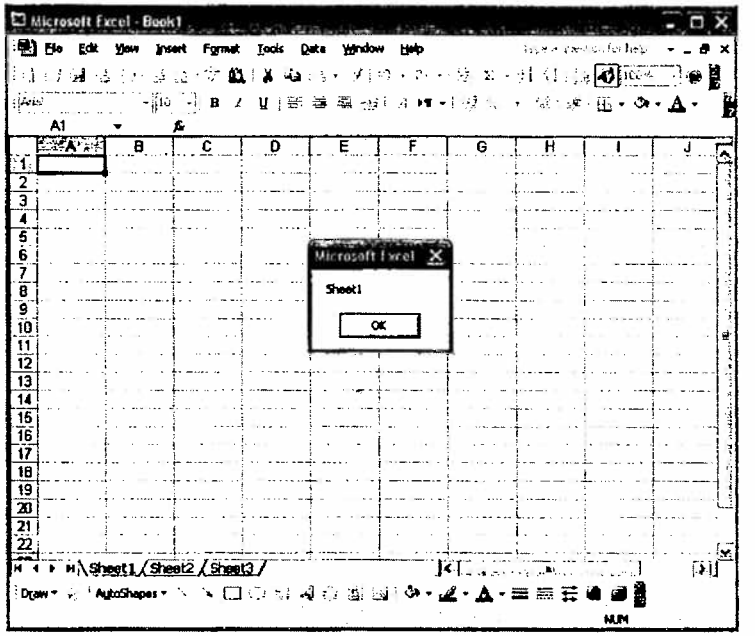
```
Sub ShowName ( )
Dim w As Worksheet
For Each w In Worksheets
        MsgBox w.Name
Next
End Sub
```

در آغاز، این کد یک متغیر با نام w ایجاد می‌کند تا شئ کاربرگ را نمایش دهد. این متغیر، کاربرگ فعلی را که در حلقه مرور می‌شود، نشان می‌دهد. کد به نوبت هر کاربرگ را می‌گیرد، خصوصیت نام آن

را مشخص می‌کند و آن نام را در یک کادر پیغام نمایش می‌دهد (تصویر ۲-۱۲). برگه کد ما باید مانند تصویر ۱-۱۲ باشد.



تصویر ۱-۱۲ نمایش نام تمام کاربرگ‌ها در مجموعه Worksheet



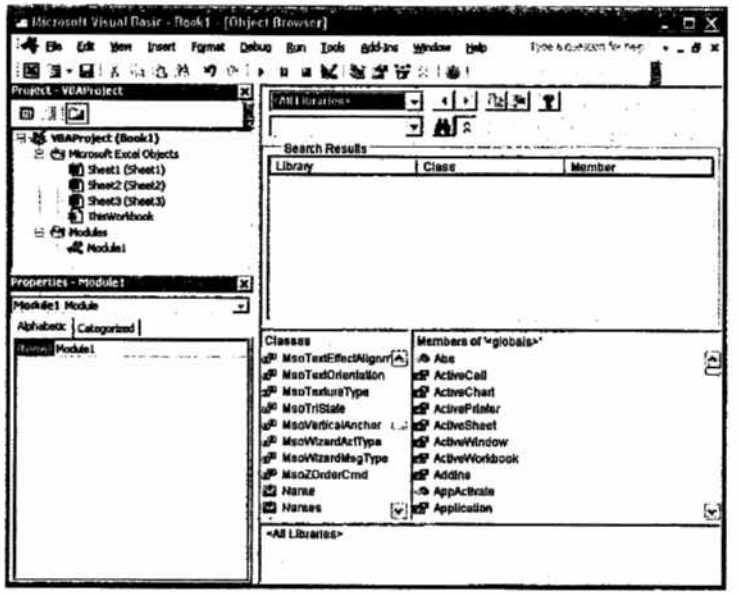
تصویر ۲-۱۲ کادر پیغامی که نام کاربرگ‌ها را نمایش می‌دهد.

دستور Dim (که مخفف Dimension است)، فضایی در حافظه برای یک متغیر ایجاد می‌کند. در این مورد، این دستور فضایی برای یک کاربرگ استاندارد ایجاد می‌کند. می‌توانیم این روال را بدون دستور Dim هم اجرا کنیم اما به‌کارگیری آن دارای این مزیت است که در به‌کارگیری خصوصیات و متدها به ما کمک خودکار می‌دهد. وقتی دستور Dim w As را تایپ کنیم. یک کادر لیستی پس از تایپ w ظاهر می‌شود که تمام خصوصیات و متدهایی را که برای آن قابل استفاده هستند، نشان می‌دهد. این امر در هنگام برنامه‌نویسی بسیار سودمند است چون به ما اجازه می‌دهد ببینیم برای بخش بعدی کد، چه گزینه‌هایی برای انتخاب وجود دارند. ساختار مدل شئ Excel بی‌نهایت پیچیده است و انتظار نمی‌رود هر شئ، مجموعه، خصوصیت و متد موجود در آن را در حافظه خود داشته باشد. اگر از کادرهای لیستی اتوماتیک که به موازات تایپ کد شئ ظاهر می‌شوند، استفاده نکنیم برای این اطلاعات

باید دایماً به Object Browser استناد کنیم چون دانستن دستور بعدی که به‌کار می‌گیریم، مشکل است.

به‌کارگیری Object Browser

Object Browser، یک ابزار مفید برای مرور خصوصیات، متدها و ثابت‌های یک شئ است، در این مورد، شئ Application مدنظر است. برای دستیابی به Object Browser، منوی VBE، View| ObjectBrowser را انتخاب کنید یا دکمه F2 را فشار دهید. از منوی باز شو، از <All Libraries> استفاده کنید تا Object Library را پیدا کنید و سپس روی آن کلیک نمایید. این باعث می‌شود تمام کلاس‌های شئ Excel و خصوصیات و متدهای آن به نمایش درآیند. هم‌چنین روابط و سلسله مراتب خود اشیا را هم نشان می‌دهد. تصویر ۳-۱۲ چگونگی نمایش آن را روی صفحه نمایش نشان می‌دهد.



تصویر ۳-۱۲ مدل شئ Excel در Object Browser

با وارد کردن رشته جستجو در کادری که زیر منوی بازشو وجود دارد و سپس کلیک بر روی نماد دوربین یا زدن Enter، یک رشته خاص را جستجو می‌کنیم. برای مثال، ممکن است بخواهیم بدانیم کدام بخش‌های شئ Excel با نمودارها (Chart) سر و کار دارند. کافی است کلمه Chart را در کادر Search تایپ کنیم وEnter را بزنیم تا تمام ارجاع‌هایی را که حاوی کلمه Chart هستند، ببینیم. این کار بسیار ساده‌تر از جستجوی ساختار سلسله مراتبی است.

می‌توانیم روی یک کلاس (که یک شئ محسوب می‌شود) کلیک کنیم تا تمام خصوصیات، متدها و مجموعه‌های زیر آن را ببینیم. مثلاً اگر کلمه Chart را جستجو می‌کنید، روی Class Chart (که مجموعه واقعی تمام نمودارها است) کلیک کنید و متدها و خصوصیات موجود در کادر Members در سمت راست کلاس را ببینید. متدها دارای آیکن سبز رنگ و خصوصیات دارای آیکن خاکستری هستند که یک دست آن را نگه داشته است.

اگر روی یک خصوصیت کلیک کنیم نشان داده خواهد شد که آیا فقط خواندنی است یا می‌توانیم در آن بنویسیم. پنجره‌ای که در پایین Object Browser قراردارد، نوع خصوصیت و read-only بودن آن‌را نشان می‌دهد. با کلیک روی یک متد، ساختار دستوری پارامترهای آن و این‌که کدام یک اجباری یا اختیاری هستند به نمایش درمی‌آید. پنجره‌ای که درپایین Object Browser است، ساختار دستوری کامل متد را همراه با خصوصیات اختیاری که در بین کاراکترهای [ ] قرار گرفته‌اند، نشان می‌دهد.

برقراری ارتباط با صفحه گسترده

یکی از کاربردهای اصلی VBA در Excel، برقراری ارتباط با صفحات گسترده و دستکاری مقادیر سلول‌ها است. برای انجام این‌کار باید از شئ Range استفاده کنیم. شئRange ، نوعی پیوند بین شئ و مجموعه موجود در آن است که برای ارجاع به یک سلول یا مجموعه‌ای از سلول‌ها قابل استفاده است. برای مثال، کد زیر برای ارجاع به یک سلول است:

```
Workbooks("book1").Worksheets("sheet1").Range("a1").Value = 10
```

این کد را در یک ماژول قرار دهید و با کلیک روی مکان‌نما روی رویه، آن را اجرا کنید. دکمه F5 را فشار دهید و مقدار سلول A1 در sheet1 واقع در book1 را کنترل کنید. مقدار آن باید ۱۰ باشد.

هم‌چنین می‌توانیم به یک مجموعه (دامنه) از سلول‌ها ارجاع کنیم:

```
Workbooks("book1").Worksheets("sheet1").Range("a1.a10").Value = 5
```

این کد باعث پرشدن سلول‌های A1 تا A10 با مقدار 5 می‌شود. توجه کنید که برای جدا ساختن

سلول‌ها، از نقطه (.) در بین a1 و a10 استفاده شده است.

این تنها شیوه جدا سازی سلول‌ها نیست و مایکروسافت چندین گزینه دیگر نیز در این‌باره عرضه می‌کند. مثلاً می‌توانیم از کاراکترهای (,) ,(:) و یا (..) هم استفاده کنیم:

```
Workbooks("book1").Worksheets("sheet1").Range("a1","a10").Value = 5
```

بر عکس این‌کار هم ممکن است. یعنی می‌توانیم مقداری را از A1 بخوانیم و در کد، مورد استفاده قرار دهیم. می‌توانیم برنامه‌ای بنویسیم که مقادیر را از کاربرگ بخواند،آن‌ها را به نحوی پردازش کرده و سپس آن‌ها را مجدداً در همان کاربرگ یا حتی در یک محل جدید قرار دهد:

```
MsgBox Workbooks("book1").Worksheets("sheet1").Range("a1").Value
```

خروجی این کد، عدد 5 خواهد بود. البته اگر در این حالت بخواهیم مقدار چندین سلول را بخوانیم با پیغام خطای Type Mismatch مواجه می‌شویم. خطای Type Mismatch زمانی رخ می‌دهد که بخواهیم داده‌ای را در متغیری قرار دهیم که این نوع داده را قبول نمی‌کند. برای مثال اگر بخواهیم یک رشته متن را در متغیری از نوع integer (عددی) قرار دهیم، این خطا رخ می‌دهد.

وقتی مجموعه‌ای از سلول‌ها را می‌خوانیم، خصوصیت Value از شئ Range هر بار تنها می‌تواند مقدار یک سلول را برگرداند. اگر آن را وادار کنیم که مجموعه‌ای از سلول‌ها را بخواند مجبور می‌شود حجم بالایی از اطلاعات را برگرداند و چون برای نگه داشتن یک آرایه از اطلاعات طراحی نشده است، باعث بروز خطای مذکور می‌شود.

احتمالاً تا به حال متوجه شده‌اید که کد تا چه میزان به ساختار درختی اشیا ارجاع می‌کند. برای مثال می‌توانیم کار را از شئ Application آغاز کنیم. به یک کارپوشه در مجموعه Workbooks که زیر آن قرار دارد ارجاع کرده و در نهایت به یک دامنه یا یک مقدار ارجاع کنیم. شئ Application، ریشه محسوب می‌شود و اشیا Workbook و Worksheet نیز شاخه‌های آن هستند و شئ Range، برگ محسوب می‌شود. البته اگر با چند خط کد سروکار داشته باشیم، کار به همین راحتی‌ها قابل تفسیر نخواهد بود. برای مثال می‌توانیم به نام یک کاربرگ ارجاع کنیم:

```
MsgBox Worksheets("sheet1").Range("a1").Value
```

این کد جواب می‌دهد اما فرض کنید که بیش از یک کارپوشه بارگذاری شده باشد و هر دوی آن‌ها نیز دارای کاربرگی با نام sheet1 باشند. حال تکلیف چیست؟ Excel، کاربرگ موجود در کارپوشه فعال را انتخاب می‌کند و ممکن است این اصلاً باب میل ما نباشد. خوشبختانه روشی برای قطع کردن ارجاع‌ها

و حفظ یکپارچگی کد با استفاده از دستور Dim برای ایجاد یک شئ Workbook در حافظه کامپیوتر وجوددارد.

ایجاد شئ Workbook در حافظه

وقتی شما Workbook را در حافظه ایجاد می‌کنید، با تعیین ابعاد و نوع یک متغیر به وسیله دستور Dim، آن متغیر را برای نمایش دادن آن کارپوشه تعریف می‌کنید. می‌توان نام متغیر را به دلخواه تعیین کرد اما نباید از موارد ذکر شده در فصل ۲ تخطی کنیم.

مزیت ایجاد شئ Workbook این است که می‌توان آن را با دستور Set طوری تنظیم کرد تا یک کارپوشه خاص را نمایش دهد و از آن پس می‌توانیم از آن متغیر برای ارجاع به آن کارپوشه استفاده کنیم و کادرهای فهرست اتوماتیک، خصوصیات، متدها و مجموعه‌های زیرین آن را نشان خواهد داد. البته می‌توانیم بدون دستور Set هم کار کنیم اما این به معنای آن است که دیگر، کادرهای فهرستی را در اختیار نخواهیم داشت و در هر خط کد باید سلسله مراتب کامل را ذکر کنیم:

```
Dim w As Workbook, s As Worksheet

Set w = Workbooks("book1")

Set s = w.Worksheets("sheet1")

MsgBox s.Range("a1") .Value
```

دستور Dim، دو متغیر ایجاد می‌کند:

w، یک کارپوشه و s، یک کاربرگ است.

دستور نخست Set،متغیر w را طوری تنظیم می‌کند که به book1 در مجموعه Workbooks اشاره کند.

دستورSet دوم، متغیر s را طوری تنظیم می‌کند که به sheet1 در مجموعه Worksheet متعلق به w که قبلاً برابر با book1 تنظیم شده است، اشاره داشته باشد.

حال می‌توانیم از متغیر s به عنوان شئ Worksheet استفاده کنیم. این امر دارای این مزیت است که تمام کادرهای فهرستی خصوصیات و متدها به موازات تایپ کردن کد به صورت خودکار ظاهر می‌شوند تا گزینه‌های موجود برای آن شئ را نمایش دهند. s را تایپ کنید و پس از آن، یک کاراکتر نقطه (.) قرار دهید و خواهید دید که کادر فهرستی در کدتان ظاهر می‌شود. برای تکمیل کد لازم است تنها روی گزینه مورد نیاز، کلیک کنیم.

سلسله مراتب

در مدل شئ Excel، یک سلسله مراتب برای اشیا وجود دارد. به خاطر محدودیت‌هایی که در ارجاع به اشیا وجود دارد، درک این سلسله مراتب بسیار مهم است. برای مثال در اغلب سازمان‌ها، یک سلسله مراتب از مشاغل وجود دارد. در نیروهای مسلح هم این سلسله مراتب را دیده‌اید. در هر سلسله مراتبی، دستورات از بالا به پایین ارسال می‌شوند. مدل شئ Excel هم بسیار مشابه با این حالت کار می‌کند.

برای صدور دستورات، سلسله مراتب بسیار مهم است و ترتیبی که این دستورات صادر می‌شوند باید رو به پایین سلسله مراتب باشند. برای مثال در مدل شئ Excel، یک شئ Worksheet دارای خصوصیات یا متدهایی (دستورات) نیست که برای شئ Workbook یا شئ Application قابل استفاده باشند. نمی‌توانیم از شئ Workbook استفاده کنیم و سپس برای ذخیره کردن کارپوشه دستوری صادر کنیم. برای مثال:

```
Worksheets("sheet1").Workbooks("book1").Save
```

این کد جواب نمی‌دهد چون در ساختار سلسله مراتب، شئ Workbooks بالاتر از شئ Worksheets قرار دارد و این کد تمام قوانین سلسله مراتب را زیر پا می‌گذارد.

اگر این مثال را در یک پنجره کد نویسی تایپ کنید، منوی بازشو چیزی را نشان نخواهد داد که بتوان روی کارپوشه‌ها اجرا کنیم (مگر آن‌که از خصوصیت Parent استفاده کنیم). اگر بازهم در تایپ این کد پافشاری کنیم، کد به رنگ قرمز درمی‌آید و یک کادر پیغام خطا صادر خواهد شد.

بالاترین شئ در ساختار سلسله مراتب را Application می‌نامیم که در واقع نشان‌دهنده خود Excel است. پرکاربردترین مجموعه‌های شئ در زیر آن به شرح زیر هستند:

جدول ۴-۱۲

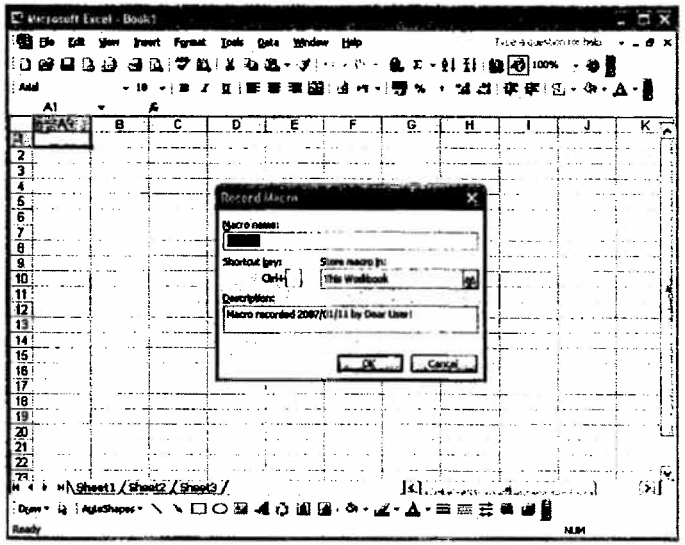
Collection	شرح
Dialogs	مجموعه تمام کادرهای محاوره‌ای داخلی Excel
Windows	برای دستیابی به تمام پنجره‌های Excel به کار می‌رود.
Workbooks	مجموعه کارپوشه‌ها در Excel
CommandBars	مجموعه تمام گزینه‌های منو داخلی Excel
Worksheets	مجموعه تمام کاربرگ‌ها در کارپوشه‌های Excel

لایه سوم، چهارم و پنجم ساختار سلسله مراتب شامل اشیا بیشتری برای دسترسی به کارآیی اشیا لایه دوم است. ساختار شئ Excel دارای ساختار شبه درختی است. شئ Application، ریشه محسوب

می‌شود، اشیا Workbook ،بدنه، اشیا Worksheet،شاخه و سلول‌ها (Cells) نیز برگ محسوب می‌شوند. برای مثال اگر از شئ Workbook به سمت پایین برویم به کاربرگ‌ها، پنجره‌ها و نمودارها می‌رسیم. ساختار مدل شئ به تفصیل در فصل ۱۳ بررسی شده است.

ضبط کردن ماکرو

یک ماکروی Excel، مجموعه‌ای از دستورالعمل‌ها است که عملیاتی را روی کاربرگ Excel انجام می‌دهد. ماکرو، یک روش جالب برای مشاهده این مورد است که چگونه عملیات خاصی روی صفحه گسترده در کد تفسیر می‌شوند و این‌که مدل شئ Excel چگونه برای انجام آن به کار گرفته می‌شود.
وقتی یک ماکرو را ضبط می‌کنیم، یک ماژول ایجاد شده و کد در آن نوشته می‌شود. می‌توانیم این کد را مرور کنیم و ببینیم Excel چگونه مسأله را در VBA حل می‌کند. می‌توانیم این کد را اصلاح کنیم یا در رویه‌های دیگر به‌کار گیریم.
برای ضبط کردن یک ماکرو از منوی Excel، Tools | Macro | Record New Macro را انتخاب کنید. صفحه نمایش شبیه تصویر ۴-۱۲ خواهدشد.



تصویر ۴-۱۲ ضبط کردن یک ماکرو

می‌توانیم نام ماکرو را تغییر دهیم یا برای آن، کلید میانبر تعریف کنیم. قوانین نام‌گذاری ماکروها، مانند قوانین نام‌گذاری توابع یا زیر روال‌ها در VBA است روی OK کلیک کنید تا پنجره کوچک Stop Recording ظاهر شود.
از این لحظه تا زمانی‌که روی دکمه Stop Recording کلیک کنیم، همه کارهایی که روی صفحه گسترده انجام می‌دهیم با استفاده از مدل شئ Excel به کد VBA ترجمه می‌شود. برای مثال، دامنه سلول‌های C8 و F16 را روی sheet1 انتخاب کنید. به sheet2 بروید و سلول D10 را انتخاب کنید. از منوی Edit | Paste،Excel را انتخاب کنید. حال روی Tools | Macro | Stop Recording کلیک کنید و ماکرو کامل خواهد شد.
با فشار دادن ALT-F11، نگاهی به پنجره کدنویسی بیندازید و متوجه خواهید شد که یک ماژول جدید درج شده است که حاوی کد مربوط به ماکرویی است که ضبط کرده‌ایم:

```
Sub Macro1( )
'
' Macro1 Macro
' Macro recorded 17-03-03 by Richard Shepherd
'

'
    Range("C8:F16") .Select
    Selection.Copy
    Sheets("Sheet2") .Select
    Range("D10") .Select
    ActiveSheet .Paste
End Sub
```

توضیحات به صورت خودکار اضافه شده‌اند تا نشان دهند چه کسی ماکرو را ضبط کرده است و در چه تاریخی این‌کار انجام شده است. دستور نخست، دامنه C8 تا F16 را انتخاب می‌کند. سپس ازمتد copy برای شئ Selection استفاده شده است. شئ Selection نشان‌دهنده دامنه‌ای است که انتخاب شده است. سپس sheet2 انتخاب شده و در ادامه، به وسیله دستور Range، سلول D10 روی آن انتخاب شده است و در آخر نیز از متد Paste متعلق به شئ ActiveSheet استفاده شده است.
می‌توانیم ببینیم Excel چگونه از مدل شئ Excel برای انجام دستورات ما استفاده می‌کند. البته Excel فرضیاتی نیز مد نظر دارد چون وقتی این ماکرو خاص ضبط شد فقط یک کارپوشه بارگذاری شده بود. اگر بیش از یک کارپوشه بارگذاری شده باشد، ماکرو روی هر کارپوشه‌ای که به عنوان کارپوشه فعال باشد، عمل می‌کند و ممکن است این اصلاً باب طبع ما نباشد.

Excel فرض می‌کند این تنها کارپوشه بارگذاری شده است. اگر چندین کارپوشه بارگذاری شده باشند و ماکرو روی هر کدام از آن‌ها اجرا شود، نتایج متفاوتی حاصل می‌شود و حتی اگر در آن کارپوشه، کاربرگ sheet2 نداشته باشیم منجر به بروز خطا نیز می‌شود.

اگر ماکرو در کارپوشه نادرستی اجرا شود می‌تواند خطر بازنویسی داده‌های معتبر در صفحه گسترده نیز وجود داشته باشد که نتایج وحشتناکی ممکن است به بار آورد. این امر به آن خاطر است که ماکرو به یک کارپوشه خاص ارجاع نمی‌کند و فرض می‌کند می‌خواهیم روی کارپوشه فعال اجرا شود.

لایه Workbook در سلسله مراتب نادیده گرفته شده است. نه کارپوشه‌ای تعریف شده و نه کاربرگی که ماکرو باید از آن آغاز شود.

نکته مهم این است که نباید فکر کنیم که یک ماکروی ضبط شده تحت هرشرایطی اجرا می‌شود. ممکن است متوجه شده باشید که باید تمام شرایط احتمالی را از قبل مد نظر داشته باشیم (مثل ارجاع به کارپوشه‌ای که باید استفاده شود).

فصل سیزدهم

مدل شئ Excel (اشیای اصلی)

در فصل قبل به بررسی چگونگی کار با مدل شئ Excel ، سلسله مراتب آن و انتقال پارامترها پرداختیم. در این فصل به صورت مشروح‌تر به بررسی مجموعه‌ها و اشیای اصلی که در کد از آن‌ها استفاده می‌کنیم تا با صفحه گسترده ارتباط برقرار کنیم، می‌پردازیم.

اشیایی که در فصل قبل از آن‌ها استفاده کردیم عبارتنداز: Application، Workbook، Windows، Worhsheet، Range.

شئ Application

شئ Application در بالاترین نقطه سلسله مراتب مدل شئ قرار دارد و نشان‌دهنده کل برنامه کاربردی وب است. این شئ حاوی مجموعه‌هایی از کارپوشه‌ها و کاربرگ‌ها است که صفحات گسترده را در Excel می‌سازند و اطلاعات سطح بالایی در مورد خود برنامه کاربردی در اختیار ما می‌گذارند، مانند سلول فعال (سلولی که در آن، مکان‌نما واقع است) و کاربرگ فعال (کاربرگی که روی آن کار می‌شود). این شئ هم چنین متدهایی برای توابع دارد (مانند محاسبه صفحه گسترده یا بازیابی نام فایل Open یا نام فایل Save As برنامه کاربردی Excel). هم‌چنین می‌توانید از برنامه Excel خارج شده و آن‌را کاملاً ببندید. توجه کنید که شئ Application هیچ گزینه‌ای برای چاپ ندارد چون تنها شئ Application است و یک کارپوشه یا کاربرگ نیست.

شئ Application، یک شئ پیش فرض است و در ساختار دستوری برای بعضی خصوصیات مشخص نمی‌شود. برای مثال دستور ActivePrinter ، جزییات چاپگر فعال را برمی‌گرداند.

خصوصیات، متدها و مجموعه‌های اصلی

این بخش به بررسی خصوصیات، متدها و برنامه‌های کاربردی اصلی که در شئ Application به کار گرفته می‌شوند، می‌پردازد.

ActiveCell

خصوصیت ActiveCell نشان‌دهنده سلول فعالی است که مکان‌نما در حال حاضر در صفحه گسترده Excel ما در آن قرار دارد. شما می‌توانید از آن برای به دست آوردن آدرس سلول فعال استفاده کنید:

```
MsgBox Application.ActiveCell.Address
```

کد فوق، آدرس سلول فعال را در قالب مطلق (مثلا $C$4) برمی‌گرداند. توجه کنید که این کد، تنها آدرس سلول را برمی‌گرداند و مقدار برگشتی آن شامل مشخصات کارپوشه و کاربرگ نیست.

ActivePrinter

این خصوصیت، نام چاپگر فعال و اتصالی را، که به کار می‌گیرد برمی‌گرداند. مثلاً LPT1 یا اگر از درگاه USB استفاده می‌کنید، EPUSB1 را برمی‌گرداند. این خصوصیت، اطلاعاتی مشابه با انتخاب گزینه File Print از منوی صفحه گسترده VBE یا Excel را برمی‌گرداند:

```
MsgBox Application.ActivePrinter
```

این کد، رشته 'EPSON Stylus CX3200 on EPUSB1:' نمایش داده می‌شود. اگر کد شما قرار است اطلاعاتی چاپ کند و شما می‌خواهید بدانید کاربر روی شبکه، آن اطلاعات را برای چاپ به کجا می‌فرستد، این خصوصیت می‌تواند مفید باشد.

ActiveSheet

این خصوصیت نشان‌دهنده کاربرگ فعالی است که در Excel نمایش داده می‌شود. یکی از کاربردهای ActiveSheet ، انتخاب یک سلول روی آن کاربرگ است:

```
Application.ActiveSheet.Cells (10,10).Select
```

این کد در کاربرگ فعال، مکان‌نما را به سلولی که ۱۰ ستون و ۱۰ ردیف پایین‌تر از سلول فعال است، می‌فرستد.

ActiveWindow

این خصوصیت، نشان‌دهنده پنجره فعال در Excel است. با انتخاب گزینه Window از منوی Excel، لیستی از تمام پنجره‌های باز به نمایش درمی‌آید که در کنار یکی از آن‌ها، تیک خورده و این علامت فعال بودن آن پنجره است. از این خصوصیت برای به‌دست‌آوردن عنوان پنجره فعال (متن موجود در نوار عنوان) استفاده می‌کنیم:


```
MsgBox Application.ActiveWindow.Caption
```

ActiveWorkbook

می‌توانید از این خصوصیت برای یافتن نام کارپوشه فعال در Excel استفاده کنید:

```
MsgBox Application.ActiveWorkbook.Name
```

این کد، "Book1" یا هر نامی را که برای کارپوشه فعلی انتخاب کرده‌اید را نمایش می‌دهد.
ممکن است ActiveWorkbook و ActiveWindow را با هم اشتباه بگیرید. بعضی افراد فرض می‌کنند که کارپوشه مثل یک پنجره تلقی می‌شود در صورتی که چنین نیست. یک کارپوشه شامل چندین پنجره است که هر کدام از آن‌ها کاربرگ نامیده می‌شود.

AddIns

این مجموعه، تمام الحاق‌هایی (Add_in) که در حال حاضر در Excel بارگذاری شده‌اند، نشان می‌دهد. شما می‌توانید نام آن‌ها را همراه با مسیری که از آن‌جا بار گذاری شده‌اند، با استفاده از زیر روال زیر فهرست کنید:

```
Sub test( )
Dim MyAddin As AddIn
For Each MyAddin In AddIns
        MsgBox MyAddin.FullName
Next
End Sub
```

اگر کد شئ وابسته به الحاق خاصی از Excel است، استفاده از این خصوصیت می‌تواند بسیار سودمند باشد. اگر آن الحاق بارگذاری نشده باشد، کد شما جواب نخواهد داد.

Assistant

این شئ، Office Assistant (دستیار Office) را نمایش می‌دهد که کاراکتر خاصی است که وقتی روی Help کلیک می‌کنیم ظاهر می‌شود. شما یا به این خصوصیت علاقه‌مند شده یا از آن متنفر می‌شوید. می‌توانید از شئ Assistant برای اختصاصی کردن دستیار Office به دلخواه خود استفاده کنید. برای دیدن چگونگی انجام این کار به فصل ۳۹ مراجعه کنید.

Calculate

این متد باعث می‌شود کل صفحه گسترده، مانند وقتی F9 را می‌زنیم مجدداً محاسبه شود:

```
Application.Calculate
```

Calculation

این خصوصیت، متد محاسبه به کار گرفته شده در برنامه Excel را تنظیم می‌کند. آن را می‌توان برای مقادیر xlCalculationAutomatic ، xlCalculationManual و xlCalculationSemiAutomatic تنظیم کرد:

```
Application.Calculation = xlCalculationManual
```

عملکرد این خصوصیت شبیه انتخاب Tools | Options از منوی Excel و سپس انتخاب زبانه Calculation و کلیک روی دکمه Manual Calculation است.

Caption

این خصوصیت، عنوان برنامه کاربردی Excel را که در نوار پنجره ویندوز مشخص شده است، در خود نگه می‌دارد. برای مثال:

```
MsgBox Application.Caption
```

عبارت "Microsoft Excel – Book1" را نمایش می‌دهد.

البته با کد زیر می‌توانیم عنوان (Caption) را تغییر دهیم:

```
Application.Caption = "MyApplication"
```

البته می‌توانیم با نوشتن یک رشته تهی، عنوان برنامه را حذف کنیم:

```
Application.Caption = " "
```

این کد تنها عنوان (نه کارپوشه فعلی) را تغییر می‌دهد.

Rows و Columns

این دو نشان‌دهنده ستون‌ها و ردیف‌های صفحه گسترده فعلی هستند؛ از آن‌ها می‌توانید برای انتخاب ستون‌ها یا ردیف‌های خاص استفاده کنید:

```
Application.Columns(3).Select
```

کد فوق، ستون C را انتخاب می‌کند و کد زیر نیز ردیف ۱۰ را انتخاب می‌نماید:

```
Application.Rows(10).Select
```


Dialogs

این مجموعه امکان دستیابی به مجموعه داخلی Dialogs را ممکن می‌سازد. برای یادگیری جزییات چگونگی استفاده از آن برای نمایش کادرهای محاوره‌ای Excel، به فصل ۱۰ مراجعه کنید.

Help

این متد، فایل Help (کمک) Excel را فرا می‌خواند. البته اگر به Microsoft Help Compiler دسترسی داشته باشید می‌توانید سیستم Help اختصاصی خودتان را بسازید:

```
Application.Help
```

MemoryTotal و MemoryUsed ، MemoryFree

این خصوصیات به شما می‌گویند که چه میزان از حافظه کامپیوتر شما آزاد است، چه میزان از آن استفاده شده است و در مجموع چه‌قدر حافظه دارید:

```
MsgBox Application. MemoryFree
```

این خصوصیت به ویژه زمانی مفید است که دارای یک صفحه گسترده بزرگ باشیم و نیازمند حافظه زیادی باشد. شما می‌توانید کاری کنید که صفحه گسترده‌تان در هنگام بارگذاری، حافظه آزاد موجود را کنترل کرده و در صورتی که از مقدار خاصی کمتر باشد، پیغام اخطار ارسال کند.

OperatingSystem

می‌توانیم از این خصوصیت برای کنترل و مشخص کردن سیستم عاملی که در حال حاضر در Excel روی آن اجرا می‌شود، استفاده کنیم:

```
MsgBox Application.OperatingSystem
```

OrganizationName

این خصوصیت، نام سازمانی را که در هنگام نصب ویندوز وارد شده است، برمی‌گرداند:

```
MsgBox Application.OrganizationName
```

یک راه جالب برای ایمن کردن برنامه شما و این‌که مطمئن شوید همکارانتان، کار شما را به فرد دیگری ارایه نخواهند کرد وجود دارد: می‌توانیم وقتی کاربرگ یا برنامه الحاقی شما بارگذاری می‌شود خصوصیت OrganizationName را کنترل کنیم و اگر مقدار آن با مقدار مورد نظر ما فرق می‌کرد،

بارگذاری برنامه را با استفاده از دستور Quit (که در ادامه بررسی خواهد شد) متوقف کنیم. البته این کار باعث محافظت صددرصد برنامه نمی‌شود اما حداقل کاربران ناشی را از سوء استفاده باز می‌دارد.

Quit

این متد، برنامه کاربردی Excel را کاملاً می‌بندد، دقیقاً انگار از منوی Excel، Exit | File را انتخاب کرده باشیم. البته قبل از بستن برنامه از شما خواسته می‌شود در صورت تمایل، تغییرات را در فایل ذخیره کنید:

```
Application.Quit
```

RecentFiles

این، مجموعه‌ای از فایل‌های اخیراً بارگذاری شده در Excel را نمایش می‌دهد:

```
Sub Recent_files ( )
For Each file In Application.RecentFiles
        MsgBox file.Name
Next
End Sub
```

این خصوصیت، فایل‌های اخیراً بارگذاری شده را نمایش می‌دهد، انگار از منوی Excel گزینه Files را انتخاب کنیم و لیست این فایل‌ها را در انتهای منوی آن ببینیم.

Selection

این خصوصیت، شئ گزینشی فعلی در برنامه کاربردی وب را در خود نگه می‌دارد. می‌توانیم آدرس سلول را از طریق خصوصیت Address مشخص کنیم:

```
MsgBox Application.Selection.Address
```

این کد، آدرس را در قالب مطلق (مثلاً $B$1)برمی‌گرداند، اما مشخص نمی‌کند این سلول روی کدام کاربرگ واقع است. شما می‌توانید با استفاده از خصوصیت Worksheet، کاربرگ مربوطه را هم مشخص کنید:

```
MsgBox Application.Selection.Worksheet.Name
```


Sheets

این مجموعه، تمام کاربرگ‌ها را در کتاب فعلی نمایش می‌دهد. این مجموعه شبیه مجموعه Worksheets است که در ادامه بررسی خواهیم کرد اما دارای خصوصیات و متدهای متفاوتی است. هم‌چنین یک شئ مستقل Sheet نیز وجود دارد که در مجموعه Worksheets به کار گرفته می‌شود. تفاوت مهم دیگر در این است که می‌توانید از Sheets برای چاپ یا مرور قبل از چاپ (PrintPreview) برگه‌های مستقل استفاده کنید اما شئ Worksheet هیچ متدی برای انجام این کار ندارد. ممکن است نیاز داشته باشید که تنها یک کاربرگ را چاپ کنید. برای انجام این‌کار از کد زیر استفاده می‌کنیم:

```
Application.Sheets("sheet1").Print
```

هم‌چنین می‌توانیم با استفاده از متد (PrintPreview)، نتیجه کار را از داخل کد خود مرورکنیم:

```
Application.Sheets("sheet1") .PrintPreview
```

ThisWorkbook

این خصوصیت نشان می‌دهد که کد ماکروی فعلی در کجا اجرا می‌شود:

```
Application.ThisWorkbook.Name
```

Undo

این متد، آخرین عملیات انجام شده در برنامه کاربردی Excel را لغو کرده و شبیه انتخاب Edit | Undo از منوی فرم عمل می‌کند:

```
Application.Undo
UserName
```

UserName

این خصوصیت، نام کاربری را که وارد سیستم ویندوز شده است، برمی‌گرداند:

```
MsgBox Application.UserName
```

ممکن است بخواهید برنامه شما، کنترل کند که آیا کاربری که به صفحه گسترده دسترسی پیدا می‌کند، همان است که مورد تأیید ماست یا خیر. می‌توانید این کار را با مقایسه لیستی از نام‌های معتبر انجام دهید:

```
If Application.UserName = "Richard Shepherd" Then
```

Version

این خصوصیت، شماره نسخه VBA را که از آن استفاده می‌کنید، برمی‌گرداند:

```
MsgBox Application.Version
```

اگر یک برنامه الحاقی نوشته باشیم ممکن است پس از شروع اجرای آن، بخواهیم آن را از لحاظ نسخه VBA مورد بررسی قرار دهیم تا در صورت استفاده کاربر از یک نسخه قدیمی‌تر، مشکلی پیش نیاید.

شئ Workbook

این شئ، کل کارپوشه بار گذاری شده در Excel را نشان می‌دهد. کارپوشه پیش‌فرض،Book1 است. این شئ، یک سطح پایین‌تر از شئ Application است. شئ Workbook، قسمتی از مجموعه Workbooks را نشان می‌دهد. این مجموعه تمام کارپوشه‌هایی را که در حال حاضر در Excel بارگذاری شده‌اند، نشان می‌دهد. نمی‌توانیم مستقیماً به شئ Workbook ارجاع کنیم بلکه باید از طریق مجموعه Workbooks به آن ارجاع نماییم. برای دیدن چگونگی انجام این کار به مثال بخش بعدی توجه کنید.

خصوصیات، متدها و مجموعه‌های اصلی

موارد زیر، خصوصیات، متدها و مجموعه‌های اصلی هستند که در شئ Workbook به‌کار گرفته می‌شوند.

Activate

این متد، کارپوشه مشخص شده در شئ Collection را فعال کرده و به صورت خودکار، به صفحه فعال در آن کارپوشه می‌رود. این‌کار شبیه انتخاب Window از منوی Excel و کلیک روی یک کارپوشه است. البته یک پنجره الزاماً با یک کارپوشه مساوی نیست. برای جزییات بیشتر به بخش "شئ Window" در ادامه همین فصل مراجعه کنید:

```
Workbooks("book1") .Activate
```

ActiveSheet

این خصوصیت به صفحه فعال در یک کارپوشه خاص اشاره می‌کند. ممکن است به یاد داشته باشید که شئ ActiveSheet در رابطه با شئ Application هم ذکر شد اما در آن مورد، این شئ با کاربرگ فعال

که در هر جای برنامه کاربردی قرار داشت، اشاره می‌کرد. اگر چندین کارپوشه بارگذاری شده باشند، این خصوصیت (در شئ Workbook) به آخرین برگی که فعال بوده است، اشاره می‌کند.

شئ Workbook به شما اجازه می‌دهد تا انتخاب خود را تنظیم کرده و مشخص کنید که در کدام کارپوشه می‌خواهید برگه فعال را مشخص نمایید:

```
MsgBox Workbooks("book1") .ActiveSheet.Name
```

کد فوق عبارت "Sheet1" را برمی‌گرداند البته اگر برگه فعال باشد.

Close

این متد، کاربرگ را درست مانند وقتی که File | Close را از منوی Excel انتخاب می‌کنیم، می‌بندد. پارامترهای اختیاری وجود دارند تا فایل ذخیره شده، نام متفاوتی به آن داده شود و مسیر کارپوشه مشخص شود:

```
Workbooks("book1") .Close  (True, "C:\Myfile.xls")
```

HasPassword

این خصوصیت مقدار True یا False را برمی‌گرداند که بستگی به این امر دارد که کارپوشه کلمه عبور دارد یا خیر:

```
MsgBox Workbooks("book1"). HasPassword
```

PrintOut

این متد، برگه فعال کارپوشه مورد نظر را به چاپگر فعال ارسال می‌کند:

```
Workbooks("book1") .PrintOut
```

PrintPreview

این متد، پیش‌نمایش چاپ برگه فعال کارپوشه مورد نظر را ارایه می‌کند:

```
Workbooks("book1") .PrintPreview
```

ReadOnly

این متد بسته به این‌که کارپوشه، فقط خواندنی باشد یا خیر، مقدار True یا False را برمی‌گرداند. اگر کد شئ، سلول‌های صفحه گسترده را تغییر می‌دهد این متد بسیار اهمیت خواهد داشت چون پس از آن لازم خواهد بود که کارپوشه با نام متفاوتی ذخیره شود:

```
MsgBox Workbooks("book1") .ReadOnly
```

Save و SaveAs

این متدها معمولاً قبل از بسته شدن کاربرگ، اطلاعات آن‌را ذخیره می‌کنند. شما می‌توانید تغییرات را با همان نام ذخیره کنید یا از متد SaveAs برای ذخیره کردن با نام متفاوتی استفاده نمایید. متد Save روی نسخه قبلی بازنویسی می‌کند اما متد SaveAs یک نسخه جدید ایجاد می‌نماید:

```
Workbooks("book1") .Save
Workbooks("book1") .SaveAs "C:\Myfile.xls"
```

Saved

این خصوصیت بسته به این‌که آیا تغییرات اعمال شده کاربر، ذخیره شده باشند یا خیر، مقدار True یا False برمی‌گرداند. اگر کارپوشه ذخیره شده باشد و هیچ تغییر جدیدی نیز اعمال نشده باشد، این خصوصیت مقدار True را نشان می‌دهد و اگر تغییری اعمال شده باشد مقدار False برمی‌گرداند:

```
MsgBox Workbooks("book1") .Saved
```

Sheets

Sheets دقیقاً مانند مجموعه Sheets که در بخش " شئ Application " توضیح دادیم، عمل می‌کند.

Windows

این مجموعه تمام پنجره‌های موجود در شئ Workbook را نشان می‌دهد. بخش "شئ Windows " را در ادامه همین بخش مطالعه کنید.

Worksheets

این مجموعه تمام کاربرگ‌های موجود در شئ Workbook را نشان می‌دهد. بخش " شئ Worksheet " را در ادامه همین بخش مطالعه کنید.

شئ Windows

این شئ تمام پنجره‌ها را در برنامه Excel نشان می‌دهد. در بسیاری از موارد، این شئ با مجموعه Workbook اشتباه گرفته می‌شود اما آن‌ها همیشه تشابهی با هم ندارند. با انتخاب Window | New Window از منوی Excel می‌توانیم پنجره جدیدی باز کنیم. این کار باعث باز شدن پنجره‌ای می‌شود که یک کپی از کارپوشه فعلی در آن قرار گرفته است. اگر مجدداً از منوی Excel، Window را انتخاب کنیم، خواهیم دید در انتهای نوار منو، دو پنجره وجود دارد:

Book1 و Book2، اما هر دوی آن‌ها بر مبنای یک کارپوشه ایجاد شده‌اند.

مجموعه Windows نشان می‌دهد که وقتی گزینه Window را از منوی Excel انتخاب می‌کنیم، چه چیزی دیده می‌شود و بسیاری از متدهای این شئ مرتبط با گزینه‌های انتخاب در منوی Window هستند (مانند Freeze Panes یا Split).

خصوصیات، متدها و مجموعه‌های اصلی

موارد زیر خصوصیات، متدها و مجموعه‌های اصلی هستند که در شئ Windows به‌کار گرفته می‌شوند:

Activate، ActivateNext و ActivatePrevious.

این متدها به شما اجازه می‌دهند تا با مشخص کردن پنجره‌ای در مجموعه Windows آن را از طریق کد فعال کنید. نام یا شماره پنجره را به عنوان ایندکس استفاده کنید و پس از متد Activate به‌کار برید:

```
Windows("Book1").Activate
```

هم‌چنین می‌توانید با استفاده از شماره ایندکس یک پنجره نیز به آن ارجاع کنید:

```
Windows(1) .Activate
```

می‌توانید از ActivateNext و ActivatePrevious برای انتقال پنجره‌ها به نسبت پنجره فعال نیز استفاده کنید:

```
ActiveWindow.ActivateNext
ActiveWindow.ActivatePrevious
```

ActiveCell

این خصوصیت، جزییات سلول فعال را در یک پنجره خاص برمی‌گرداند. سلول فعال، سلولی است که مکان‌نما در آن قرار دارد. مثال زیر نشان می‌دهد که آدرس سلول فعال را چگونه به‌دست آوریم:

```
MsgBox Windows("Book1").ActiveCell.Address
```

ActivePane

این خصوصیت تنها منحصر به مجموعه Windows است، چون با توجه به خود پنجره عمل می‌کند و ربطی به کاربرگ ندارد. این خصوصیت به کاربر اجازه می‌دهد از خود بخش (Pane) اطلاعات کسب کند، مانند منطقه مرئی(یعنی آدرس سلول‌هایی که کاربر می‌تواند روی صفحه ببیند).

از منوی Split،Excel | Windows را انتخاب کنید به طوری که پنجره کاربرگ شما به چهار بخش

تقسیم شود. روی یک بخش، سلول فعال قرار خواهد داشت که آن بخش نیز بخش فعال محسوب می‌شود. با استفاده از کد زیر می‌توانیم منطقه مرئی این بخش را مشخص کنیم:

```
MsgBox Windows(1).ActivePane.VisibleRange.Address
```

در این کد، فرض بر این است که ما در پنجره شماره ۱ هستیم. این کد، دامنه آدرس سلول‌های مرئی در کاربرگ خواهد بود (مانند"$L$7:"$C$1).

ActiveSheet

می‌توانیم از این خصوصیت برای یافتن نام کاربرگی که در یک پنجره خاص فعال است، استفاده کنیم:

```
MsgBox Windows(1).ActiveSheet.Name
```

Sheet1 یا هر نام دیگری که برای کاربرگ انتخاب کرده‌ایم خروجی این کد، است.

Caption

این خصوصیت، عنوان پنجره را تغییر می‌دهد:

```
ActiveWindow.Caption = "MyWindow"
```

نکته جالب این است که اگر کد فوق را مساوی یک رشته تهی قرار دهیم، عنوان پنجره کاملاً حذف خواهد شد. اگر لازم باشد تا دوباره عنوان را به تنظیمات اصلی برگردانیم، باید عنوان اصلی را در یک متغیر ذخیره کنیم:

```
ActiveWindow.Caption = " "
```

Close

این متد، پنجره را می‌بندد. درست مانند وقتی که روی نماد× در کوشه بالا- سمت راست پنجره کلیک می‌کنیم. می‌توانیم پارامترهای اختیاری برای SaveChanges، FileName و RouteWorkBook را نیز اضافه کنیم.

Display Properties

شئ Windows مجموعه متنوعی از گزینه‌های انتخابی برای نمایش دارد که اجازه تنظیمات زیر را به ما می‌دهد:

```
DisplayFormulas
DisplayGridlines
DisplayHeadings
```



```
DisplayHorizontalScrollBar
DisplayOutline
DisplayRightToleft
DisplayVerticalScrollBar
DisplayWorkBookTabs
DisplayZeros
```

این خصوصیات همگی از نوع بولین هستند و این به معنای آن است که آن‌ها مقدار True یا False را در خود دارند. وقتی آن‌ها را از منوی Tools | Options،Excel انتخاب می‌کنیم، تنظیمات را نشان می‌دهند. روی زبانه View کلیک کنید تا کادرهای انتخاب تمام آن‌ها را ببینید.

می‌توانیم با تغییر این خصوصیت، نمایش را به صورت دلخواه درآوریم. برای مثال:

```
ActiveWindow.DisplayWorkbookTabs = False
```

باعث حذف زبانه (tab) از انتهای پنجره فعال می‌شود.

FreezePanes

این خصوصیت مانند قرار دادن مکان‌نما روی یک سلول کاربرگ و سپس انتخاب Window | Freeze Panes از منوی Excel عمل می‌کند. این خصوصیت، یک مقدار بولین را در خود نگه می‌دارد. بخش‌ها(Pane) در محل فعلی مکان‌نما به صورت ثابت (Freeze) در می‌آیند:

```
ActiveWindow.FreezePanes = True
```

GridLineColor

این خصوصیت، رنگ خطوط شبکه در نمایش پنجره را تغییر می‌دهد:

```
ActiveWindow.GridLineColor = QBColor(14)
```

می‌توانیم از رنگ‌های RGB (قرمز، سبز، آبی) هم استفاده کنیم.

NewWindow

این خصوصیت برمبنای پنجره فعال، یک پنجره جدید درست می‌کند، انگار که از منوی Excel، Window | New Window را انتخاب کرده‌ایم:

```
ActiveWindow.NewWindow
```

Panes

این خصوصیت، مجموعه‌ای از تمام بخش‌ها در پنجره است که می‌توانیم برای یافتن این مورد که چند بخش در پنجره خاص وجود دارد، از آن استفاده کنیم:

```
MsgBox ActiveWindow.Panes.Count
```

شئ Pane در مجموعه Panes به ما اجازه می‌دهد تا به خصوصیات و متدهای بیشتری دست پیدا کنیم.

RangeSelection

این خصوصیت بسیار مفید، به ما می‌گوید که کاربر چه دامنه‌ای را انتخاب کرده است:

```
MsgBox ActiveWindow.RangeSelection.Address
```

کد فوق، یک سلول (مانند $C$10) یا مجموعه‌ای از سلول‌ها (مانند "$B$10:$E$12") را که کاربر انتخاب کرده است، نمایش می‌دهد.

SelectedSheets

این مجموعه، مجموعه مفیدی است که مشخص می‌کند کاربر، چه انتخابی انجام داده است. ممکن است در خصوصیت RangeSelection متوجه شده باشید که اگرچه اطلاعات برگشتی در مشخص شدن سلول‌های انتخاب شده توسط کاربر بسیار سودمند هستند اما مشخص نمی‌شود این سلول‌ها روی کدام کاربرگ انتخاب شده‌اند.

اگر بخواهیم کد حرفه‌ای برای Excel بنویسیم ممکن است در نظر بگیریم که کاربر می‌تواند سلول‌های مشابه را از چندین کاربرگ انتخاب کند و این‌کار محدود به یک کاربرگ نیست. البته این کاربرگ‌ها ممکن است همگن نباشند. برای مثال کاربر ممکن است Sheet1، Sheet4 و Sheet5 را انتخاب کند.

این متد برای برنامه‌های الحاقی، یعنی جایی که کاربر می‌تواند در بین چندین کاربرگ و حتی چندین کارپوشه عمل کند، بسیار سودمند است.

با چرخیدن در مجموعه SelectedSheets می‌توانیم بفهمیم کدام برگه‌ها انتخاب شده‌اند:

```
Dim MySheet As Wroksheet
For Each MySheet In ActiveWindow.SelectedSheets
        MsgBox MySheet.Name
Next MySheet
```

این کد، تمام برگه‌های انتخاب شده را به نوبت نمایش می‌دهد. برای مشخص کردن تمام سلول‌هایی که انتخاب شده‌اند، این کد را به RangeSelection الحاق کنید.

در فصل‌های ۲۰ تا ۴۱ این مورد در چند مثال عملی به‌کار گرفته خواهد شد.

Split

این خصوصیت، پنجره فعلی را به چند بخش تقسیم می‌کند یا آن‌ها را مجدداً به یک بخش تبدیل خواهد کرد. تقسیم شدن پنجره در محل فعلی مکان‌نما انجام می‌شود:

```
ActiveWindow.Split = True
```

این کد مانند انتخاب Window | Split از منوی Excel عمل می‌کند.

TabRadio

این خصوصیت، اندازه منطقه نمایش زبانه(tab) را تنظیم می‌کند. مقدار آن از 0 تا 1 است و نشان می‌دهد چه مقدار از پایین صفحه نمایش برای زبانه‌های کاربرگ و چه مقدار برای نوار پیمایشی افقی خواهد بود. مقدار صفر برای این خصوصیت به معنای آن است که هیچ زبانه‌ای نشان داده نشود و تنها نوار پیمایشی افقی نشان داده شود. مقدار 1 نشان می‌دهد که زبانه‌ها وجود دارند اما نوار پیمایش افقی وجود نخواهد داشت. مقدار 0.5 به معنای آن است که 50 درصد به زبانه‌ها و 50 درصد به نوار پیمایش افقی اختصاص می‌یابد.

```
ActiveWindow. TabRatio= 0.5
```

WindowState

این خصوصیت به شما اجازه می‌دهد تا وضعیت یک پنجره را مشخص کنید یا این‌که آن را در یکی از حالات زیر تنظیم کنید:

جدول ۱-۱۳

حالت پنجره	مقدار خصوصیت
پنجره maximized	xlMaximized
پنجره minimized	xlMinimized
پنجره normal	xlNormal

```
ActiveWindow.WindowState = xlMinimized
```

این کد، ActiveWindow را به حالت کوچک شده (Minimized) درمی‌آورد، درست مانند وقتی که روی دکمه Minimized در گوشه بالا- سمت راست پنجره کلیک کنیم. هم‌چنین می‌توانیم از آن برای کنترل حالت پنجره نیز استفاده کنیم.

Zoom

این خصوصیت، خصوصیت Zoom پنجره را تنظیم می‌کند، درست مانند وقتی که View | Zoom را از منوی Excel انتخاب می‌کنیم.

```
ActiveWindow.Zoom = 80
```

این کد باعث می‌شود بزرگ‌نمایی (Zoom) به میزان 80 درصد به دست آید.

شئ Worksheet

این شئ، کاربرگ واقعی را که روی آن کار می‌کنیم، نشان می‌دهد. در سلسله مراتب مدل شئ Excel، در زیر شئ Workbook قرار می‌گیرد چون Worksheets جزیی از Workbook خواهد بود.

خصوصیات، متدها و مجموعه‌های اصلی

موارد زیر، عبارتند از خصوصیات، متدها و مجموعه‌هایی که در شئ Worksheet قابل استفاده هستند.

Calculate

این متد با فرض این‌که متد محاسبه به صورت manual (دستی) تنظیم شده است، یک کاربرگ خاص را محاسبه می‌کند.

```
Workheets(1).Calculate
```

در صورتی که کاربرگ ما دارای محاسبات بسیار پیچیده باشد و تنها بخواهیم یک برگه خاص محاسبه شود، این متد بسیار مفید خواهد بود.

CheckSpelling

این متد، تلفظ در یک کاربرگ را کنترل می‌کند، درست مانند وقتی که Tools | Spelling را از منوی Excel انتخاب می‌کنیم:

```
Worksheets("Sheet1") .CheckSpelling
```


می‌توانید پارامترهای اختیاری برای Custom Dictionary، Ignore Uppercase و Always Suggest نیز اضافه کنیم.

Comments

مجموعه‌ای از تمام توضیحات (Comments) اضافه شده به کاربرگ انتخاب شده را در برمی‌گیرد. Comments، متن اضافی است که به یک سلول چسبانده می‌شود. پس از اضافه کردن توضیحات به یک سلول، یک مثلث قرمز رنگ در گوشه بالا- سمت راست سلول ظاهر می‌شود و هنگامی که ماوس را روی آن ببریم، توضیحات مذکور ظاهر می‌شوند. با راست کلیک کردن روی سلول و انتخاب گزینه مناسب از منوی باز شو می‌توانیم توضیحی در سلول درج کرده و یا حذف و ویرایش کنیم.
می‌توانیم از کد زیر برای مشخص کردن این‌که چه تعداد توضیح در یک کاربرگ وجود دارد استفاده کنیم:

```
MsgBox Worksheets("sheet2") .Comments.Count
```

Delete

این متد، یک کاربرگ را حذف می‌کند، درست مانند این است که Edit | Delete Sheet را از منوی Excel انتخاب کرده باشیم:

```
Worksheets("sheet1").Delete
```

PrintOut و PrintPreview

این متدها به شما اجازه می‌دهند تا یک کاربرگ خاص را چاپ کنید یا پیش نمایش چاپ آن را ببینید:

```
Worksheets("sheet2") .PrintOut
Worksheets("sheet2") .PrintPreview
```

Protect

این متد شما را قادر می‌سازد تا از یک کاربرگ محافظت کنید، دقیقاً مانند وقتی که از منوی Excel، Tools | protection | Protect Sheet را انتخاب می‌کنید. می‌توانید کلمه عبور را نیز به صورت یک پارامتر اضافی به کد آن الحاق کنید:

```
Worksheets("sheet2") .Protect
Worksheets("sheet2") .Protect ("apple")
```

در مثال اول، هیچ کلمه عبوری اضافه نشده است پس برای حذف محافظت، از شما تقاضای وارد کردن کلمه عبور نخواهد شد. در مثال دوم، کلمه عبور "apple" است.

Range

این شئ، یک شئ بسیار مهم در کاربرگ است و در ادامه این فصل در بخش "شئ Range" به تفصیل شرح داده خواهد شد.

SaveAs

این متد، کارپوشه را با یک نام جدید ذخیره می‌کند. هرچند بخشی از شئ Worksheet محسوب می‌شود اما کل کارپوشه را ذخیره می‌کند:

```
Worksheets("sheet2").SaveAs  ("MyFile")
```

Select

این متد یک Worksheet را از یک Workbook انتخاب می‌کند مانند وقتی که روی دکمه‌های زبانه در پایین پنجره کلیک می‌کنیم.

```
Worksheets("sheet2").Select
```

این کد، Sheet2 را انتخاب می‌کند.

SetBackGroundPicture

این متد، یک تصویر مانند یک BMP را در پس زمینه صفحه گسترده قرار می‌دهد:

```
Worksheets("sheet2").SetBackgroundPicture ("C: \Mypic.bmp")
```

Unprotect

این متد به شما اجازه می‌دهد تا یک برگه را از حالت حفاظت خارج کرده و با متد Protect که قبلاً بررسی شد، مشارکت می‌کند.

```
Worksheets("sheet2"),Unprotect ("apple")
```

می‌توانید یک کلمه عبور نیز اضافه کنید که البته اختیاری است. اگر کلمه عبور مورد نیاز باشد اما ارایه نشود، کادر محاوره‌ای کلمه عبور ظاهر خواهد شد.

اگر یک برگه حفاظت شده داشته باشیم که با استفاده از کدنویسی بخواهیم تغییراتی در آن اعمال کنیم، متدهای Unprotect و Protect می‌توانند مفید واقع شوند. در واقع می‌توانیم با استفاده از کلمه عبور، یک برگه را از حالت حفاظت شده خارج کنیم، تغییرات را روی آن اعمال کنیم و سپس مجدداً

آن‌را به حالت حفاظت شده درآوریم.

Visible

تنظیم این خصوصیت به True یا False مشخص می‌کند که آیا کاربرگ مشاهده شود یا خیر. این خصوصیت مانند پنهان کردن کاربرگ با استفاده از Format | Sheet | Hide از منوی Excel عمل می‌کند. برای مثال:

```
Worksheets("sheet2").Visible = False
```

این کد، Sheet2 را پنهان می‌کند. برای مربی کردن مجدد برگه، خصوصیت Visible را True تنظیم می‌کنیم.

شئ Range

این شئ با مجموعه‌ای از سلول‌های تکی سروکار دارد و تغییراتی را در آن‌ها اجرا می‌کند. بدون این شئ نمی‌توانیم چندین سلول را به صورت یکباره روی کاربرگ تغییر دهیم.

خصوصیات، متدها و مجموعه‌های اصلی

موارد زیر خصوصیات، متدها و مجموعه‌های اصلی هستند که در شئ Range به‌کار گرفته می‌شوند:

Activate

این متد، یک سلول خاص یا مجموعه‌ای از سلول‌ها را فعال می‌کند تا آن‌ها را در یک سلول فعال یا چند سلول فعال وارد نماید. برای اجرای کد زیر، Sheet1 باید کاربرگ فعال باشد:

```
Worksheets("sheet1").Range("a1").Activate
```

این کد، تنها یک سلول را فعال می‌کند حتی اگر یک مجموعه سلول مانند a1.b1 را وارد کنیم.

AddComment

این متد به ما اجازه می‌دهد توضیحی را به سلول تعریف شده در مجموعه اضافه کنیم:

```
Worksheets("sheet1").Range("a1").AddComment ("MyComment")
```

اگر بخواهیم توضیحی را به سلولی که از قبل توضیحی دارد اضافه کنیم، پیغام خطایی صادر خواهد شد. اگر در عوض ارجاع به یک سلول، یک مجموعه سلول را وارد کنیم باز هم پیغام خطا صادر

می‌شود. برای ویرایش توضیح، لازم است در مجموعه Comments به توضیح ارجاع کنیم. برای یادگیری چگونگی انجام این کار به فصل ۲۳ رجوع کنید.

Address

این خصوصیت بسیار مهم، آدرس یک مجموعه سلول را ارایه می‌کند- برای مثال، مجموعه سلول‌هایی که کاربر انتخاب کرده است - می‌توانیم از این خصوصیت برای مثال‌های قبلی استفاده کنیم:

```
MsgBox Worksheets("sheet2") .Range("a3") .Address
```

این کد، $A$3 را برمی‌گرداند.

BorderAround

این متد، لبه‌ای دور گروه سلول‌ها یا یک سلول ترسیم می‌کند:

```
Worksheets("sheet2") .Range("a3.b10") .BorderAround (1)
```

این کد، یک لبه تک خطی دور مجموعه سلول a3.b10 روی Sheet2 کارپوشه ترسیم می‌کند. با تغییر شماره پارامترها می‌توانیم دیگر انواع لبه را نیز ترسیم کنیم.

Calculate

این متد با فرض این که خصوصیت Autocalculation (محاسبه خودکار) در حالت On تنظیم نشده باشد، مجموعه سلول‌های خاص مشخص شده را محاسبه می‌کند.

```
Worksheets("sheet2") .Range("a3.d12") .Calculate
```

در فصل ۲۳ خواهیم دید که این متد به عنوان بخشی از یک برنامه کاربردی به کار می‌رود.

Cells

این متد، مجموعه‌ای از سلول‌ها در دامنه سلول‌های مشخص شده است. برای مثال می‌توانیم بفهمیم چه تعداد سلول در مجموعه انتخابی وجود دارند.

```
MsgBox Worksheets("sheet2") .Range("a3.d12") .Cells.Count
```

این کد، عدد 40 را نمایش می‌دهد.

CheckSpelling

این متد، تلفظ در یک مجموعه سلول مستقل را کنترل می‌کند:


```
Worksheets("sheet2") .Range("a3.d12") .CheckSpelling
```

می‌توانیم پارامترهای اختیاری برای Custom Dictionary ، Ignore Uppercase و Always Suggest و ... نیز اضافه کنیم.

Clear

این متد، محتویات موجود در دامنه سلول‌های مشخص شده را پاک می‌کند. به یاد داشته باشید که این متد، همه چیز حتی توضیحات و قالب‌بندی‌ها (format) را نیز پاک می‌کند:

```
Worksheets("sheet2") .Range("a3.d12") .Clear
```

ClearComment

این متد، فقط توضیحات را از مجموعه سلول‌های مشخص شده حذف می‌کند:

```
Worksheets("sheet2") . Range("a3.d12") .ClearComments
```

ClearContents

این متد، فقط محتویات سلول یا سلول‌های انتخاب شده را پاک می‌کند - یعنی فقط داده‌هایی را که در سلول‌ها تایپ شده‌اند، پاک می‌کند و کاری به لبه‌ها و یا قالب‌بندی سلول‌ها ندارد:

```
Worksheets("sheet2") .Range("a3.d12") .ClearContents
```

ClearFormats

این متد، قالب‌بندی مجموعه‌ای از سلول‌ها را پاک می‌کند:

```
Worksheets("sheet2") .Range("a3.d12") .ClearFormats
```

Column و Row

این خصوصیت‌ها، شماره ستون یا ردیف نخست ستون را در مجموعه سلول‌های انتخاب شده برمی‌گردانند:

```
MsgBox Worksheets("sheet2") .Range("b3.d12") .Column
```

این کد، عدد 2 را برمی‌گرداند چون B (که ستون نخست مجموعه است)، ستون 2 است.

```
MsgBox Worksheets("sheet2") .Range("b3.d12") .Row
```

این کد، عدد 3 را برمی‌گرداند چون عنصر نخست مجموعه، B3 است که ردیف 3 می‌باشد.

Rows و Columns

این مجموعه‌ها مثل خصوصیات Column و Row که قبلاً بررسی کردیم عمل می‌کنند. اما شماره واقعی ستون‌ها و ردیف‌ها را در دامنه مشخص شده برمی‌گردانند. اگر بخواهیم از حلقه For..Next برای کار با هر سلول در دامنه انتخاب شده استفاده کنیم، این کد بسیار مفید است:

```
MsgBox Worksheets("sheet2") .Range("b3.d12") .Columns .Count
```

این کد، عدد 3 را نمایش می‌دهد که تعداد ستون‌ها در مجموعه سلول‌ها را نشان می‌دهد:

```
MsgBox Worksheets("sheet2") .Range("b3.d12") .Rows .Count
```

این کد، عدد 10 را نمایش می‌دهد که تعداد ردیف‌ها در مجموعه انتخاب شده است.

RowWidth و ColumnWidth

این خصوصیات، پهنای ستون‌ها یا ارتفاع ردیف‌ها در دامنه سلول‌های مشخص شده را برمی‌گردانند یا این‌که مقدار جدیدی برای آن‌ها تنظیم می‌کند:

```
Worksheets("sheet2") .Range("b3.d12") .ColumnWidth = 4
Worksheets("sheet2") .Range("b3.d12") .RowHeight = 10
```

Copy و PasteSpecial

این متدهای سودمند، مجموعه‌ای از سلول‌ها را کپی (Copy) می‌کنند و می‌چسبانند (Paste) در فصل21 خواهید دید که این متدها چطور کار می‌کنند:

```
Worksheets("sheet2") .Range("f19.g20") .Copy
Worksheets("sheet2") .Range("h19") .PasteSpecial
```

این کد، سلول‌ها را در دامنه F19.G20 روی Sheet2 انتخاب می‌کند و در دامنه H19 بر روی Sheet2 می‌چسباند.

PasteSpecial، اجازه استفاده از پارامترهایی را می‌دهد که مشخص می‌کند که آیا می‌خواهیم فقط مقادیر یا فقط قالب‌بندی‌ها را بچسبانیم:

```
Worksheets("sheet2") .Range("h19") .PasteSpecial Type:=xlPasteValues
```


PrintOut و PrintPreview

وقتی این متدها را با شئ Range به کار ببریم، اجازه چاپ کردن و دیدن پیش نمایش چاپ مجموعه سلول‌های انتخاب شده را می‌دهد.

```
Worksheets("sheet2") .Range("f19.g20") .PrintPreview
Worksheets("sheet2") .Range("f19.g20") .PrintOut
```

Replace

این متد، یک متد مفید است که یک کاراکتر خاص را که در مجموعه سلول‌های مشخص شده پیدا می‌شود با کاراکتر دیگری عوض می‌کند:

```
Worksheets("sheet2") .Range("f19.g20") .Replace "a", "b"
```

در کد فوق، تمام کاراکترهای a با b عوض می‌شوند.

Select

این متد مهم به ما اجازه می‌دهد تا مجموعه‌ای از سلول‌ها را انتخاب کنیم:

```
Worksheets("sheet2") .Range("f19.g20") .Select
```

کد فوق، دامنه F19 تا G20 را انتخاب می‌کند (درست مانند وقتی که با ماوس روی آن‌ها کلیک کنیم).

Text

این خصوصیت، متن موجود در دامنه سلول را برمی‌گرداند. اگر دامنه فقط شامل یک سلول باشد، متن آن برمی‌گردد و در غیر این‌صورت، یک پیغام خطا صادر می‌شود. نام Text برای این خصوصیت زیاد دقیق نیست چون می‌تواند یک عدد را هم به عنوان یک رشته متنی برگرداند. نمی‌توانیم از این خصوصیت برای قرار دادن یک مقدار در سلول استفاده کنیم و برای این کار باید از خصوصیت Value استفاده کنیم:

```
MsgBox Worksheets("sheet2") .Range("f26") .Text
```

Value

این خصوصیت مشابه با Text است اما می‌تواند عملیات خواندن/ نوشتن را انجام دهد پس می‌توانیم با کمک آن، داده‌هایی را در یک صفحه گسترده بنویسیم:

```
MsgBox Worksheets("sheet2") .Range("f26") .Value
Worksheets("sheet2") .Range("f26") .Value = 10
```

در هنگام خواندن اطلاعات فقط می‌توانیم از یک سلول استفاده کنیم و در غیر این‌صورت، پیغام خطای Type Mismatch صادر خواهد شد. اما در هنگام نوشتن نمی‌توانیم یک مقدار را در چندین سلول بنویسیم:

```
Worksheets("sheet2") .Range("f26.g40") .Value = 10
```

فصل چهاردهم

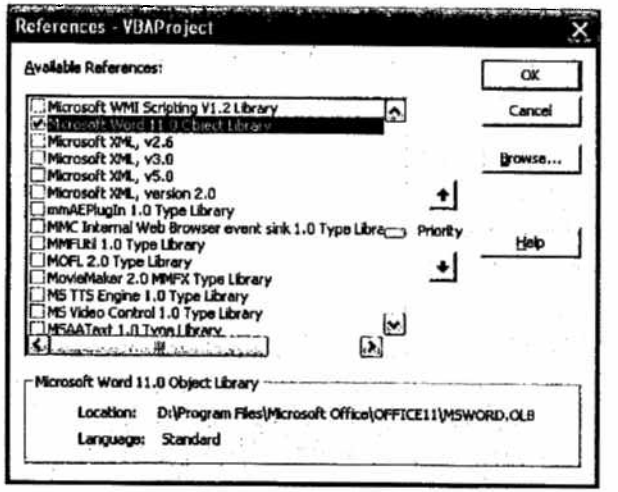
استفاده از Excel برای برقراری ارتباط با دیگر برنامه‌های Office

تمام برنامه‌های کاربردی Office مایکروسافت از VBA به عنوان زبان ماکرو نویسی اصلی خود استفاده می‌کنند و مانند Excel، همه آن‌ها نیز دارای مدل شئ خاص خود هستند. به این دلیل VBA مزیت بیشتری نسبت به دیگر زبان‌های برنامه‌نویسی غیر مایکروسافت دارد چون می‌تواند برای راه اندازی دیگر برنامه‌های Office به کار گرفته شود. برای مثال می‌توانیم حتی بدون آن‌که برنامه Word روی صفحه نمایش ظاهر شود از داخل Excel یک سند Word ایجاد کنیم. این امر ممکن است خیلی بعید به نظر رسد اما در واقع انجام این‌کار بسیار ساده است. برای مثال ممکن است کدی طراحی کنیم تا داده‌های موجود روی یک صفحه گسترده را دستکاری کند. ممکن است لازم باشد کاربر خروجی را در یک سند Word قرار دهد یا این‌که خروجی را به عنوان بخشی از یک سند قرار دهد. Excel، این امکان را می‌دهد تا یک سند خارجی Word را باز کنید، داده‌هایتان را در آن وارد کرده و سپس فایل را ذخیره کرده و ببندید، بدون این‌که لازم باشد حتی شناختی درباره چگونگی عملکرد ساختار فایل در Word داشته باشید.

این‌کار را می‌توان با متد CreateObject در VBA انجام داد. برای استفاده از CreateObject باید ابتدا در برنامه خود، ارجاع مناسبی به فایل Microsoft Office Object Library اضافه کنیم (در این مورد نام فایل مذکور، Word Object Library است). اگر قبلاً Office را نصب کرده باشیم این فایل در دسترس خواهد بود و به طور خودکار در لیست References قرار می‌گیرد و اگر Office را کامل نصب نکرده باشیم، آن‌گاه این فایل کتابخانه در دسترس نخواهد بود مگر آن‌که قبلاً به عنوان بخشی از یک برنامه کاربردی دیگر نصب شده باشد.

وقتی ارجاعی به یک Object Library اضافه می‌کنیم به ما اجازه می‌دهد تا اشیایی را برای آن برنامه کاربردی ایجاد کرده و از مدل شئ آن برنامه نیز استفاده کنیم (انگار که در حال برنامه‌نویسی در VBA در داخل آن برنامه کاربردی هستیم).

با انتخاب References | Tools از منوها می‌توانیم یک ارجاع اضافه کنیم. آن‌گاه تمام فایل‌های ارجاع موجود، در یک کادر محاوره‌ای ظاهر خواهند شد. لازم است که Microsoft Word Object Library را انتخاب کرده و کادر انتخابی آن‌را تیک بزنیم (تصویر ۱-۱۴).



تصویر ۱-۱۴ انتخاب Microsoft Word Object Library

توجه کنید که محل نشان داده شده در انتهای پنجره References به یک فایل OLB اشاره می‌کند که در اصل، Object Library برای Word است. قسمت Location به فایل OLB فهرستی اشاره می‌کند که مایکروسافت Office در آن‌جا نصب شده است. روی OK کلیک کنید تا آماده استفاده از Word در VBA باشید.

در این‌جا، یک نمونه کد برای ایجاد یک سند جدید Word و ذخیره کردن آن در درایو سخت ذکر می‌شود:

```
Sub Test_Word()
Dim oWd As Word.Application, oWdoc As Word.Document
Set oWd = CreateObject("Word.Application")
Set oWdoc = oWd.Documents.Add
oWdoc.Sections(1) .Range.Text = "My new Word Document"
oWdoc.SaveAs ("c:\MyTest.doc")
oWdoc.Close
```



```
oWd.Quit
Set oWdoc = Nothing
Set oWd = Nothing
End Sub
```

خط نخست، متغیرهایی را برای برنامه کاربردی و اشیا سند اعلان می‌کند. چون ارجاعی به Object Library منضم شده است اشیاء، متدها و ویژگی‌های Word را خواهید دید که به موازات تایپ کد، در یک لیست ظاهر می‌شوند. اگر ارجاعی به Object Library نداشته باشیم، VBA، این نوع اشیا را نخواهد شناخت و برنامه کاربردی به مشکل برمی‌خورد.

سپس از متغیر برنامه کاربردی (oWd) برای نگه داشتن شئ ایجاد شده برای برنامه کاربردی Word استفاده می‌کنیم. توجه کنید که در پارامتر CreateObject شرح، داخل علامت نقل قول قرار می‌گیرد. این شرح از نام کلاس برای شئ استفاده می‌کند و به طور خودکار ظاهر نخواهد شد چون به صورت یک رشته وارد شده است. برای این شرح، لیستی از انتخاب‌ها ارایه نخواهد شد. رشته "Word.Application"، نام کلاس برای شئ برنامه کاربردی در Word است.

حال متغیر oWdoc طوری تنظیم شده است که بر مبنای متغیر برنامه کاربردی (oWd)، یک سند جدید را نگه دارد. این امر دقیقاً شبیه آن است که سند Word بارگذاری شده و سپس از منو، File | New انتخاب شده باشد.

سپس متن "My new Word Document" به بخش نخست سند اضافه می‌شود. این متن در قسمت نخست ظاهر می‌شود چون ما سند Word را بارگذاری کرده‌ایم، در واقع به همان شیوه‌ای که انگار برنامه Word را باز کرده و سپس فایلی را در آن بارگذاری نموده‌ایم. وقتی این مورد رخ می‌دهد، مکان‌نما به طور خودکار به بالای سند می‌رود و دقیقاً همین مورد در کد ما نیز رخ خواهد داد چون ما سندی را که باز می‌شود، مورد تقلید قرار می‌دهیم.

سند در درایو سخت تحت عنوان C:\MyTest.doc ذخیره شده و سپس سند Word بسته می‌شود. باز گذاشتن این سند باعث بروز مشکلاتی به خصوص در هنگام خروج از Excel می‌شود. چون این یک برنامه کاربردی مجازی است پس اگر سند به درستی بسته نشود، در حافظه باقی می‌ماند (حتی اگر از Excel خارج شویم) هر چند Excel تنها یک ارجاع به آن دارد. البته این مورد خاص Word روی نوار وظیفه ویندوز ظاهر نمی‌شود، چون در کد به صورت مجازی ایجاد شده است. تنها روش کشف حضور آن، نگاه کردن به Task Manager است.

اگر شئ مجازی را به درستی نبندیم، وقتی ویندوز خاموش می‌شود، برنامه کاربردی Word از ما می‌پرسد که آیا لازم است سند ذخیره شود یا خیر، چون هنوز فرض می‌کند یک سند باز وجود دارد.

این امر ممکن است باعث گیجی کاربر شود چون او انتظار ندارد که یک برنامه Word باز باشد. برای پرهیز از چنین مواردی، مقدار متغیرهای oWd و oWdoc را برابر Nothing قرار دادیم و بدین وسیله، تمام حافظه اشغال شده توسط آن‌ها آزاد می‌شود.

اگر فایل تازه ایجاد شده را در Word بارگذاری کنیم خواهیم دید که مانند یک سند نرمال Word کار می‌کند، درست انگار که آن‌ها را با استفاده از Microsoft Word ایجاد کرده‌ایم.

این، یک مثال ساده از دستکاری Word از داخل Excel با استفاده از VBA است. این مورد می‌تواند بسیار مفید باشد، برای مثال ممکن است یک سند استاندارد همراه با جدول‌هایی داشته باشیم که ماکرو، آن را با داده‌های صفحه گسترده پر کرده است. می‌توانیم ماکرو را اجرا کنیم، داده‌ها به جدول‌های سندWord انتقال خواهند یافت.

اخیراً برنامه‌ای نوشتم که اطلاعات یک SLA Report (گزارش توافقنامه سطح خدمات) را پر می‌کرد. SLA Report، یک سند Word با جدول‌های داده و نمودارهای متعدد بود اما ورودی آن از ۹ صفحه گسترده حاصل می‌شد. من با استفاده از متدهایی که قبلاً با هم مرور کردیم قادر به نوشتن کدی بودم که روی صفحات گسترده مستقل کار می‌کرد. داده‌های مربوطه را استخراج کرده و آن‌ها را در جدول‌ها یا نمودارهای صحیح در سند Word قرار می‌داد. انجام این کار به صورت دستی حدوداً نصف روز یک کارمند را اشغال می‌کرد اما کد، این کار را در کمتر از ۵ دقیقه انجام می‌داد:

```
Sub Test_Word( )
Dim oWd As Word.Application, oWdoc As Word.Document
Dim or As Word.Range, ot As Word.Table
Set oWd = CreateObject("Word.Application")
Set oWdoc = oWd.Document.Add
Set or = oWdoc.Range
Set ot = oWdoc.Tables.Add(r, 4, 5)
ot.Cell(1, 1) .Range.Text = "test"
oWdoc.SaveAs ("c:\MyTest.doc")
oWdoc.Close
oWd.Quit
Set oWdoc = Nothing
Set oWd = Nothing
End Sub
```

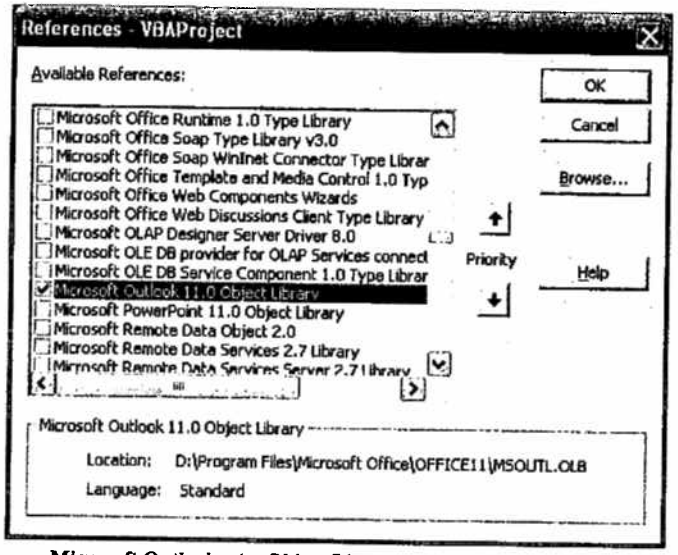
این مثال را نمی‌توانیم مستقیماً از داخل Excel اجرا کنیم مگر آن‌که به صورت دستی، داده‌ها را کپی کرده و بچسبانیم که در صورت زیاد بودن داده‌ها، کار پرزحمتی خواهد بود. این، یک مثال خوب از زبان برنامه نویسی ماکرو است که قدرت زیادی به کاربر می‌دهد تا کارها را در مایکروسافت به صورت خودکار درآورد.

راه اندازی Microsoft Outlook

می‌توانیم از همان تکنیک برای راه اندازی Microsoft Outlook استفاده کرده و آن را وادار کنیم تا نامه‌های الکترونیکی را از صفحه گسترده شما ارسال کند و موارد ثبت شده در کتاب آدرس را به دست آورد. البته خصوصیات نامه الکترونیکی در Excel برای ارسال یک صفحه گسترده کامل وجود دارند اما این متد به شما اجازه می‌دهد تنها بخشی از برگه را در عملیات شرکت دهید.

برای استفاده از این کد باید برنامه Microsoft Outlook روی کامپیوتر شما نصب شده باشد (با Outlook Express فرق دارد).

برای شروع کار، فایل Object Library را برای Outlook اضافه کنید. این عمل را می‌توانیم با انتخاب References | Tools از منوی Visual Basic Editor انجام دهیم. Microsoft Outlook Object را انتخاب کرده و سپس روی کادر کناری آن تیک می‌زنیم (تصویر ۲-۱۴).



تصویر ۲-۱۴ انتخاب فایل Object Library برای Microsoft Outlook

سپس کد زیر را در یک ماژول وارد می‌کنیم:

```
Sub Test_Outlook()
Dim oFolder As Outlook.MAPIFolder
Dim oItem As Outlook.MailItem
Dim oOutlook As New Outlook.Application
Dim MoOutlook As Outlook.NameSpace
Set MoOutlook = oOutlook.GetNameSpace("MAPI")
Set Ofolder = MoOutlook.GetDefaultFolder(olFolderOutbox)
Set oItem = oFolder.Items.Add(olMailItem)
With oItem
        .Recipients.Add ("richard.shepherd@anywhere.com")
        .Subject = "Test Excel Email"
        .Body = "This is a test of an email from Excel VBA"
        .Importance = olImportanceHigh
        .Send
End with
Set oItem = Nothing
Set ofolder = Nothing
End Sub
```

چهار خط اول کد، متغیرهایی را برمبنای Microsoft Outlook ایجاد می‌کنند که برای برنامه Outlook، NameSpace پوشه Outlook و گزینه Mail به کار می روند. متغیر Outlook برای اشاره به namespace MAPI ایجاد شده است که یکی از لایه‌های ارایه‌گر خدمات انتقال پیغام (messaging) را نشان می‌دهد که Outlook برای ذخیره سازی داده‌ها به آن نیاز دارد. MAPI تنها نوع namespace است که Outlook از آن پشتیبانی می‌کند.

سپس متغیر oFolder به عنوان پوشه پیش‌فرض برای Outlook برای namespace تنظیم می‌شود. این باعث ایجاد شیئی می‌شود که Outlook را نشان می‌دهد و از این پس می‌توانیم گزینه پست (mail item) را در آن قرار دهیم. این‌کار را با تنظیم یک گزینه پستی جدید در آن پوشه برای oItem انجام می‌دهیم. این امر دقیقاً مانند وقتی است که از خود Microsoft Outlook، یک گزینه پستی جدید را باز می‌کنیم.

آدرس گیرنده اضافه می‌شود. در این بخش می‌توانیم برای آزمایش کردن این مثال، آدرس شخصی خود را به صورت یک رشته وارد کنیم. اگر این فرد در داخل شبکه ما باشد و در لیست آدرس Outlook قرار داشته باشد کافی است تنها نام او را وارد کنیم.Subject چیزی است که دوست داریم در عنوان پیغام دیده شود و Body متن نامه الکترونیکی است.

Importance به ما اجازه می‌دهد تا تقدم نامه الکترونیکی را تنظیم کنیم. به موازات تایپ این دستور،

مقادیر ثابت برای Importance در یک کادر فهرست ظاهر می‌شوند و به شرح زیر هستند:

olImportanceHigh
olImportanceNormal
olImportanceLow

Send به معنای ارسال واقعی نامه الکترونیکی است. وقتی نامه الکترونیکی ارسال شد، مقادیر متغیرهای oItem و oFolder برابر با Nothing تنظیم شده و حافظه آزاد می‌شود.

توجه کنید که این کد از همان تکنولوژی استفاده می‌کند که بسیاری از ویروس‌های پست الکترونیکی از آن استفاده می‌کنند (از همه مهم‌تر، ویروس Love Bug است). لازم است در استفاده از این کد دقت زیادی به خرج دهیم چون اگر مراقب نباشیم ممکن است با ارسال نامه‌های الکترونیکی بسیار زیاد به صورت خودکار خاتمه یابد که باعث بروز مشکل در سیستم نامه‌رسانی می‌شود و گیرندگان را عصبانی می‌کند.

البته مایکروسافت برای نسخه‌های جدید Outlook، امنیت بیشتری ایجاد کرده است و هرچند این کد مشکل آفرین نیست اما کاربر، یک کادر پیغام خواهد دید که به او اطلاع می‌دهد یک نامه الکترونیکی ارسال می‌شود و لازم است روی OK کلیک کند تا نامه برایش ارسال شود.

راه اندازی Excel از دیگر برنامه‌های Office

در بخش‌های قبلی دیدید که از Excel چگونه برای راه اندازی دیگر برنامه‌های مایکروسافت مانند Word و Outlook استفاده می‌کنیم.

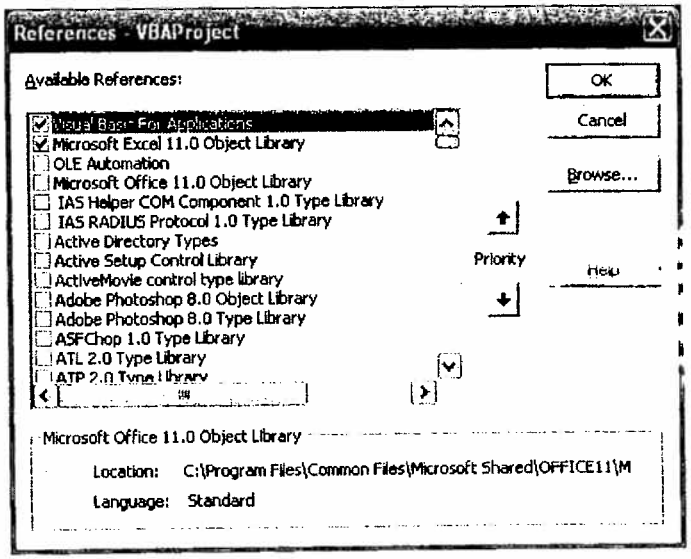
به همین نحو می‌توانیم از دیگر برنامه‌ها نیز برای کنترل Excel استفاده کنیم. برای مثال می‌توانیم یک ماکرو در Word یا Access بنویسیم که یک صفحه گسترده Excel را باز کرده و سپس آن‌را ذخیره کند. مجبور نیستیم حتماً خود برنامه Excel را اجرا کنیم و حتی لازم نیست خود صفحه گسترده روی صفحه نمایش دیده شود. اگر Excel اجرا شود یک کارپوشه جدید ایجاد خواهد شد اما می‌توانیم ویژگی Visible را برای برنامه خود برابر False قرار دهیم تا صفحه گسترده هرگز ظاهر نشود. وقتی این نوع برنامه اجرا می‌شود، بسیار جالب خواهد بود که ویژگی Visible را برابر True قرار دهیم و اعدادی را که در صفحه گسترده ظاهر می‌شوند، ببینیم. انگار یک اپراتور نامریی در حال اضافه کردن آن‌ها به صفحه گسترده است. اما اگر کسی به اشتباه روی صفحه کلید کلیک کند یا حتی اگر صفحه گسترده بسته شود می‌تواند باعث بروز نتایج غیرمترقبه‌ای شود. اگر ویژگی Visible را برابر False تعیین کنیم، کاربر هیچ کاری نمی‌تواند انجام دهد و حتی لازم نیست از آن‌چه در حال انجام است آگاه باشد.

مثالی که من از آن استفاده می‌کنم، یک صفحه گسترده Excel را از داخل یک سند Word ایجاد می‌کند. VBA دقیقاً مانند تمام برنامه‌های مایکروسافت Office کار می‌کند به استثنای این‌که مدل شئ در هر برنامه کاربردی با دیگر برنامه‌ها متفاوت است تا کارآیی خاص هر برنامه را ایجاد کند.

وقتی برنامه Word بارگذاری شود لازم است وارد پنجره کدنویسی VBA شویم. این عملیات دقیقاً مانند Excel و با فشار دادن کلید ALT+F1 انجام می‌شود. عملکرد پنجره کدنویسی هم مانند Excel است.

وقتی از برنامه‌ای غیر از Excel استفاده می‌کنیم باید ابتدا ارجاعی به فایل Object Library داشته باشیم. باید همین کار را در Word با قرار ارجاع به Excel Object Library انجام دهیم تا به Word بگوییم چگونه مدل شئ Excel را پیدا کند.

مانند قبل، از Tools | References از منوی Visual Basic Editor استفاده می‌کنیم اما این‌بار، Excel Object Libraryرا انتخاب کرده و کادر کنار آن‌را تیک می‌زنیم (تصویر ۳-۱۴):



تصویر ۳-۱۴ انتخاب فایل Excel Library

این‌کار تمام آن‌چه را که برای دستکاری Excel مورد نیاز است، در اختیار کد شما قرار می‌دهد. با انتخاب Insert | Module از منوی Code، یک ماژول درج کرده و سپس کد زیر را در آن وارد کنید:

```
Sub Test_Excel ()
Dim oEapp As Excel.Application
Set oEapp = CreateObject("Excel.Application")
Dim oWBook As Workbook, oWSheet As Worksheet
Set oWBook = oEapp.Workbooks.Add
Set oWSheet = oWBook.Worksheets(1)
oWSheet.Range("a1") .Value = "My test Excel spreadsheet"
oWBook.SaveAs ("c:\TestExcel.xls")
oWBook.Close
oEapp.Quit
Set oWSheet = Nothing
Set oWBook = Nothing
End Sub
```

هنگامی که این کد اجرا شود، یک صفحه گسترده با نام c:\TestExcel.xls ایجاد می‌شود و مقدار "My test Excel spreadsheet" در سلول A1 روی Sheet1 قرار می‌گیرد. کد، متغیری با نام oEapp ایجاد می‌کند تا شی برنامه کاربردی Excel را در خود نگه دارد. سپس آن متغیر به عنوان یک شئ Excel تنظیم می‌شود. پس از آن، متغیرهای oWBook و oWSheet ایجاد می‌شوند تا یک کارپوشه و یک کاربرگ را نمایش دهند و کارپوشه به صورت یک کارپوشه جدید تنظیم شود که به مجموعه Workbooks اضافه می‌شود. شئ Worksheet به عنوان نخستین کاربرگ در مجموعه Worksheets تنظیم می‌شود (همیشه در یک کارپوشه باید حداقل یک کاربرگ وجود داشته باشد).

سلول A1 به گونه‌ای روی برگه نخست تنظیم می‌شود تا "My test Excel spreadsheet" خوانده شده و سپس Workbook به عنوان "c:\TestExcel.xls" تنظیم می‌شود. سپس با فراخواندن متد Clos، کارپوشه بسته می‌شود. این امر مانند انتخاب File | Exit از منوی Excel عمل می‌کند. در نهایت، مقدار متغیرهای Workbook و Worksheet برابر Nothing تنظیم می‌شود تا تمام حافظه اشغال شده توسط اشیا Excel آزاد شود.

همان‌طور که قبلاً هم گفتیم، بسیار مهم است که مطمئن شویم برنامه به درستی بسته شده و مقدار تمام اشیا ایجاد شده در کد VBA برابر Nothing تنظیم شده است. در غیر این صورت حافظه مصرف شده اشغال باقی می‌ماند و امکان استفاده از آن در دیگر برنامه‌ها وجود نخواهد داشت. این امر می‌تواند برای کاربران، در هنگام خروجشان از برنامه شما دردسر آفرین باشد چون یک برنامه به درستی بسته نشده و حافظه‌ای را که ممکن است مورد نیاز باشد، نگه داشته است.

بخش سوم

تکنیک‌های پیشرفته در VBA

در این بخش به بررسی چگونگی ایجاد نمودارها و گرافها در کد،کار با پایگاه داده‌های خارجی و استفاده از فراخوان‌های API (رابط برنامه‌های کاربردی) برای انجام کارهایی مانند اجرای یک فایل صوتی Wav می‌پردازیم. هم‌چنین به بررسی انیمیشین و ماژول‌های کلاس نیز خواهیم پرداخت. این‌ها موضوعات مهمی هستند که به شما اجازه می‌دهند تا کارهای غیر متعارف و جالبی با VBA و Excel انجام دهید.

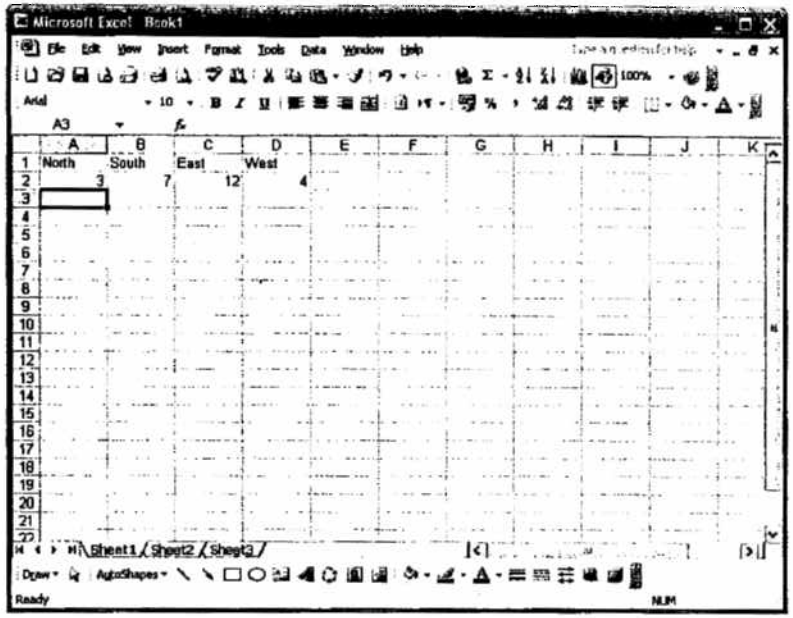
فصل پانزدهم

نمودارها و گراف‌ها

می‌توانیم به سادگی با استفاده از VBA و دستور Chart Wizard، نمودارها و گراف‌ها را ایجاد کنیم. از این دستور دقیقاً همان‌طور که آن‌را در برنامه Excel به‌کار می‌گرفتیم، استفاده می‌کنیم اما تمام دستورها و خصوصیات با استفاده ازVBA تنظیم می‌شوند. چون هر کاری که می‌توانید به عنوان یک کاربر Excel انجام دهید در مدل شئ نمایش داده شده است، انجام این‌کار بسیار ساده و صریح خواهد بود. ابتدا محدوده داده‌های مناسب را برای نمودار دایره‌ای انتخاب می‌کنیم (تصویر ۱-۱۵).

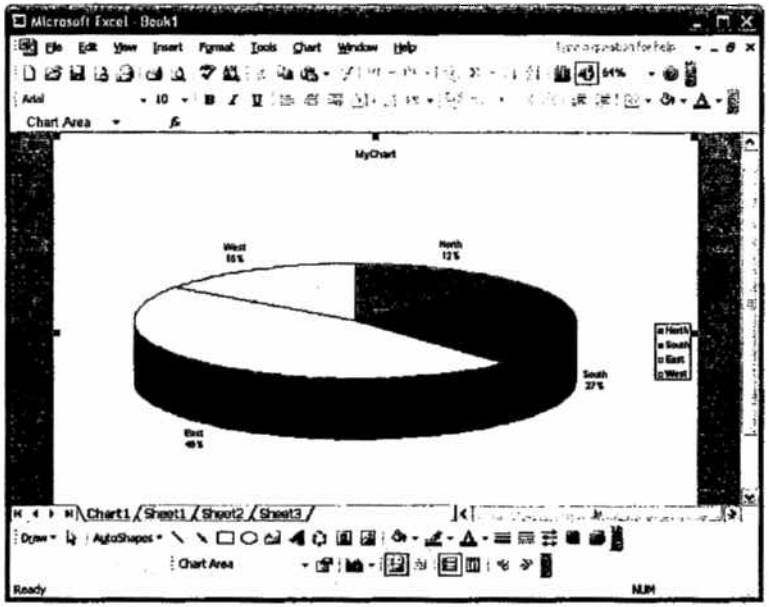
با انتخاب Insert | Name | Define از منوی صفحه گسترده، نام North را به محدوده انتخاب شده داده و سپس روی OK کلیک می‌کنیم تا کادر محاوره‌ای بسته شود. حال کد زیر را در یک ماژول وارد می‌کنیم:

```
Sub test_chart ()
Dim c As Excel.Chart
Set c = ActiveWorkbook.Charts.Add
c.ChartWizard Source:="north", gallery:=xl3DPie, Format:=7, _
Title:="MyChart" , categorylabels:=1
End Sub
```



تصویر ۱-۱۵ مجموعه داده‌های مناسب برای یک نمودار دایره‌ای

این کد ابتدا یک شئ نمودار با نام c ایجاد می‌کند و سپس یک نمودار جدید به کارپوشه فعال اضافه می‌کند. سپس از Chart Wizard برای اضافه کردن دامنه داده‌های منبع، عنوان و نوع نمودار استفاده می‌کند. با اجرای این کد باید نموداری را شبیه تصویر ۲-۱۵ ببینیم.



تصویر ۲-۱۵ یک نمودار دایره‌ای که با VBA ساخته شده است.

ساختار دستوری کامل متد ChartWizard را در این‌جا می‌بینید:

ChartWizard (Source, Gallery, Format, PlotBy, CategoryLabels, SeriesLabels, HasLegend, Title, CategoryTitle, ValueTitle, ExtraTitle)

پارامترهایی که برای متد ChartWizard بیش از همه مهم هستند عبارتند از :

- PlotBy
- CategoryLabels
- SeriesLabels

این پارامترها تعیین می‌کنند که منبع داده‌ها چگونه برای ارایه اطلاعات برچسب و داده‌ها و این‌که آیا در ستون‌ها یا ردیف‌ها نمایش داده شوند یا خیر، به کار گرفته می‌شوند. سعی کنید کد مثال قبلی را بدون پارامتر PlotBy اجرا کنید. می‌توانید اهمیت این پارامتر را به راحتی متوجه شوید چون نمودار شما، محدوده داده‌ها را به صورت نادرست می‌خواند و شاهد یک نمودار دایره‌ای خواهیم بود که تنها یک رنگ در آن وجود دارد. سعی کنید پارامتر Categorylabels را که در حال حاضر نشان می‌دهد

Row1 حاوی برچسب‌ها است. حذف کنید. آن‌گاه خواهید دید که هیچ برچسبی روی بخش‌های مختلف نمودار دایره‌ای وجود نخواهد داشت.

از لحاظ Chart Type (Gallery)، Excel نسخه 97 به بعد، پارامترهای داخلی برای تعریف انواع مختلف نمودارها دارد. این‌ها همگی مطابق با Gallery در Chart Wizard است که اگر یک نمودار را در یک صفحه گسترده درج کنیم خواهیم دید:

جدول ۱-۱۵

ثابت	شرح	مقدار
xlArea	نمودار مساحت	1
xlBar	نمودار میله‌ای	2
xlColumn	نمودار ستونی	3
xlLine	نمودار خطی	4
xlPie	نمودار دایره‌ای	5
xlCombination	نمودار ترکیبی	-4111
xl3DArea	نمودار سه بعدی مساحت	-4098
xl3DBar	نمودار سه بعدی میله‌ای	-4099
xl3DColumn	نمودار سه بعدی ستونی	-4100
xl3DLine	نمودار سه بعدی خطی	-4101
xl3DPie	نمودار سه بعدی دایره‌ای	-4102
xl3DSurface	نمودار سه بعدی سطحی	-4103
xlDoughnut	نمودار سه بعدی حلقه‌ای	-4120

برای ویژگی PlotBy از پارامترهای زیر استفاده می‌کنیم:

xlRows = 1
xlColumns = 2

با بعضی از این نمودارها، بهتر است که از Chart Wizard در داخل صفحه گسترده استفاده کنیم تا ببینیم چه گزینه‌های قابل انتخابی وجود دارند و نیز ببینیم نتایج، پیش از نوشتن کد VBA برای انجام آن، چگونه خواهند بود.

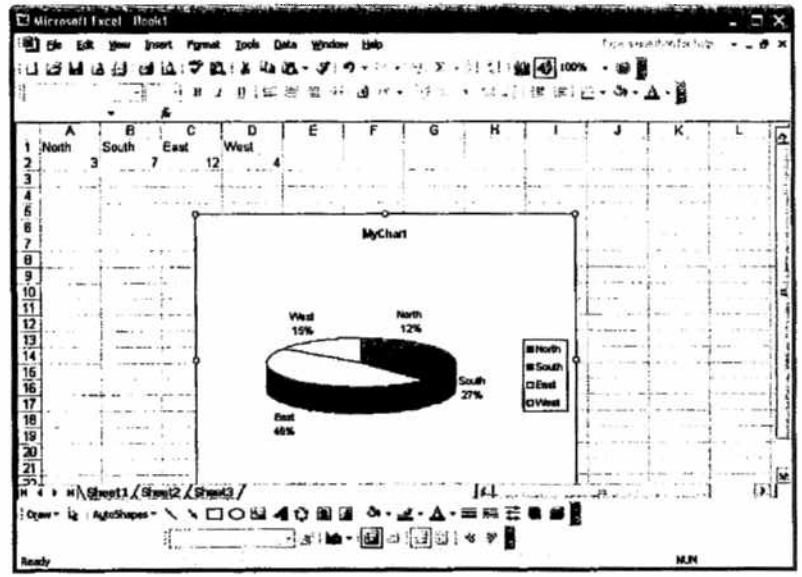
تا این‌جای کار تنها با ایجاد یک برگه نمودار برای نمایش نمودار روی آن برگه سروکار داشته‌ایم اما می‌توانیم نموداری روی صفحه گسترده نیز ایجاد کنیم اگر چه کار، کمی پیچیده‌تر می‌شود:

```
Sub test_chart ()
Dim c As Excel.Chart, a As Worksheet, co As ChartObjects
Set a = Worksheets("sheet1")
Set co = a.ChartObjects
Set c = co.Add(60, 60, 300, 300).Chart
co.Select
C.chartwizard Plotby:=XLRows, Source:="north", gallery:=xl3DPie, Format:=7,
Title:="MyChart" , categorylabels:=1
End Sub
```

این کد، مشابه مثال قبلی است و متد ChartWizard هنوز هم برای ایجاد نمودار استفاده می‌شود اما از اشیای بیشتری استفاده می‌کند.

در خط اول کد، یک شئ برای یک کاربرگ ایجاد می‌کنیم. سپس یک شئ برای تعریف مجموعه ChartObjects ایجاد می‌شود. بعد شئ Worksheet تنظیم می‌شود تا جایی در کاربرگ، که می‌خواهید نمودار در آن‌جا به نمایش درآید، مشخص کند. مجموعه ChartObjects را تنظیم می‌کنیم تا به مجموعه ChartObjects برای آن کاربرگ اشاره کند. سپس یک نمودار جدید اضافه می‌کنیم اما باید مختصاتی که محل ظاهر شدن نمودار روی صفحه گسترده را تعریف می‌کند، مشخص کنیم (همراه با ارتفاع و پهنا).

سپس شئ Chart انتخاب می‌شود؛ در غیر این صورت، این شئ مرئی نخواهد بود تا زمانی‌که کاربر مکان‌نما را جابه‌جا کند. سپس متد ChartWizard، نموداری را به همان شیوه قبل ایجاد می‌کند. اگر این کد را اجرا کنیم، نتیجه نشان داده شده در تصویر ۳-۱۵ را به‌دست خواهیم آورد.



تصویر ۳-۱۵ نموداری که با استفاده از VBA در یک صفحه گسترده قرار گرفته است.

فصل شانزدهم

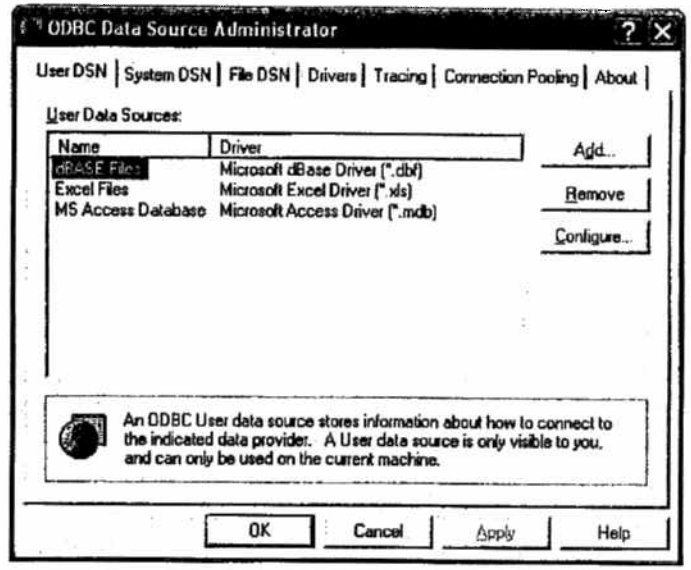
کار با پایگاه داده‌ها

تا زمانی که یک پایگاه داده، سازگار با ODBC (اتصال پایگاه داده باز) باشد و درایو مناسب هم موجود باشد می‌توانیم کاری کنیم که کد ما با پایگاه داده ارتباط پیدا کند. ODBC به برنامه‌های کاربردی اجازه می‌دهد تا به شیوه‌ای استاندارد از بسیاری از زبان‌های برنامه‌نویسی مختلف (مانندVBA) به پایگاه داده دسترسی پیدا کنند. این امر در صورتی که بخواهیم داده‌ها را از یک پایگاه داده به صفحه گسترده خود انتقال دهیم بسیار مفید خواهد بود. برای مثال ممکن است بخواهیم داده‌ها را از یک سیستم حسابداری وارد برنامه خود کنیم. اگر پایگاده داده، یک درایو ODBC در اختیار داشته باشد، می‌توانیم داده‌ها را به هر شکل و فرم وارد برنامه VBA کنیم تا در صورت نیاز از آن‌ها استفاده نماییم.

پایگاده داده‌ها Access، Microsoft SQL Server و Oracle، همگی از ODBC پشتیبانی می‌کنند و به شرط آن‌که اجازه دسترسی به پایگاه داده را داشته باشیم، می‌توانیم داده‌ها را درصفحه گسترده خود بخوانیم و حتی داده‌ها را در صورت نیاز مجدداً در پایگاه داده بازنویسی کنیم. البته بازنویسی داده‌ها در پایگاه داده باید بسیار با احتیاط انجام شود چون ممکن است به سادگی باعث از بین رفتن یکپارچگی پایگاه داده مربوطه شود.

لینک‌های ODBC

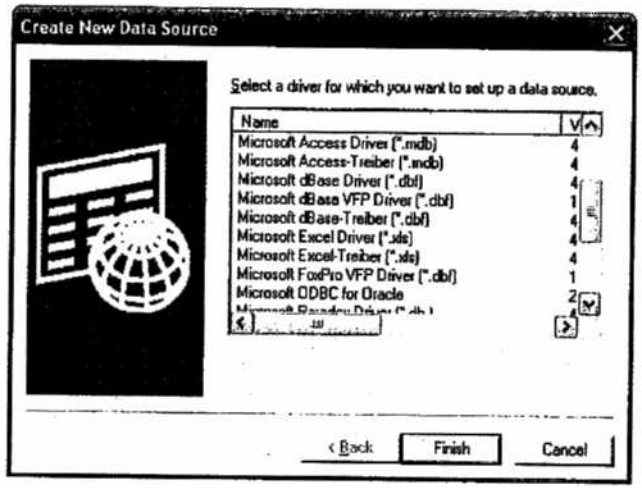
لازم است ابتدا یک لینک ODBC ایجاد شود تا پایگاه داده‌ای را که قرار است با آن کار کنیم مشخص کند. با انتخاب Control Panel از ویندوز و سپس Administrative Tools و Data Sources(ODBC) می‌توانیم این کار را انجام دهیم. خواهید دید که کادر محاوره‌ای Data Sources Administrator ODBC شبیه تصویر ۱-۱۶ است.



تصویر ۱-۱۶ تنظیم نام منبع داده

اگر قبلاً یک DSN (نام منبع داده) برای پایگاه داده ایجاد نشده باشد باید روی دکمه Add کلیک کنید. این کار باعث می‌شود لیستی از درایوهای پایگاه داده موجود در اختیار شما قرار گیرد(تصویر ۲-۱۶).

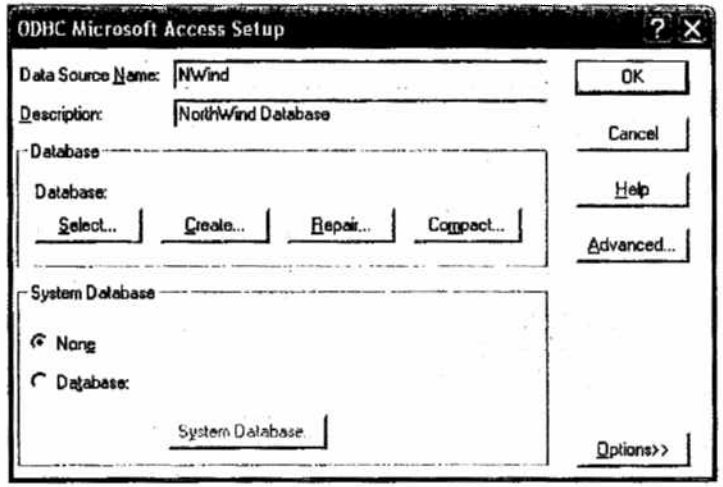
DSN به معنای نام یک لینک ODBC به پایگاه داده‌ای است که می‌توانیم از آن پایگاه داده استفاده کنیم تا به آن لینک ODBC ارجاع نماییم. این پایگاه داده‌ها بسیار متنوع هستند اما ما در این‌جا روی Microsoft Access متمرکز می‌شویم چون یک برنامه مهم در مجموعه Microsoft Office محسوب می‌شود و اگر برنامه Excel را نصب کرده باشیم، به احتمال قوی این برنامه هم روی کامپیوتر نصب شده است.



تصویر ۲-۱۶ انتخاب درایور ODBC برای منبع داده

در بالای لیست، دوبار روی درایو Access کلیک می‌کنیم تا به صفحه‌ای که در تصویر ۳-۱۶ نشان داده شده است برویم. در آن صفحه به جای Data Sources Name، مقدار NWind و به جای Description مقدار Northwind را وارد کنید. این کار باعث می‌شود یک نام به DSN داده شود که می‌توانیم در کدمان از آن استفاده کنیم و شرحی (Description) ارایه می‌شود که بیان می‌دارد DSN درباره چیست. اگر بخواهیم، می‌توانیم فیلد Description را خالی بگذاریم چون تنها مشخص می‌کند که DSN درباره چیست و اصلاً در کد ما به کار گرفته نمی‌شود.

روی دکمه Select کلیک کرده و پایگاه داده Northwind را انتخاب کنید. این یک پایگاه داده ساده (فایل MDB) است که همراه با Microsoft Access وجود دارد و باید در بین فایل‌های Access قرار داشته باشد. اگر نتوانستید آن‌را پیدا کنید از ویندوز، Search را انتخاب کرده و در قسمت Files and Folders، نام آن‌را وارد کنید. در مرحله بعد روی OK کلیک کنید تا یک لینک ODBC به پایگاه داده‌ای با نام NWind ایجاد کنید. دوباره روی OK کلیک کنید تا Control Panel بسته شود.



تصویر ۳-۱۶ مراحل نهایی برای راه اندازی یک DSN

به همین روش می‌توانیم لینک‌های ODBC را به دیگر پایگاه داده‌ها روی سرورهایی مانند SQL یا Oracle نیز راه اندازی کنیم. البته لازم است نام سرور و نام پایگاه داده را بدانیم و هم‌چنین به یک ID و کلمه عبور نیاز داریم که حداقل Select Access را ایجاد کند. هم‌چنین بسته به درایور ODBC که به کار می‌گیریم، ممکن است به تنظیمات دیگری هم نیاز داشته باشیم.

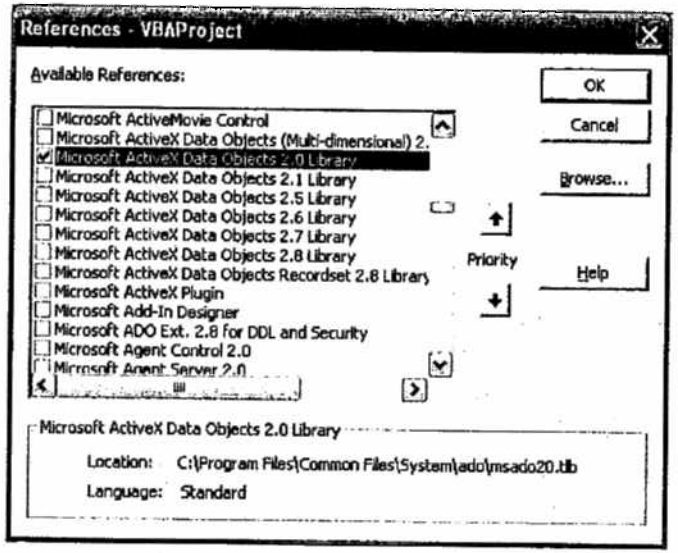
استفاده از ADO

ADO[1]، جدیدترین فناوری مایکروسافت برای اتصال به پایگاه داده‌هاست. ADO، یک COM (مدل شئ مؤلفه‌ای یا Component Object module) است که می‌تواند ما را در استفاده از داده‌های لینک ODBC که قبلاً ایجاد کرده‌ایم راهنمایی کند. لینک ODBC به کد ما می‌گوید پایگاه داده در کجا واقع است و ID و کلمه عبور برای رسیدن به آن‌را در اختیارمان قرار می‌دهد. ADO، ابزاری در اختیار ما قرار می‌دهد تا با استفاده از لینک ODBC و برای خواندن و نوشتن داده‌ها، به آن پایگاه داده متصل شویم.

برای استفاده از ADO در کد، باید ابتدا با استفاده از Tools | References از منوی VBE، ارجاعی به Object Library داشته باشیم. در لیست، پایین روید تا به Microsoft ActiveX Data Objects

1- ActiveX Data Objects

Recordset 2.7 Library برسید (تصویر ۴-۱۶). ممکن است بسته به نسخه سیستم عامل ویندوزتان، نسخه‌های قدیمی‌تر آن‌را داشته باشید اما آن‌ها هم به همان شیوه عمل می‌کنند. اگر نسخه 2.7 را ندارید از آخرین نسخه‌ای که در اختیار دارید استفاده کنید. هر دو کادر انتخابی سمت چپ را تیک بزنید و روی OK کلید کنید.



تصویر ۴-۱۶ قرار دادن یک ارجاع به Active Data Objects

اکنون می‌توانیم از کد مثالی زیر استفاده کنیم:

```
Sub Test_Db ( )
Dim MyCon As New Connection
MyCon.Open "NWind"
Set rs = New Recordset
rs.Open "select firstname, lastname, title from employees", MyCon, _
adOpenForwardOnly, adLockReadOnly, adCmdText
co = 1
Do Until rs.EOF
        ActiveSheet.Range("a" & co).Value = rs!firstname

        ActiveSheet.Range("b" & co).Value = rs!lastname
```

```
        ActiveSheet.Range("c" & co).Value = rs!Title

        co = co + 1

        rs.MoveNext
        Loop
rs.Close
MyCon.Close
End Sub
```

اولین کاری که این کد انجام می‌دهد ایجاد شئ MyCon به عنوان شئ Connection است، سپس شئ Connection با استفاده از DSN با نام NWind که قبلاً ایجاد کرده‌ایم باز می‌شود و بر مبنای اطلاعاتی که در DSN وارد کرده‌ایم، یک اتصال به پایگاه داده ایجاد می‌کند.

سپس کد، شئ Recordset را ایجاد می‌کند. Recordset، شیئی است که دسته‌ای از داده‌ها را که می‌توانند یک پرس و جو (query)، جدول یا دستور باشند، نمایش می‌دهد. در این مورد، این شئ داده‌ها را از جدول Employee می‌گیرد.

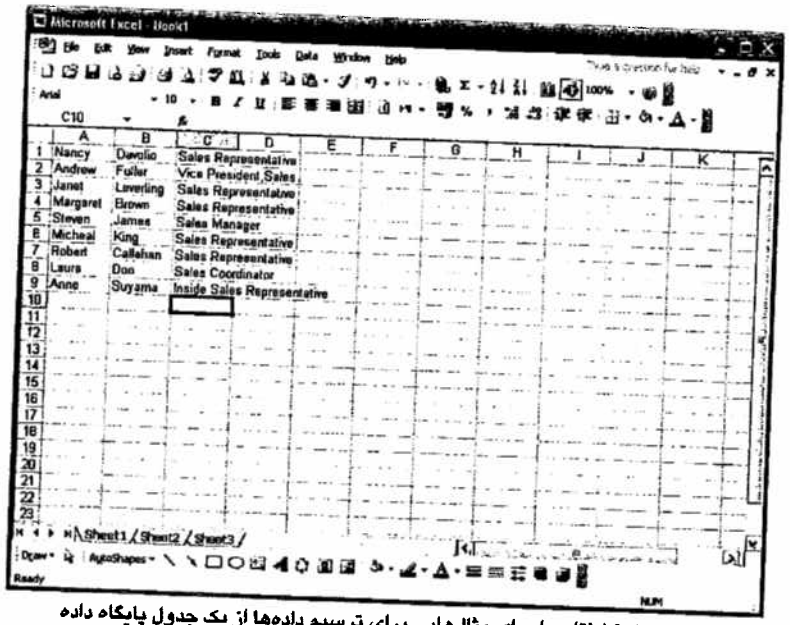
متغیری با نام co برابر با ۱ تنظیم شده است که یک نقطه ارجاع دینامیکی برای نوشتن داده‌ها به صورت ردیف به ردیف در صفحه گسترده ایجاد می‌کند. به موازات این‌که هر ردیف داده را می‌نویسید، مقدار این متغیر افزایش می‌یابد تا به ردیف بعدی اشاره کند.

مطمئن شوید که خط کد rs.MoveNext حتماً نوشته شده باشد. این خط دستورالعمل انتقال اشاره‌گر رکورد به رکورد بعدی در Recordset (مجموعه رکورد) است. جا افتادن این خط یکی از اشتباهات متداول است اما حذف این خط باعث می‌شود هربار همان رکورد قبلی خوانده شود و هرگز به مشخصهEOF نرسد و در واقع کد، در یک حلقه بی‌پایان گیر کند.

سپس کد، حلقه را اجرا می‌کند، ردیفی از پایگاده داده را می‌خواند و ردیف داده‌ها را در صفحه گسترده می نویسد و این کار را ادامه می‌دهد تا به مشخصه EOF در مجموعه رکورد برسد و این به معنای آن است که همه رکوردها مرور شده‌اند.

سپس کد با استفاده از شئ ActiveSheet، فیلدهای firstname ، lastname و title را از مجموعه رکورد، به ترتیب در ستون‌های A ، B و C می‌نویسد. توجه کنید که روی شئ Recordset به جای نقطه (.) از علامت تعجب (!) استفاده شده است. این به آن دلیل است که نشان دهیم روی شئ Recordset، از نام فیلد استفاده می‌کنیم نه از یک ویژگی یا متد (که برای آن‌ها از علامت نقطه(.) استفاده می‌کردیم).

در آخر Connection و Recordset بسته می‌شوند چون اگر آن‌ها باز بمانند ممکن است دیگر کاربران با مشکل مواجه شوند. اکنون صفحه گسترده باید شبیه تصویر ۵-۱۶ باشد.



تصویر ۵-۱۶ نتایج اجرای مثال‌هایی برای ترسیم داده‌ها از یک جدول پایگاه داده

این کد VBA به ما اجازه می‌دهد تا پرس و جوهایی (query) روی پایگاه داده دیگری اجرا کنیم و داده‌ها را در هر جایی از کارپوشه که می‌خواهیم، قرار دهیم. با استفاده از به روز کردن یک پرس و جو حتی می‌توانیم داده‌ها را دوباره در پایگاه داده بازنویسی کنیم.

فصل هفدهم

فراخوان‌های API

اگر چه مدل شئ Excel و کد VBA در ارایه متدهایی برای انجام عملیات گوناگون روی صفحه گسترده، بسیار جامع و کامل است، اما ممکن است متوجه شده باشید که کارها و عملیات را باید در قبال ویندوز انجام دهید که قادر به اجرای آن‌ها نیستید. حتی خود ویژوال بیسیک هم ابزار انجام این کارها را به طور مستقیم در اختیار ندارد.

برای مثال نمی‌توان مقدار فضای خالی موجود روی یک دیسک را مشخص کرد. نمی‌توان مستقیماً صفحه کلید را خواند (تنها می‌توانیم کلیدهای ورودی روی فرم کاربری را بخوانیم). نمی‌توان موقعیت و محل ماوس را مشخص کرد. واضح است که روش‌هایی برای انجام این کارها وجود دارد چون Windows Explorer می‌تواند فضای خالی دیسک را مشخص کند و ویندوز هم می‌داند کدام کلید روی صفحه کلید فشرده شده است. VBA، دستورات مستقیم برای انجام این موارد را ندارد، اما اجازه دستیابی به رابط برنامه‌نویسی کاربردی WIN32 (WIN32API) را به ما می‌دهد که آن نیز به نوبه خود، اجازه می‌دهد مستقیماً به مجموعه با ارزشی از اطلاعات دست پیدا کنیم.

فراخوان‌های API، موضوع بسیار پیشرفته‌ای هستند و می‌توانند مقدار کارآیی بسیار زیادی برای برنامه‌های ما ایجاد کنند. البته بررسی کامل و مشروح فراخوان‌های API در حوزه کاری این کتاب نیست اما چند مثال از چگونگی استفاده از آن‌ها را ارایه می‌کنیم.

فراخوان API چیست؟

API (رابط برنامه‌نویسی کاربردی) به شما اجازه می‌دهد تا به توابع برنامه‌نویسی داخلی فایل‌های DLL و EXE دست پیدا کنید (به‌ویژه توابعی که ویندوز را راه‌اندازی می‌کنند). دیگر برنامه‌های کاربردی طرف سوم نیز از فایل‌های DLL برای کتابخانه‌های توابع استفاده می‌کنند.

فراخوان‌های API معمولاً توابعی هستند که مقداری را برمی‌گردانند هرچند که در همان زمان، عملیاتی را نیز اجرا می‌کنند. هم‌چنین زیرروال‌هایی هستند که تنها یک عملیات را اجرا می‌کنند اما مقداری برنمی‌گردانند (تفاوت تابع و زیرروال را که در فصل‌های قبلی ذکر شد، به یاد آورید). برای استفاده از آن‌ها باید ابتدا تابع یا زیرروالی را که می‌خواهیم از آن استفاده کنیم، اعلان کنیم و این، سخت‌ترین بخش کار است. دستور Declare، شرحی از تابع یا زیرروال را در فایل DLL (کتابخانه پیوند

دینامیکی) ایجاد می‌کند. این دستور، شرح می‌دهد که کدام DLL به کار گرفته شده است، نام تابع یا زیرروال چیست و چه پارامترهایی به آن انتقال یافته‌اند. اعلان‌ها خیلی پیچیده هستند اگر خطایی رخ دهد فراخوان جواب نخواهد داد و حتی ممکن است کل سیستم از کار بیفتد. پیش از این‌که یک فراخوان API صادر شود، مطمئن شوید فایل شما ذخیره شده است چون در صورت بروز خطا، نتایج اسفباری به بار خواهد آمد. البته باید کامپیوتر را مجدداً بارگذاری کنید (reboot) تا همه چیز دوباره اجرا شود و این امر باعث می‌شود داده‌هایی که ذخیره نشده‌اند، از بین بروند. فراخوان‌های API در صورت نادرست بودن، اصلاً قابل بخشش نیستند و حتی فشار دادن CTRL+BREAK برای متوقف کردن آن‌ها نیز بی‌اثر است.

استفاده از فراخوان API

در این‌جا چند مثال از فراخوان‌های API و چگونگی استفاده از آن‌ها در کد VBA ذکر می شود:

به دست آوردن فضای موجود در دیسک

قصد داریم در این مثال از یک فراخوان API استفاده کنیم تا دیسک یدکی را از یک دیسک مشخص به دست آوریم. برای رکورد، این فراخوان API خاص تا حدود 2GB را می‌خواند. فراخوان الی API مایکروسافت تنها درایوهای دیسک کمتر از 2GB را می‌خواند. به موازات آن‌که اندازه درایو دیسک سخت افزایش یافت، مایکروسافت مجبور شد فراخوان API جدید را به کار گیرد.

پیش از هر چیز باید یک اعلان ایجاد کنیم. این کار را در بخش declaration یک ماژول انجام می‌دهیم. ساختار دستوری این اعلان خاص به صورت زیر است:

```
Private Declare Function GetDiskFreeSpaceEx Lib "kernel32" _
Alias " GetDiskFreeSpaceExA" (ByVal lpDirectoryName As _
String , lpFreeBytesAvailableToCaller As Currency, _
lpTotalNumberOfBytes As Currency, lpTotalNumberOfFreeBytes _
As Currency) As Long
```

این، یک دستور کاملاً طولانی است و باید بسیار دقیق باشد تا جواب دهد. اگر به ویژوال بیسیک دسترسی داشته باشید، تمام اعلان‌های API در فایلی با نام API32.TXT قرار خواهند داشت و می‌توان آن‌ها را به سادگی کپی کرده و در بخش اعلان‌های خود بچسبانید. اطلاعات بیشتر در این زمینه را می‌توان از MSDN (شبکه توسعه گر مایکروسافت) به آدرس www.microsoft.msdn.com به دست آورد که مکان مناسبی برای یافتن اطلاعات پیشرفته درباره فراخوان‌های API و VBA است.

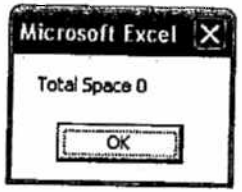
در اصل این دستور، ارجاعی به kernel32.dll ایجاد می‌کند که در پوشه system ویندوز قرار می‌گیرد و روش به کارگیری پارامترها و آن‌چه فرا می‌خوانند، مشخص می‌کند. هدف آن، قرار دادن تابعی در کد شما است که می‌توانید از آن برای فراخوانی این تابع کتابخانه از kernel32.dll استفاده کنید.

مرحله بعدی، قرار دادن کدی برای فراخواندن این تابع است:

```
Sub Test_Api ( )
Dim FreeBytesAvailableToCaller As Currency, TotalNumberOfBytes As _
Currency, TotalNumberOfFreeBytes As Currency
x = GetDiskFreeSpaceEx("c:\", FreeBytesAvailableToCaller, _
TotalNumberOfBytes, TotalNumberOfFreeBytes)
MsgBox "Total Space " & Format(TotalNumberOfBytes * 10000 , "#,##0")
MsgBox "Free Space " & Format(TotalNumberOfFreeBytes * 10000, "#,##0")
End Sub
```

این کد، متغیرهایی را برقرار می‌کند تا مقادیر برگشتی از فراخوان API را در خود نگه دارد. مقدار واقعی برگشتی از طریق متغیر فراخوان آن (که در این مورد، متغیر x است)، مرتبط با خطاهاست. اگر مقدار آن برابر 1 باشد به معنای موفقیت در انجام کار است.

سپس فراخوان، فهرست ریشه "C:\" و متغیرهای از قبل تعریف شده را انتقال می‌دهد. سپس مقدار در 10000 ضرب شده و با کاما (،) قالب بندی می‌شود تا نمایش آن ساده‌تر شود. می‌توانید مقادیر را با اجرای Windows Explorer یا NT Explorer و انتخاب Properties | File کنترل کنید (تصویر ۱-۱۷).



تصویر ۱-۱۷ گرفتن مجموع فضا از یک درایو دیسک

خواندن و نوشتن در فایل‌های INI

فراخوان‌های API می‌توانند برای نوشتن و خواندن در فایل‌ها نیز به کار گرفته شوند. عملیات مفید دیگر آن، ایجاد فایل‌های INI برای برنامه است. فایل‌های INI، ابزار ذخیره کردن تنظیمات برنامه ما

است. متغیرها و ویژگی‌ها در VBA تنها در هنگام اجرای برنامه مقادیرشان را نگه می‌دارند. وقتی برنامه بسته شود، همه چیز از بین می‌رود و همه آن‌ها به حالت پیش فرض درمی‌آیند.
برای مثال می‌توان به یک کادر متنی اشاره کرد که مسیر رسیدن به یک فهرست را که توسط کاربر تنظیم شده است در خود نگه می‌دارد. کاربر ممکن است مسیر دیگری را وارد کند اما وقتی برنامه بسته شود این کادر به حالت پیش‌فرض برمی‌گردد. این کار برای کاربر دردسرآفرین خواهد بود چون هر بار مجبور است این مسیر را دوباره وارد کند.
فایل INI این اطلاعات را به صورت محلی نگه می‌دارد به طوری که وقتی برنامه در مرتبه بعدی بارگذاری می‌شود، بتواند آن اطلاعات را بازیابی کند. البته می‌توانیم بخشی از صفحه گسترده را برای ذخیره کردن این مقادیر به کار گیریم اما این بخش ممکن است توسط داده‌های دیگر مورد بازنویسی قرار گیرد و امنیت کمی دارد.
دو اعلان وجود دارد که می‌توانیم برای فایل INI به کار گیریم (البته اعلان‌های دیگری هم برای خواندن و نوشتن در فایل‌های INI وجود دارند اما در این مثال فقط از این دو اعلان استفاده می‌کنیم):

```
Private Declare Function GetPrivateProfileString Lib "kernel32" Alias _
"GetPrivateProfileStringA" (ByVal lpApplicationName As String, ByVal _
lpKeyName As Any, ByVal lpDefault As String, ByVal lpReturnedString, As _
String, ByVal nSize As Long, ByVal lpFileName As String) As Long

Private Declare Function WritePrivateProfileString Lib "kernel32" Alias _
"WritePrivateProfileStringA" (ByVal lpApplicationName As String, ByVal _
lpKeyName As Any, lpString As Any, ByVal lpFileName As String) As Long
```

لازم است کد فوق را به بخش declaration ماژول اضافه کنیم. آن‌ها ارجاع‌هایی به فایل kernel32.dll ایجاد کرده و چگونگی انتقال پارامترها را تشریح می‌کنند. سپس می‌توانید از کد برای انتقال پارامترها و نوشتن در فایل استفاده کنید:

```
Sub Test_INI ( )
x = WritePrivateProfileString("Parameters", "Path", "C:\temp", "myini.ini")
s$ = Space$(256)
x = GetPrivateProfileString("Parameters", "Path", "", s$, 256, "myini.ini")
MsgBox s$
MsgBox x
End Sub
```

خط نخست کد، فایل INI را می‌نویسد. اگر فایل از قبل در آن‌جا نباشد، به طور خودکار ایجاد خواهد شد. محل پیش‌فرض در پوشه Windows است هرچند می‌توانیم این مسیر را تغییر دهیم.

پارامتر اول (Parameters)، بخشی از فایل INI برای نوشتن رشته جدید است. پارامتر دوم، نام کلید یا ورودی مجموعه است که اگر Null باشد، تمام کلیدها (keys) را در این بخش پاک می‌کند. پارامتر سوم، مقداری برای نوشتن کلید در خود دارد. مقدار این پارامتر، "C:\temp\" است و اگر برابر با Null تنظیم شود، رشته موجود برای آن کلید را پاک می‌کند. پارامتر چهارم، نام فایل INI است. می‌توانیم از هر پسوندی استفاده کنیم و لازم نیست حتماً از INI استفاده نماییم.
اکنون این کد، MYINI.INI را ایجاد می‌کند. اگر نگاهی به پوشه Windows بیندازیم خواهیم دید که فایل ایجاد شده است. اگر دوبار روی آن کلیک کنید، در Notepad بارگذاری خواهد شد (چون یک فایل متنی است) و باید شبیه کد زیر باشد:

```
[Parameters]
Path=C:\temp\
```

فراخوان بعدی API، مسیر را دوباره می‌خواند. البته پیش از آن‌که این امر رخ دهد باید متغیری برای آن ایجاد کند تا مسیر در آن‌جا ذخیره شود.
دستور $Space برای ایجاد رشته‌ای به طول ۲۵۶ کاراکتر استفاده شده است. هم‌چنین این متغیر باید از نوع string تعریف شده باشد در غیر این صورت خطا رخ خواهد داد. مقدار برگشتی به صورت یک رشته خواهد بود و سپس آن را باید در یک متغیر string قرار دهیم.
GetPrivateProfileString مانند WritePrivateProfileString عمل می‌کند اما پارامترهای بیشتری به آن انتقال می‌یابد. پارامترهای اول و دوم مانند قبل هستند و جزییات بخش و کلیدی را که قرار است خوانده شود، مشخص می‌کنند. پارامتر سوم یک مقدار پیش‌فرض دارد تا اگر مورد مربوطه پیدا نشود آن مقدار را برگرداند. پارامتر چهارم حاوی نام متغیر برای نتیجه‌ای است که قرار است انتقال یابد. پارامتر پنجم حاوی حداکثر تعداد کاراکترهایی است که در متغیر قرار می‌گیرند. پارامتر ششم، نام فایلی است که به دنبال آن هستیم.
متغیری که API(x) را فرامی‌خواند، تعداد کاراکترهای برگشتی را مشخص می‌کند. S$ حاوی کلیدی است که با یک کاراکتر Null تمام می‌شود. اگر نام کلید را Null فرض کنیم، تمام ورودی‌ها به آن بخش که با کاراکتر Null پایان می‌یابند به عنوان خروجی، برمی‌گردند.
هر دوی این فراخوان‌های API، در عملیات خود بسیار سهل انگار هستند. اگر فایل در آن‌جا وجود نداشته باشد، آن را ایجاد می‌کنند. اگر کلیدی وجود نداشته باشد نیز ایجاد می شود. وقتی از دستور خواندن استفاده می‌کنیم اگر فایل وجود نداشته باشد آن‌گاه مقدار Null برمی‌گردد.
این متد، اغلب در برنامه‌ها به کار گرفته می‌شود تا مسیر تنظیمات کاربر را حفظ کند به طوری که وقتی از آن پس کاربر وارد برنامه شد، بر طبق تنظیمات شخصی کاربر تنظیم شود.

مایکروسافت دیگر فایل‌های INI را به کار نمی‌برد و در عوض از کلیدهای موجود در Registry برای ضبط این اطلاعات استفاده می‌کند.

خواندن فعالیت صفحه کلید

هدف دیگر و مفید فراخوان‌های API، خواندن صفحه کلید و پیدا کردن این است که آیا کلید خاصی فشرده شده است یا خیر. رویدادهایی روی فرم‌های کاربری وجود دارند تا رویدادهای صفحه کلید را مدیریت کنند اما آن‌ها فقط روی یک فرم خاص یا کنترل خاصی روی فرم که در آن لحظه، انتخاب شده است (got focus) عمل می‌کنند.

برای مثال فرض کنید یک ماکرو نوشته‌اید تا یک کار پرزحمت و طولانی را اجرا کند که شامل اجرای حلقه‌ای است که هزاران مرتبه اجرا می‌شود. ممکن است بخواهید به کاربر اجازه دهید تا در صورت طولانی شدن عملیات از برنامه خارج شود. می‌توانید این کار را فقط با استفاده از کنترل صفحه کلید انجام دهید چون تنها روشی که به‌وسیله آن می‌توان یک رویداد صفحه کلید را دید، روی UserForm است؛ اما هنگامی که کاربر دکمه "خروج" را می فشارد UserForm از حالت انتخاب (focus) خارج شده و این امر با مشکل مواجه می‌شود. می‌توانید از فراخوان API با نام GetKeystate برای انجام این کار استفاده کنید. باید کد زیر را در قسمت declaration یک ماژول قرار دهید:

```
Private Declare Function GetKeyState Lib "user32" (ByVal nVirtKey As Long) As Integer
```

این کد به شما اجازه می‌دهد تا هر کلیدی را در هنگام اجرای برنامه، آزمایش کنید. حتی می‌توانید بین کلیدهای SHIFT یا CTRL چپ و راست تفاوت قایل شوید:

```
Sub Test_Key ( )
x = 0
Do until x = 1
        If GetKeyState(&H9) < 0 Then x = 1
        DoEvents
Loop
MsgBox "You pressed the TAB key"
End Sub
```


این برنامه، یک حلقه ساده Do Until..Loop را راه‌اندازی می‌کند و تا زمانی که x=1 شود به اجرای خود ادامه می‌دهد . این امر هرگز رخ نمی‌دهد تا وقتی که خط GetKeystate، یک مقدار برای کلید مشخص شده‌ای را که کمتر از صفر است، برگرداند (یعنی این کلید فشار داده شود). برای این مثال، کلید TAB به کار گرفته شده است که مقدار آن در قالب هگزادسیمال برابر 09 است. وقتی کلید TAB فشرده شود، x مقدار خود را به 1 تغییر داده و اجرای حلقه متوقف می شود و سپس کادر پیغام به نمایش در می‌آید.

در این‌جا دستور Do Events بسیار مهم است چون به سیستم عامل اجازه می‌دهد پیش از آن‌که به سراغ دستورالعمل بعدی برود، پردازش نام پیغام‌ها را تمام کند. اگر از این دستور استفاده نکنیم، پیغام صفحه کلید پیش از شروع حلقه بعدی، پردازش نخواهد شد و کلید فشرده شده، حذف شده و در عملیات شرکت داده نخواهد شد.

این کد را اجرا کنید تا در حلقه‌ای بی‌پایان گرفتار شوید چون مقدار متغیر x هرگز برابر 1 نخواهد شد. هر کلیدی را که روی صفحه کلید فشار دهید تأثیری نخواهد داشت و اتفاقی نخواهد افتاد. اما اگر کلید TAB را فشار دهیم، برنامه به پایان می‌رسد و یک کادر پیغام به نمایش درمی‌آید. البته اگر بخواهیم از دیگر ترکیب‌های کلید استفاده کنیم، کمک می‌کند تا مقادیر Virtual Key Codes را به دست آوریم. فهرست آن‌ها را در زیر می بینید:

جدول ۱-۱۷

نام ثابت نمادی	مقدار (هگزادسیمال)	معادل ماوس یا صفحه کلید
VK_LBUTTON	01	دکمه چپ ماوس
VK_RBUTTON	02	دکمه راست ماوس
VK_CANCEL	03	CTRL – BREAK
VK_MBUTTON	04	دکمه وسط ماوس
-	05-07	تعریف نشده
VK_BACK	08	BACKSPACE
VK_TAB	09	TAB
-	0A-0B	تعریف نشده
VK_CLEAR	0C	CLEAR
VK_RETURN	0D	ENTER
-	0E-0F	تعریف نشده
VK_SHIFT	10	SHIFT
VK_CONTROL	11	CTRL
VK_MENU	12	ALT

نام ثابت نمادی	مقدار (هگزادسیمال)	معادل ماوس یا صفحه کلید
VK_PAUSE	13	PAUSE
VK_CAPITAL	14	CAPS LOCK
-	15-19	برای سیستم های KANJI
-	1A	تعریف نشده
VK_ESCAPE	1B	ESC
-	1C-1F	برای سیستم های KANJI
VK_SPACE	20	SPACEBAR
VK_PRIOR	21	PAGE UP
VK_NEXT	22	PAGE DOWN
VK_END	23	END
VK_HOME	24	HOME
VK_LEFT	25	LEFT ARROW
VK_UP	26	UP ARROW
VK_RIGHT	27	RIGHT ARROW
VK_DOWN	28	DOWN ARROW
VK_SELECT	29	SELECT
-	2A	تعریف نشده
VK_EXECUTE	2B	EXECUTE
VK_SNAPSHOT	2C	PRINT SCREN
VK_INSERT	2D	INS
VK_DELETE	2E	DEL
VK_HELP	2F	HELP
VK_0	30	0
VK_1	31	1
VK_2	32	2
VK_3	33	3
VK_4	34	4
VK_5	35	5
VK_6	36	6
VK_7	37	7
VK_8	38	8
VK_9	39	9
-	3A-40	تعریف نشده
VK_A	41	A
VK_B	42	B
VK_C	43	C

نام ثابت نمادی	مقدار (هگزادسیمال)	معادل ماوس یا صفحه کلید
VK_D	44	D
VK_E	45	E
VK_F	46	F
VK_G	47	G
VK_H	48	H
VK_I	49	I
VK_J	4A	J
VK_K	4B	K
VK_L	4C	L
VK_M	4D	M
VK_N	4E	N
VK_O	4F	O
VK_P	50	P
VK_Q	51	Q
VK_R	52	R
VK_S	53	S
VK_T	54	T
VK_U	55	U
VK_V	56	V
VK_W	57	W
VK_X	58	X
VK_Y	59	Y
VK_Z	5A	Z
VK_LWIN	5B	LEFT WINDOWS
VK_RWIN	5C	RIGHT WINDOWS
VK_APPS	5D	APPLICATIONS
-	5E-5F	تعریف نشده
VK_NUMPAD0	60	Numeric keypad 0
VK_NUMPAD1	61	Numeric keypad 1
VK_NUMPAD2	62	Numeric keypad 2
VK_NUMPAD3	63	Numeric keypad 3
VK_NUMPAD4	64	Numeric keypad 4
VK_NUMPAD5	65	Numeric keypad 5
VK_NUMPAD6	66	Numeric keypad 6
VK_NUMPAD7	67	Numeric keypad 7
VK_NUMPAD8	68	Numeric keypad 8
VK_NUMPAD9	69	Numeric keypad 9
VK_MULTIPLY	6A	MULTIPLY

نام ثابت نمادی	مقدار (هگزادسیمال)	معادل ماوس یا صفحه کلید
VK_ADD	6B	ADD
VK_SEPARATOR	6C	SEPARATOR
VK_SUBTRACT	6D	SUBSTRACT
VK_DECIMAL	6E	DECIMAL
VK_DIVIDE	6F	DIVIDE
VK_F1	70	F1
VK_F2	71	F2
VK_F3	72	F3
VK_F4	73	F4
VK_F5	74	F5
VK_F6	75	F6
VK_F7	76	F7
VK_F8	77	F8
VK_F9	78	F9
VK_F10	79	F10
VK_F11	7A	F11
VK_F12	7B	F12
VK_F13	7C	F13
VK_F14	7D	F14
VK_F15	7E	F15
VK_F16	7F	F16
VK_F17	80H	F17
VK_F18	81H	F18
VK_F19	82H	F19
VK_F20	83H	F20
VK_F21	84H	F21
VK_F22	85H	F22
VK_F23	86H	F23
VK_F24	87H	F24
-	88-8F	تعریف نشده
VK_NUMLOCK	90	NUM LOCK
VK_SCROLL	91	SCROLL LOCK
VK_LSHIFT	A0	KEFT SHIFT
VK_RSHIFT	A1	RIGHT SHIFT
VK_LCONTROL	A2	LEFT CTRL
VK_RCONTROL	A3	RIGHT CTRL
VK_LMENU	A4	LEFT MENU
VK_RMENU	A5	RIGHT MENU

نام ثابت نمادی	مقدار (هگزادسیمال)	معادل ماوس یا صفحه کلید
-	E7-E8	تعریف نشده
-	E9-F5	مخصوص OEM
VK_ATTN	F6	ATTN
VK_CRSEL	F7	CRSEL
VK_EXSEL	F8	EXSEL
VK_EREOF	F9	ERASE EOF
VK_PLAY	FA	PLAY
VK_ZOOM	FB	ZOOM
VK_NONAME	FC	ذخیره شده برای آینده
VK_PA1	FD	PA1
VK_OEM_CLEAR	FE	CLEAR

اجرای صداهای مالتی مدیا

می‌توانیم از یک فراخوان API برای اجرای یک صدا در کد خود از طریق یک فایل WAV استفاده کنیم. زبان برنامه‌نویسی ماکروی قدیمی یک دستور برای انجام این کار دارد اما در VBA اضافه نشده است. اگر بخواهیم از توابع Multimedia در کد خود استفاده کنیم تا فایل‌های صوتی اجرا شوند، آن‌گاه به ابزاری برای اجرای فایل‌های صوتی نیاز خواهیم داشت. اعلان این دستور، در بخش declaration یک ماژول قرار می‌گیرد:

```
Public Declare Function PlaySound Lib "winmm.dll" Alias "PlaySoundA" _
(ByVal  lpszName As String, ByVal  hModule As Long, ByVal dwFlags As _
Long) As Long
```

سپس می‌توانید فایل‌های WAV را در کد خود اجرا کنید:

```
Sub test_sound ( )
x = PlaySound("c:\windows\media\chord.wav", 0, 2)
x = PlaySound("c:\windows\media\ding.wav", 0, 2)
End Sub
```

این مثال، دو صدای استاندارد ویندوز را اجرا می‌کند. اگر روی کامپیوتر یک میکروفن داشته باشیم می‌توانیم صدای خود را در یک فایل WAV ذخیره کرده و بعداً آن‌ها را به همین روش اجرا کنیم.
این مثال‌ها، قدرت فراخوان‌های API را در یک برنامه نشان می‌دهند. اگر به این قضیه زیاد علاقه دارید پیشنهاد می‌کنم به کتاب‌هایی که در این باره نوشته شده‌اند مراجعه کنید.

فصل هجدهم

ماژول‌های کلاس

به موازات برنامه‌نویسی در VBA و درج ماژول‌ها برای نگهداری از کد شما، از منو متوجه خواهید شد که می‌توانید آنچه را که ماژول‌های کلاس نامیده می‌شود، درج کنید. این ماژول‌ها متفاوت از ماژول‌های معمولی هستند چون به شما اجازه می‌دهند به وسیله ایجاد یک برنامه الحاقی، یک مدل شئ مؤلفه‌ای (COM) ایجاد کنید.
ماژول‌های کلاس را نمی‌توان مانند یک رویه ماژول استاندارد اجرا کرد و باید از داخل یکی از ماژول‌های موجود در کد به آن‌ها ارجاع کرد. این امر به شما اجازه می‌دهد تا مجموعه‌ها و اشیا خاص خود را ایجاد کنید (مانند مجموعه Workbooks یا Worksheets). متأسفانه، نمی‌توانیم یک فایل DLL (کتابخانه پیوند دینامیکی) ایجاد کنیم (وقتی در حال برنامه‌نویسی در ویژوال بیسیک یا C هستیم باید این‌کار انجام شود). البته یک ماژول کلاس می‌تواند به یک برنامه الحاقی تبدیل شود (برنامه الحاقی، یک شئ مؤلفه‌ای است) و این‌کار به طور مؤثری می‌تواند معماری چند لایه‌ای به برنامه کاربردی ما اضافه کند. برنامه الحاقی، مؤلفه‌ای است که می‌تواند توزیع شود و مستقل از یک صفحه گسترده خاص به‌کار گرفته شود.
در اوایل این کتاب یاد گرفتیم کهExcel ، یک برنامه چند لایه‌ای است چون یک لایه خدمات سرویس گیرنده وجود دارد که در زیر آن، مدل شئ Excel واقع شده است و زیر آن نیز، لایه خدمات داده‌ای واقع است. ماژول‌های کلاس به ما اجازه می‌دهند تا لایه دیگری بین خدمات سرویس گیرنده و مدل شئ Excel یا بین لایه خدمات سرویس گیرنده و یک منبع داده خارجی (مانند Access یا SQL Server) قرار دهیم.
می‌توانیم برنامه کاربردی را به یک برنامه الحاقی (add in) تبدیل کرده و سپس از آن به عنوان مرجع برای شئ خود استفاده کنیم. البته دیگر برنامه‌هایی که از آن برنامه استفاده می‌کنند،کد زیر آن و قوانینی را که در آن تعبیه کرده‌اید، نخواهید دید. به محض آن‌که برنامه الحاقی که ایجاد کرده‌ایم بارگذاری شود، می‌توان از دیگر ماژول‌های موجود در صفحه گسترده به توابع عمومی و زیر روال‌های مربوطه دسترسی پیدا کرد هرچند اگر از کلمه عبور برای محافظت استفاده کرده باشید، دیگر برنامه‌نویسان نمی‌توانند کد مربوطه را ببینند. برای دیدن جزییات بیشتر در مورد ایجاد یک برنامه الحاقی به فصل ۴۱ مراجعه کنید.

به عنوان مثال، کارپوشه یک شئ است. وقتی یک کارپوشه ذخیره می‌شود می‌توانیم یک کلمه عبور برای امنیت آن به کد اضافه کنیم و وقتی کارپوشه مجدداً باز می‌شود باید برای موفقیت در اجرای کد حتماً کلمه عبور وارد شود. حتی یک خصوصیت با نام HasPassword هم روی شئ Workbook وجود دارد اما آن‌چه در اختیار ما قرار نمی‌گیرد این است که خصوصیتی نداریم که حاوی خود کلمه عبور باشد. البته تمام کدهایی که برای پنهان‌سازی و کشف کلمات عبور به کار می‌رود در مدل شئ Excel پنهان شده است اما هیچ متد یا خصوصیتی برای دستیابی به کلمه عبور واقعی ارایه نشده است.

طراحان ماژول شئ Excel قوانین مکتوبی دارند که ابراز می‌دارد: " می‌توانید یک فایل را با یک کلمه عبور ذخیره کنید اما نمی‌توانید آن کلمه عبور را ببینید." به همین نحو می‌توانیم اشیا اختصاصی طراحی کنیم و قوانینی وضع نماییم که درباره کار با این اشیا باشد (این‌که کدام خصوصیت‌ها یا متدها در آن وجود داشته باشند و این‌که آیا مجموعه می‌تواند تغییر کند یا فقط خواندنی (read-only) است).

در مثال‌های زیر یک شئ می‌سازیم که از مجموعه چندین نام از سلول‌های صفحه گسترده گرفته شده است. مجموعه شئ را PNames خواهیم نامید که مجموعه‌ای از اشیا PName خواهد بود. نام‌ها می‌توانند نام افراد یا اماکن باشند.

درج یک ماژول کلاس

از منوی کد، Insert | Class Module را انتخاب کنید تا یک ماژول کلاس با نام Class1 ایجاد شود. لازم است بلافاصله نام را تغییر دهید چون این نام، نام شئ شما بوده و Class1 یک نام بی‌معنا است چون چیزی از عملکرد آن کلاس را نشان نمی‌دهد. از این نام در کد برای ارجاع به شئ استفاده می‌کنیم و اگر بخواهیم به Class1 ارجاع نماییم کمی گیج‌کننده خواهد بود.

نام را در پنجره Properties تغییر می‌دهیم و باید توجه کنید که ماژول‌های کلاس تنها یک خصوصیت name دارند. فیلد نام را انتخاب کرده و عبارت PName را در آن بنویسید. این نام نشان دهنده اشیا موجود در مجموعه است. دوبار روی ماژول کلاس در نمودار درختی کلیک کنید تا به نمایش درآید.

ایجاد یک شئ

لازم است ابتدا شئ واقعی برای اقلام موجود در مجموعه را بسازیم چون این شئ در ساختار شئ Collections به کار گرفته خواهد شد. شئ PName دارای یک خصوصیت است: نام شئ که PName نامیده می‌شود. به یاد دارید که این چه خصوصیتی است؟ این خصوصیت، مقدار چیزی در آن شئ را در خود نگه می‌دارد که در این حالت، نام واقعی است.

ابتدا لازم است متغیری ایجاد شود تا مقدار خصوصیت را در خود نگه دارد و این کار را در بخش declarations ماژول انجام می‌دهیم:

```
Private mPname As String
```

این کد باعث ایجاد یک متغیر رشته‌ای با نام mPname می‌شود. توجه کنید این متغیر، خصوصی (Private) است و به عنوان قسمتی از شئ دیده نمی‌شود. این امر بدان خاطر است که یک متغیر کاری در ماژول کلاس وجود دارد که برای استفاده توسط کاربر نیست. هم‌چنین نام این متغیر باید از نام واقعی خصوصیت که کاربر می‌تواند ببیند متفاوت باشد. در این مورد، حرف m به جلوی نام اضافه شده است تا با هم فرق داشته باشند.

سپس لازم است کدی برای کنترل دستیابی به خصوصیت ایجاد کنیم. ما به یک دستور Property Get و Property Let نیاز داریم و دستورها باید عمومی باشند، در غیر این صورت نمی‌توانیم آن‌ها را در شئ ببینیم. این دستورها، مقادیر را در خصوصیت نوشته و از آن می‌خوانند:

```
Public Property Let Pname(vdata As String)
        mPname = vdata
End Property
```

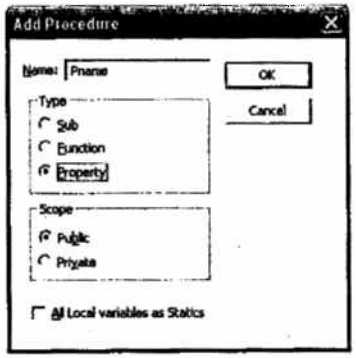
پارامتر vdata نشان دهنده مقداری است که در دستور Let به خصوصیت انتقال می‌یابد.

```
Public Property Get Pname( ) As string
        Pname = mPname
End Property
```

بسیار مهم است که دستورهای خصوصیت Get و Let نام مشابه داشته باشند در غیر این صورت می‌توانیم یک خصوصیت را بخوانیم (Get) اما نمی‌توانیم چیزی در آن بنویسیم (Let) یا این‌که می‌توانیم در آن بنویسیم (Let) اما نمی‌توانیم از آن بخوانیم (Get).

خصوصیت Let از یک متغیر رشته‌ای برای انتقال داده‌ها به متغیر mPname استفاده می‌کند. خصوصیت Get، مقدار آن خصوصیت را برابر مقداری که در متغیر mPname وجود دارد، قرار می‌دهد.

می‌توانیم از منوی کد، Insert | Procedure را انتخاب کنیم تا اسکلت بندی کد برای یک خصوصیت ایجاد شود. این‌کار باعث باز شدن کادر محاوره‌ایAdd Procedure می‌شود که در تصویر ۱-۱۸ نمایش داده شده است. این به طور خودکار، دستورات خصوصیت Get و Let را می‌نویسد و ما را مطمئن می‌کند که هر دوی‌آن‌ها یک نام دارند.



تصویر ۱-۱۸ استفاده از کادر محاوره‌ای Add Procedure برای وارد کردن یک خصوصیت

اکنون یک شئ با نام Pname راه اندازی کرده‌ایم. این شئ در مجموعه‌ای با نام PNames قرار می‌گیرد (مانند شئ Worksheet که در مجموعه‌ای با نام Worksheets واقع است).

ایجاد یک Collection

مرحله بعدی، ایجاد شئ Collections است تا اشیای مستقل PName را در خود نگه دارد. از منوی VBE، با استفاده از Insert | Class Module ، یک ماژول کلاس دیگر را درج کنید. نام آن را به PNames تغییر داده و دوبار روی ماژول کلاس کلیک کنید تا وارد آن شوید.
نخستین کاری که باید انجام دهیم، تعریف یک متغیر خصوصی است تا به عنوان شئ Collection عمل کند و لازم است این‌کار در بخش declarations ماژول کلاس انجام شود:

```
Private mPnames As Collection
```

این متغیر باید یک نام اختصاصی و منحصر به فرد داشته باشد. سپس یک حرف m جلوی نام مجموعه قرار داده شده است. حرف m، متغیر را به عنوان یک متغیر عضوی (member) معرفی می‌کند.
سپس زیرروال Class_Initialize را انتخاب می‌کنیم. این روال وقتی اجرا می‌شود که برای بار نخست از کلاس استفاده می‌کنیم و مشخص می‌کند که محتویات آن چه باشد. برای اجرای آن، روی بخش Class در منوی بازشوی بالا- سمت چپ کلیک کنید و در منوی بازشوی بالا- سمت راست روی Initialize کلیک کرده و این کد را وارد کنید:


```
Private Sub Class_Initialize()

Dim objPname As Pname

Set mPnames = New Collection
For n = 1 To 3

        Set objPname = New Pname

        objPname.Pname = Worksheets("sheet1") .Range("a" & n + 1) .Value

        mPnames.Add objPname
Next n
End Sub
```

ابتدا یک شئ با نام objPname را بر مبنای شئ Pname که قبلاً ایجاد کرده‌ایم تعریف می‌کنیم. توجه کنید که در حال حاضر، اشیا PName و PNames به موازات تایپ کد، در یک لیست بازشو نمایش داده می‌شوند. سپس متغیر محلی mPNames را به عنوان شئ جدید Collection در نظر می‌گیریم. در این مرحله، مقدار آن متغیر تهی است.
حلقه For..Next، اشیا را از یک منبع داده به مجموعه اضافه می‌کند. این منبع می‌تواند یک پایگاه داده خارجی باشد یا تنها یک آرایه محلی باشد که اطلاعات را در خود جای می‌دهد. در این مثال، قسمتی از صفحه گسترده به کار گرفته شده است هرچند در این مرحله، هیچ داده‌ای وجود ندارد.
متغیر objPname که به صورت شئ PName تعریف شده بود، برای هر حلقه به صورت یک Pname جدید تنظیم می‌شود و این به معنای ایجاد یک مورد جدید از PName است که بتواند داده‌ها را نگه دارد و به مجموعه PNames اضافه شود. سپس داده‌ها از ستون A در Sheet1 گرفته شده و در خصوصیت Pname ایجاد شده، قرار می‌گیرند.
سپس شئ objPname به مجموعه Pnames اضافه می‌شود. در انتهای این حلقه For..Next، سه شئ به مجموعه PNames اضافه شده‌اند که این‌کار بر مبنای داده‌های موجود در Sheet1 انجام شده است.
هم‌چنین لازم است یک تابع عمومی با نام Item را هم اضافه کنیم به طوری که بتوانیم توسط شماره شاخص اشیا مستقل، به آن‌ها ارجاع کنیم. این تابع را مانند بقیه کد، در همان ماژول کلاس وارد کنید:

```
Public Function Item(Index As Integer) As Pname
        Set Item = mPnames.Item(Index)

End Function
```

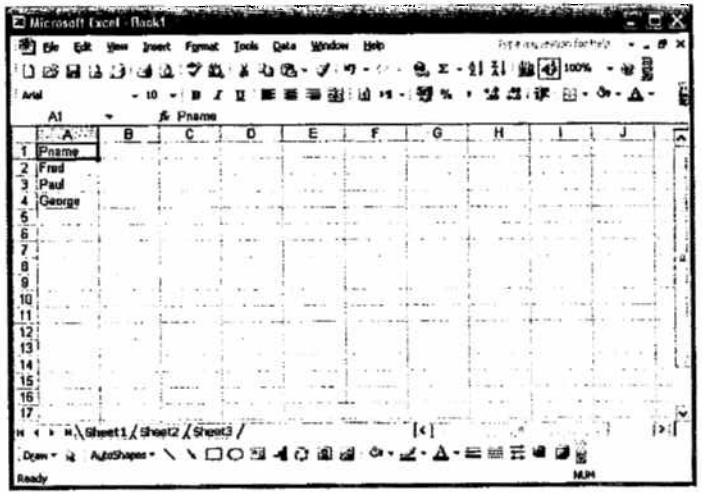
این قسمت از کد، Item را به صورت تابعی از شئ Pnames تعریف می‌کند و Item را بر مبنای شماره شاخص ارایه شده طوری تنظیم می‌کند تا به مجموعه mPnames اشاره کند. چون mPnames به صورت یک شئ Collection تعریف شده، از قبل تابع Item در آن تعبیه شده است. این تابع مانند آیینه، آن‌چه را که در مجموعه mPnames انجام می‌شود، منعکس می‌کند.

لازم است یک خصوصیت با نام Count را اضافه کنیم به طوری که کد ما بتواند مشخص کند چه تعداد شئ Pname در مجموعه وجود دارد:

```
Public Property Get Count( ) As Long
    Count = mPnames.Count
End Property
```

در این مورد ما تنها به خصوصیت Get نیاز داریم چون این یک خصوصیت فقط خواندنی (read-only) است و توسط روال Class_Initializeکه در داده‌های منبع اضافه می‌شود، به‌روز شده است.

اکنون باید مجموعه‌ای با نام PNames متشکل از اشیایی با نام PName را ایجاد کنیم که به بلوکی از داده‌های صفحه گسترده در ستون A از Sheet1 استناد می‌کند (تصویر ۲-۱۸). مجموعه دارای یک خصوصیت Count و یک متد Item است.



تصویر ۲-۱۸ داده‌های به‌کار گرفته شده برای مجموعه Pname

به‌کارگیری مجموعه PNames

حال می‌توانیم از PNames متعلق به شئ Collection مانند هر شئ دیگری در کد استاندارد خود استفاده کنیم. ما از کد در یک ماژول استاندارد Excel VBA استفاده می‌کنیم:

```
Sub test_class( )
Dim pn  As New Pnames, n As Integer
MsgBox pn.Count
For n = 1 To pn.Count

        MsgBox pn.Item(n). Pname
Next n
End Sub
```

این کد باعث ایجاد یک مورد از شئ Pnames می‌شود. وقتی این رویه اجرا شود،اولین کاری که انجام می‌دهد، راه اندازی (initialize) ماژول کلاس است که به معنای برداشتن داده‌ها از صفحه گسترده و اضافه کردن اشیا به مجموعه است.

متغیر n که در حلقه For...Next به‌کار گرفته شده از نوع integer (عدد صحیح) تعریف شده است چون در تابع Item مجموعه، این متغیر به صورت integer(index) تعریف شده است. به این دلیل از عدد صحیح استفاده نشده است که به راحتی می‌توان از آن به عنوان متغیر حلقه For..Next استفاده کرد. اگر از چنین متغیری استفاده نکنیم در هنگام اجرای کد با خطای Type Mismatch مواجه می‌شویم.

سپس یک حلقه For..Next را راه اندازی می‌کنیم تا روی تک تک اشیا موجود در مجموعه اجرا شود. این امر بر مبنای تابع Count در شئ انجام می‌شود. توجه کنید که به موازات تایپ کد، تابع Item و خصوصیت Count مانند دیگر اشیا داخلی در کادرهای لیستی ظاهر می‌شوند.

اکنون با استفاده از شئ Collection می‌توانیم از تابع Item متغیر n برای ارجاع به اشیا موجود در مجموعه استفاده کنیم. خصوصیت Pname برای نمایش هر کدام از نام‌ها به کار گرفته می‌شود.

کد را در ماژول اجرا کنید تا یک کادر پیغام که مقدار 3 در آن نمایش داده شده است به دست آورید (به معنای سه شئ در مجموعه) که نام هر شئ به نوبت ذکر شده است. سپس مقدار یک سلول را روی صفحه گسترده تغییر دهید و دوباره کد را اجرا کنید. نام موجود در مجموعه نیز تغییر می‌کند چون وقتی شئ Pnames یا (pn) را در خط نخست کد ایجاد می‌کنیم، این شئ مجدداً راه‌اندازی شده و

مقادیر جدیدی می‌گیرد. اگر Collection را به صورت Static ایجاد کنیم، دوباره راه‌اندازی نخواهد شد و مقادیر قدیمی مجموعه را نگه خواهد داشت:

```
Sub test_class( )
Static pn  As New Pnames, n As Integer
MsgBox pn.Count
For n = 1 To pn.Count

        MsgBox pn.Item(n). Pname
Next n
End Sub
```

فصل نوزدهم

انیمیشن

می‌توانیم اشیا را در Excel به صورت متحرک دربیاوریم. حتی اگر حرفه‌ای عمل کنیم می‌توانیم یک بازی نیز در Excel طراحی کنیم چون تمام ابزارهای انجام این‌کار را در اختیار داریم. اما در این کتاب فقط یک شئ Image (تصویر) را روی صفحه جابه‌جا می‌کنیم که به صورت هم‌زمان، رنگ آن نیز تغییر می‌یابد.

پیش از هر چیز لازم است که یک کنترل Image روی صفحه گسترده ترسیم کنیم. این‌کار را با انتخاب View | Toolbars | Control Toolbox برای باز کردن پنجره Toolbox انجام می‌دهیم. با کلیک روی کنترل Image در جعبه ابزار، آن را انتخاب می‌کنیم. نام آن باید Image1 باشد.

یک دکمه فرمان نیز انتخاب کنید و آن را روی صفحه گسترده بکشید. ویژگی Caption در پنجره Properties را به My Animation تغییر دهید. صفحه گسترده باید شبیه تصویر ۱-۱۹ باشد.

تصویر ۱-۱۹ ایجاد یک شئ متحرک و یک دکمه فرمان

روی دکمه فرمان دابل کلیک کنید تا وارد رویه کدنویس برای رویداد Click دکمه فرمان شوید. این کد را اضافه کنید:

```
Private Sub CommandButton1_Click ( )
again:
With VBAProject.Sheet1.Image1
Randomize
.BackColor = QBColor(Int((Rnd * 15) + 1))
.Top = .Top + 2
.Left = .Left + 2
If .Top > 100 Then .Top = 1
If .Left > 100 Then .Left = 1
End with
GoTo again
End Sub
```

توجه کنید که خط اول کد، یک برچسب با نام again ایجاد می‌کند. این باعث ایجاد حلقه‌ای بر مبنای برچسب می‌شود تا روال به صورت مستمر اجرا شود. یک دستور With برای شئ VBAProject.Sheet1.Image1 به کار گرفته شده است تا مجبور نباشیم هر بار که می‌خواهیم به تصویر دسترسی داشته باشیم کد نویسی کنیم.

دستور Randomize باعث ایجاد اعداد تصادفی می‌شود. عدد تصادفی، عددی است که بدون هیچ الگوی از پیش تعیین شده‌ای انتخاب می‌شود.

رنگ پس زمینه کادر تصویر با استفاده از رنگ‌های Q-Basic تنظیم می‌شود. بد نیست بدانید که در Q-Basic فقط امکان نمایش 16 رنگ روی صفحه وجود داشت و این امر هنوز هم برای تغییر رنگ با استفاده اعداد تصادفی، کار مناسبی است. تابع Rnd، اعداد تصادفی بین صفر و 1 ایجاد می‌کند. عدد ایجاد شده در 15 ضرب می‌شود تا یک عدد صحیح حاصل شود و امکان در برگرفتن مجموعه تمام 16 رنگ حاصل شود. چون تابع INT تمام اعداد را با حذف قسمت اعشاری، گرد می‌کند، عدد یک به نتیجه اضافه می‌شود. این دستور باعث می‌شود کنترل Image به موازات حرکت روی صفحه نمایش،رنگ‌های تصادفی، نمایش دهد.

پس 2 واحد به مقدار ویژگی Top اضافه می‌شود تا باعث شود شئ به سمت پایین صفحه نمایش حرکت کند. 2 واحد هم به ویژگی Left اضافه می‌شود تا باعث شود شئ به سمت چپ صفحه نمایش حرکت کند. اگر شئ، درهر جهت (Left یا Top) به سمت مقدار 100 واحد جابه‌جا شود، مقدار آن

مجدداً برابر با 1 تنظیم می‌شود و حرکت مجدداً از سر گرفته می‌شود و به صورت مورب به سمت راست و پایین حرکت می‌کند . بلوک With خاتمه می‌پذیرد و برنامه از خط again دوباره اجرا می‌شود.

مطمئن شوید که در گوشه سمت چپ- بالای پنجره Toolbox روی آیکن Design Mode کلیک کرده‌اید تا دکمه فرمان از حالت طراحی خارج شود و به صورت فعال دربیاید. این کد را با کلیک روی دکمه My Animation روی صفحه گسترده اجرا کنید تا ببینید که کنترل Image به سمت راست و رو به پایین حرکت می‌کند و همزمان به صورت تصادفی، رنگ آن نیز تغییر می‌کند. برای توقف اجرای کد، دکمه CTRL_BREAK را فشار دهید.

با تغییر رنگ پس زمینه یک سلول نیز می‌توانیم انیمیشن ساده‌ای ایجاد کنیم:

```
Sub cell_animation( )
again1:
Dim w As Worksheet
Set w = worksheets("sheet1")
Randomize
For n = 65 To 69
  For m = 1 To 5

w.Range(Chr(n) & m).Rows.Interior.Color =  QBColor(Int((Rnd * 15) + 1))
DoEvents

  Next m

Next n
GoTo again1
End Sub
```

این کد به صورت مستمر حول یک برچسب با نام again1 اجرا می‌شود چون اگر این رویه در همان ماژول مثال قبلی قرار گیرد نمی‌توانیم از برچسب again استفاده کنیم و باید نام آن را تغییر دهیم.

شئ نخست (w) به عنوان یک کاربرگ ساخته شده است که به Sheet1 اشاره می‌کند. سپس دستور Randomize نیز مثل قبل، اعداد تصادفی ایجاد می‌کند.

سپس دو حلقه تودرتو قرار گرفته‌اند. حلقه For..Next که با متغیر n مشخص شده است از 65 تا 69 (کد اسکی A تا E که نشان دهنده ستون‌ها هستند) اجرا می‌شود. چون می‌خواهیم حروف از طریق کد VBA تغییر یابند (یعنی از A تا E) پس لازم است در عوض خود کاراکترها، از مقدار عددی آن‌ها استفاده کنیم. حلقه For..Next که با m مشخص می‌شود از 1 تا 5 (معرف ردیف‌ها) اجرا می‌شود. با تغییر مقدار n به یک کاراکتر و الحاق آن به مقدار m (که هر بار تغییر می‌کند) می‌توانیم آدرس هر 25

سلول را در ساختار حلقه‌ای به دست آوریم (یعنی A1 تا E5). دستور DoEvents نیز اضافه شده است تا اطمینان حاصل شود که پیش از رفتن کد به خط بعدی، دستور قبلی اجرا شده است. در فصل 17 از DoEvents در مثالی استفاده کردیم تا کلیدهای فشرده شده روی صفحه کلید را بخواند.

رنگ پس زمینه سلول با توجه به یک عدد تصادفی بین 1 و 16 تنظیم می‌شود و این همان رنگ‌های کوئیک بیسیک است که در مثال قبلی شرح دادیم. درآخر، کد مجدداً به again1 باز می‌گردد و رنگ پس زمینه مجدداً تنظیم می‌شود.

اگر این کد را اجرا کنیم و روی صفحه گسترده کلیک کنیم، شاهد آن خواهیم بود که سلول‌ها در قسمت چپ - بالای صفحه گسترده به صورت پیوسته تغییر رنگ می‌دهند. برای توقف اجرای کد، کلیدهای CTRL_BREAK را فشار دهید.

بخش چهارم

VBA در کاربردهای واقعی

در بخش‌های قبلی درباره تکنیک‌ها و تاکتیک‌های کدنویسی VBA مطالبی آموختیم. اکنون نوبت آن است که برای به‌کارگیری عملی مطالبی که فرا گرفتیم تجربه عملی نیز به‌دست آوریم.

این بخش حاوی چندین مثال عملی کامل درباره چگونگی استفاده از VBA است. این مثال‌ها مطالب خوبی درمورد چگونگی استفاده از کدنویسی در Excel برای ساده‌تر کردن کارها و بالا بردن کارآیی Excel به ما می‌آموزند.

من همیشه اعتقاد داشته‌ام که بهترین روش یادگیری برنامه نویسی، مرور مثال‌ها به طور عملی و دیدن چگونگی عملکرد آن‌هاست. هر چند ممکن است این مثال‌ها دقیقاً آن‌چه را که ما می‌خواهیم انجام ندهند اما می‌توانند مطالب خوبی درباره اصول کار به ما بیاموزند.

در فصل آخر این بخش یعنی فصل ۴۱ یاد خواهیم گرفت که چگونه تمام این مثال‌ها را در یک برنامه الحاقی Excel کنار هم قرار دهیم و آن‌ها را تبدیل به یک برنامه حرفه‌ای و کاربردی نماییم.

فصل بیستم

تبدیل برچسب‌ها به اعداد و اعداد به برچسب‌ها

در مثال‌های این فصل به کاربر اجازه داده می‌شود تا از یک یا چند کاربرگ انتخاب‌هایی را انجام داده و سپس هر جا که لازم باشد، اعداد را به برچسب[1] و برچسب‌ها را به عدد تبدیل کند. اگر عادت کرده باشید که داده‌ها را از دیگر برنامه‌های کاربردی کپی کرده و بچسبانید (Paste) حتماً متوجه شده‌اید که داده‌ها در قالب داده‌ای نادرست چسبانده می‌شوند. مثلاً جایی که ما می‌خواهیم روی اعداد کار کنیم برنامه، آن‌ها را رشته در نظر می‌گیرد. Excel دستوری برای تصحیح این قضیه ندارد. اما برای این کار می‌توانیم از فرمول‌ها استفاده کنیم که مستلزم مقدار زیادی کپی کردن و چسباندن است تا به نتیجه مطلوب دست پیدا کنیم. اما فرآیندی که من در این‌جا نشان خواهم داد به شما اجازه می‌دهد این‌کار را به شیوه بسیار ساده‌ای انجام دهید.

اگر داده‌ها از برنامه دیگری ارسال یا چسبانده می‌شوند در خاتمه کار اغلب به جای آن‌که عدد در نظر گرفته شوند، برچسبی شناخته می‌شوند که از نوع داده‌های رشته هستند. این به معنای آن است که از قالب بندی نرمال عددی تبعیت نمی‌کنند و معمولاً در سلول به صورت چپ چین قرار می‌گیرند. اگر Excel داده‌هایی را برچسب فرض کند آن‌گاه حتی اجرای قالب بندی عددی هم روی آن‌ها جواب نخواهد داد. هم‌چنین اگر در یک سلول، ترکیبی از کاراکترهای عددی و متنی داشته باشیم نیز غیر ممکن است که بتوانیم با فرمول‌ها کاری روی آن سلول انجام دهیم چون سلول همیشه به جای آن‌که یک عدد فرض شود، رشته‌ای از چند کاراکتر تلقی می‌شود.

در کدنویسی، حفاظ‌های خاصی وجود دارند که باید به یاد داشته باشیم. تبدیل تنها در صورتی انجام می‌گیرد که رشته‌ای که قرار است تبدیل شود حاوی مقادیر عددی در خود باشد. هم‌چنین اگر سلولی حاوی یک فرمول باشد نباید تبدیل روی آن انجام شود چون فرمول از بین رفته و نتایج غیرمنتظره‌ای در صفحه گسترده رخ خواهد داد.

1- Label

نکته: هر جا یک برنامه کاربردی یا ماکرو می‌نویسید همیشه پیشاپیش سعی کنید خطاها و مشکلاتی را که ممکن است برای کاربر به‌وجود آیند، در نظر بگیرید. تبدیل یک فرمول کار بسیار خطرناکی است و نتیجه چنین کاری ممکن است کاربران شما را بی‌نهایت خشمگین کند.

در این‌جا یک کد ساده را نشان می‌دهیم:

```
Sub label_to_number ( 0 )

Dim addr As String

For Each window In Windows
        For Each Worksheet In window.SelectedSheets

                For Each cell In Application.Selection

                addr = Worksheet.Name & "!" & cell.Address

                If IsNumeric(Range(addr).Value) = True And Range(addr).Text _
                < >"" And Range(addr).HasFormula = False Then

                        Range(addr) = Val(Range(addr))

                End If

        Next cell

        Next worksheet
Next window
End Sub
```

دقت کنید که حلقه‌ها چگونه به صورت تودرتو درآمده‌اند و این‌که هر حلقه جدید کمی تو رفته‌تر از حلقه قبلی نوشته شده است.

اولین کار، تعریف متغیری است تا آدرس هرسلول در مجموعه سلول‌های انتخاب شده را در خود نگه دارد. این متغیر را addr می‌نامیم. قسمت بعدی کد نیز مجموعه انتخاب شده را مشخص می‌کند.

مشکل اصلی با ماکرو آن است که کاربر ممکن است گزینش را از بین چند کاربرگ انجام داده و توقع داشته باشد که ماکروی شما کل آن گزینش را در نظر بگیرد. اگر این امر را در نظر نگرفته باشیم، ماکرو تنها روی کاربرگ گزینش شده نخست عمل کرده و دیگر بخش‌های گزینش شده را که از کاربرگ‌های دیگر هستند در نظر نمی‌گیرد. کاربر نیز تصور می‌کند ماکرو کل انتخاب او را در نظر

گرفته است چون پیغامی مبنی بر بروز خطا یا این‌که ماکرو فقط کاربرگ نخست را در نظر می‌گیرد، صادر نشده است.

هر چند کاربر در ادامه، متوجه بروز خطا می‌شود اما نمی‌تواند بفهمد چه اتفاقی افتاده است.

بنابراین نه تنها می‌خواهیم انتخاب از کاربرگ فعلی انجام شود بلکه در نظر داریم این امکان به کاربر داده شود تا با کلیک کردن روی زبانه برگه‌های مختلف از بین چندین برگه، انتخاب انجام دهد. حتی ممکن است کاربر بخواهد انتخاب را از بین چند کارپوشه انجام دهد.

این مثال کمی از پیچیدگی‌های موجود را در هنگام طراحی ماکرو به ما نشان می‌دهد. در واقع کارکردن روی یک کاربرگ تکی بسیار ساده است اما اگر گزینش از بین چندین کاربرگ یا کارپوشه انجام شود، چه اتفاقی می افتد؟

شئ Selection روی شئ Application، آدرس واقعی یک سلول از بین سلول‌های انتخاب شده را بر می‌گرداند اما نام برگه‌ای را که سلول از روی آن انتخاب شده است، مشخص نمی‌کند. برای انجام این کار ابتدا باید با استفاده از ساختار حلقه For Each..Next یک بار شئ Windows را مرور کنیم. هر کاربرگ، پنجره‌ای ایجاد می‌کند که قسمتی از مجموعه Windows است. این مجموعه را با سیستم عامل Windows اشتباه نگیرید چون مجموعه‌ای منحصر به فرد برای پنجره‌های موجود در Excel است و نام هر کاربرگ موجود در Excel را بدون توجه به کارپوشه مرتبط با آن برمی‌گرداند. سپس می‌توانیم در یک حلقه، تمام کاربرگ‌هایی را که با استفاده از مجموعه SelectedSheets انتخاب شده‌اند، مرور کنیم.

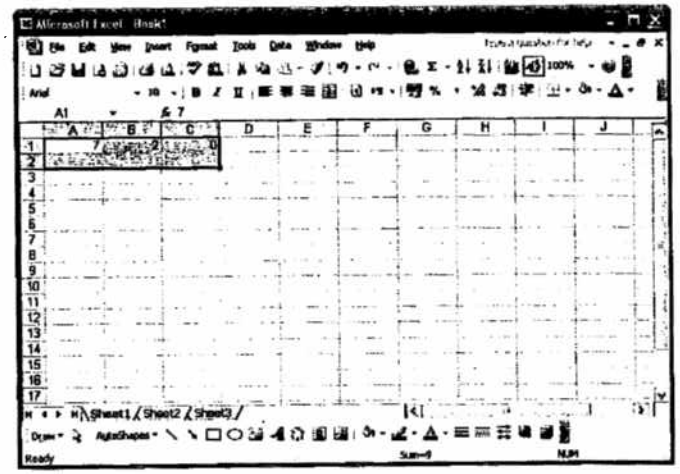
در هر کاربرگ انتخاب شده نگاهی به شئ Selection بیندازیم. این شئ به عنوان انتخابی که کاربر با کشیدن مکان‌نما روی گروهی از سلول‌ها انجام داده تعریف شده است. دقت کنید که با کشیدن مکان‌نما روی سلول‌ها، یک بلوک متمایز (highlight) ایجاد می‌شود. وقتی این کد اجرا می‌شود، شئ Selection حاوی آن مجموعه انتخاب شده خواهد بود.

در شئ Selection چندین سلول وجود دارند. پس مجدداً با استفاده از حلقه For Each..Next تمام سلول‌های انتخاب شده را مرور می‌کنیم. آدرس هر سلول، مختصات آن سلول را به ما می‌گوید اما مشخص نمی‌کند که سلول روی کدام کاربرگ قرار دارد. به این دلیل است که در مرتبه نخست یک‌بار حلقه را روی شئ Windows اجرا کردیم تا نام تمام کاربرگ‌های موجود را به دست آوریم.

سپس آدرس سلول با استفاده از کاراکتر ! به نام کاربرگ چسبانده شده و در متغیر addr ذخیره می‌شود. در این مرحله لازم است تا خصوصیات خاص سلول را کنترل کنیم و برای این‌کار از یک دستور If استفاده می‌کنیم. از متد Range روی متغیر addr برای انجام این کار استفاده کنید.

کد، کنترل می‌کند که آیا مقدار عددی در سلول وجود دارد، سلول خالی نباشد و حاوی فرمول هم نباشد. کد از تابع IsNumeric و ویژگی HasFormula برای این‌کار استفاده می‌کند. برای کنترل خالی نبودن سلول نیز یک عبارت ساده به کار می‌رود. اگر تمام شرایط رضایت بخش باشند آن‌گاه با استفاده از تابع Val مقدار سلول را به یک مقدار عددی تبدیل می‌کنیم.

عبارت End If، شرط If را به پایان برده و دیگر دستورات Next، تا زمان تکمیل گزینش‌ها، تمام مجموعه‌ها را مرور می‌کنند. در این روش، تمام برگه‌های انتخاب شده در عملیات شرکت داده شده و تمام سلول‌های گزینش شده در آن برگه‌ها نیز در حلقه مرور می‌شوند (تصویر ۱-۲۰).



تصویر ۱-۲۰ تغییر برچسب‌ها به اعداد در یک صفحه گسترده

ممکن است داده‌های برچسبی در سلول‌ها قرار گیرند و فرمولی در سلول A1 قرار گرفته باشد. برای تبدیل یک عدد به یک برچسب، ابتدا کاراکتر (') درج کنید تا به صورت یک برچسب تراز بندی شود سپس با کشیدن مکان‌نما روی مجموعه‌ای از داده‌ها انتخاب را انجام دهید.

ماکرو را اجرا کرده و هر داده‌ای که در سلول یک عدد صحیح باشد تبدیل به یک عدد خواهد شد و از نوع رشته‌ای در می‌آید. البته سلول‌های خالی و فرمول‌ها بدون تغییر باقی می‌مانند.

اگر کمی ذکاوت به خرج دهیم می‌توانیم این کد را به صورت حرفه‌ای تر در آوریم. می‌توانیم قسمت‌های کاراکتری را از قسمت‌های متنی سلول‌ها جداکنیم تا داده‌های آن سلول‌ها تبدیل به عدد شوند. از همین کد می‌توانیم برای تبدیل محتویات سلول به برچسب استفاده کنیم:


```
Sub contents_to_label ( )
Dim addr As String
For Each window In Windows
        For Each Worksheet In window.SelectedSheets

                For Each cell In Application.Selection
                        addr = Worksheet.Name & "!" & cell.Address

                        If Range(addr).Value < > 0 Then Range(addr) = "'" & _
                        CStr(Range(addr))
                Next cell
        Next worksheet

Next window
End Sub
```

این کد برای تبدیل برچسب‌ها به اعداد به کار می‌رود به استثنای دستور تبدیل که کنترل می‌کند آیا یک مقدار عددی وجود دارد یا خیر و برای تبدیل دوباره به یک رشته از تابع CStr استفاده می‌کنند. این تابع،کاراکتر (') را به جلوی عبارت موجود در سلول می‌چسباند تا از این پس به صورت یک برچسب خوانده شود.

فصل بیست و یکم

انتقال مجموعه‌ای از سلول‌ها

اغلب اتفاق افتاده است که متوجه شده‌ایم داده‌های یک ستون یا یک ردیف از یک کاربرگ صفحه گسترده با اطلاعات ستون یا ردیف دیگری از آن صفحه گسترده هیچ فرقی ندارد. می‌توانیم با عملیات کپی و چسباندن، این اطلاعات را به محل جدید نیز منتقل کنیم اما این کار در چند مرحله انجام می‌شود و همیشه این خطر وجود دارد که مقداری از اطلاعات در طی این تبادل از بین بروند. در این فصل به شما نشان داده خواهد شد که چگونه می‌توانید با یک جابه‌جایی ساده روی مجموعه سلول‌های انتخاب شده، این‌کار را انجام دهید.

در این مثال به صورت پیش‌فرض در نظر می‌گیریم که Sheet1 به عنوان محل ذخیره سازی موقت داده‌هاست و در حالت معمولی، تهی است. هم‌چنین می‌توانیم از صفحه گسترده بعدی استفاده کنیم یا این‌که یک صفحه گسترده جدید ایجاد کنیم. کد زیر را در یک ماژول وارد کنید:

```
Sub transpose ( )
For Each window In Windows
        Set mysheets = Window.SelectedSheets
        For Each Worksheet In window.SelectedSheets
                tempbook = ActiveWorkbook.Name

                Worksheet.Select

                Worksheet.Range(Application.Selection.Address).Copy

                VBAProject.sheet1.Range(Application.ActiveCell.Address).PasteSpecial

                VBAProject.Sheet1.Range(Application.Selection.Address).Copy

                Workbooks(tempbook).Activate

                temp = Application.ActiveCell.Address

                For Each cell In Application.Selection

                        cell.Value = " "
```

```
                Next cell

                Worksheet.Range(temp).PasteSpecial transpose := True

                Next worksheet
Next window
mysheets.Select
End Sub
```

مانند دیگر مثال‌ها، کد، پنجره‌های موجود در مجموعه Windows را در یک حلقه مرور می‌کند تا برگه‌های انتخاب شده را به دست آورد. متغیری با نام mysheets، برگه‌های انتخاب شده را در خود نگه می‌دارد. سپس کد از طریق یک حلقه، هر کاربرگ موجود در مجموعه برگه‌های انتخاب شده را مرور می‌کند.

نام کارپوشه فعال در متغیری با نام tempbook ذخیره می‌شود. سپس کاربرگی که از طریق حلقه انتخاب شده است، گزینش می‌شود. سپس متدCopy،مجموعه را بر اساس آدرس کاربرگ انتخاب شده کپی کرده و متد PasteSpecial آن‌را در Sheet1 از شئ VBAProject می‌چسباند. مجموعه درست در همان جایی چسبانده می‌شود که در برگه اصلی از آن‌جا کپی شده است.

شئ VBAProject نشان دهنده پروژه‌ای است که ماژول شما در آن درج شده است. با استفاده از VBAProject Properties | Tools از منوی کد و تایپ کردن یک نام جدید، آن‌را تغییر نام می‌دهیم اما دقت کنید که در صورت استفاده از این شئ در کدنویسی باید نام جدید آن‌را در کد درج کنیم.

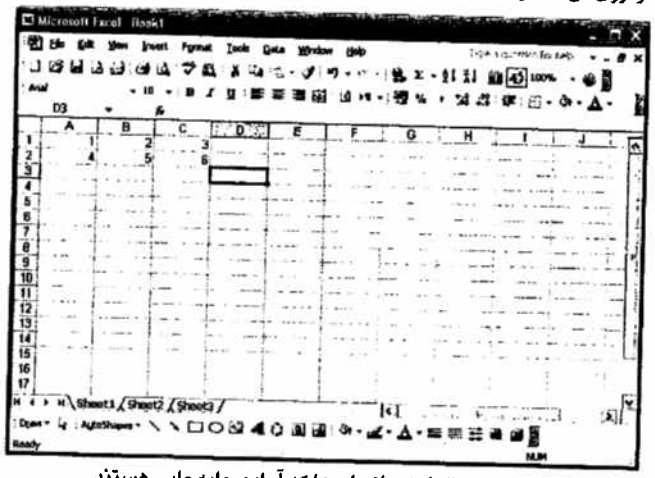
سپس با استفاده از نام موجود در متغیر tempbook ،کارپوشه اصلی فعال می‌شود (این متغیر جایی است که نام کارپوشه فعال را قبلاً در آن ذخیره کرده‌ایم). سپس برای کاربردهای بعدی، آدرس سلول فعال را در متغیری با نام temp ذخیره می‌کنیم.

با استفاده از حلقه For Each..Next ، همه سلول‌های موجود در مجموعه انتخاب شده برابر با Null تنظیم می‌شوند. باید همه آن‌ها را پاک کنیم تا داده‌های انتقال یافته جدید به جای آن‌ها درج شوند. اگر آن‌ها را پاک نکنیم، ترکیبی از داده‌های قدیمی و جدید را خواهیم دید که اصلاً کار دقیقی نیست.

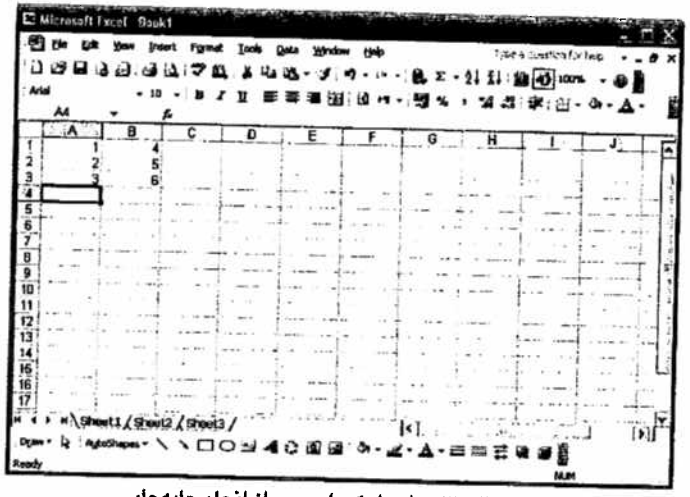
سپس با استفاده از آدرس سلول ذخیره شده در متغیر temp ، داده‌های موجود در حافظه clipboard را با استفاده از متد PasteSpecial چسبانده و مقدار متغیر Transpose را True قرار خواهیم داد.

در آخر با استفاده از متغیر mysheets، برگه‌هایی را که در ابتدا انتخاب شده بودند، دوباره انتخاب می‌کنیم. این کار باعث می‌شود مطمئن شویم که کارپوشه در همان وضعیتی قرار دارد که پیش از شروع کار کاربر و اجرای کد قرار داشت.

مانند تصویر ۱-۲۱، یک بلوک داده‌ها را وارد کنید. این داده‌ها را با کشیدن مکان‌نما انتخاب کرده و سپس کد را روی آن‌ها اجرا نمایید. نتایج کار در تصویر ۲-۲۱ نشان داده شده‌اند.



تصویر ۱-۲۱ بلوکی از داده‌ها که آماده جابه‌جایی هستند.



تصویر ۲-۲۱ همان بلوک داده پس از انجام جابه‌جایی

فصل بیست و دوم

اضافه کردن جزییات فرمول‌ها به توضیحات

وقتی یک صفحه گسترده کامل داشته باشیم اغلب لازم است برای اطمینان از صحت نتایج، فرمول‌های موجود را کنترل کنیم و این کار مستلزم مرور تعدادی سلول است که به یکدیگر زنجیر شده‌اند. تنها روش موجود در Excel برای بازبینی فرمول‌ها، کلیک کردن روی سلولی است که حاوی آن‌ها می‌باشد. این کار به زمان زیادی برای پیدا کردن سلول مربوطه نیاز دارد.

یک روش ساده‌تر، کپی کردن فرمول در یک عبارت توضیحی(Comment) برای آن سلول است. در این روش، فقط لازم است مکان‌نما را روی آن سلول نگه داریم تا فرمول در یک کادر توضیحات ظاهر شود. در واقع می‌توانیم فرمول موجود در سلول را بدون فعال کردن آن سلول ببینیم.

تنها مشکل موجود این است که ممکن است فرمول‌های زیادی در صفحه گسترده وجود داشته باشند که توضیح نوشتن برای تمام آن‌ها کمی سخت است. تازه اگر یکی از آن‌ها تغییر کند تکلیف چه خواهد بود؟

کد زیر، تمام فرمول‌های موجود در سلول انتخاب شده کاربر به کادر توضیحات را اضافه می‌کند. اگر توضیحی برای هر کدام از سلول‌ها وجود داشته باشد، آن‌را بدون تغییر حفظ می‌کند اما اگر توضیح موجود برای یک سلول به صورت یک فرمول باشد آن‌را به‌روز می‌نماید. برای جدا کردن توضیحات از فرمول‌ها از نماد (|) یا (SHIFT-\) استفاده می‌کنیم:

```
Sub note( )

For Each window In Windows
        For Each Worksheet In window.SelectedSheets

        For Each cell In Application.Selection

        addr = Worksheet.Name & " !" & cell.Address

         temp = " "

         On Error Resume Next

          temp = Range(addr).Comment.Text
```

```
    If InStr(temp, " |") Then
  temp = Mid(temp , InStr(temp, " |") + 1)
    End If
      Range(addr).ClearComments
      If Range(addr).HasFormula = True Then
          Range(addr).AddComment (cell.Formula & " |" & temp)
      Else
          If temp < > " " Then Range(addr).AddComment temp
      End If
                  Next cell
    Next worksheet
Next window
End Sub
```

این کد، تمام پنجره‌های موجود در مجموعه Windows و کاربرگ‌های موجود در برگه‌های انتخاب شده را در یک حلقه مرور می‌کند تا ببیند کدام کاربرگ‌ها انتخاب شده‌اند. سپس روی هر سلول موجود در آن برگه که در مجموعه انتخابی کاربر باشد عملیات اجرا می‌شود. متغیر addr نام کاربرگ و آدرس سلول را که با کاراکتر ! به آن چسبیده است، نگه می‌دارد.

برای ساختن رشته note که در آغاز برابر با Null در نظر گرفته شده است از متغیری با نام temp استفاده می‌کنیم. سپس با استفاده از ویژگی comment.text، متغیر temp را با متن موجود در توضیح فعلی بارگذاری می‌کنیم. البته اگر هیچ توضیحی وجود نداشت، یک خطا رخ می‌دهد. به این دلیل است که قبل از این بخش، خط کد On Error Resume Next نوشته شده است.

سپس با استفاده از تابع Instr جستجویی در رشته temp انجام می‌شود تا ببینیم آیا نماد | در آن‌جا وجود دارد یا خیر. اگر این کاراکتر در آن‌جا پیدا شد به معنای آن است که عملیات قبلاً روی آن سلول انجام شده و آن سلول حاوی یک فرمول است. البته اگر نماد | معنای دیگری در توضیحات شما دارد می‌توانید کد قبلی را تغییر دهید تا از نماد دیگری مانند (/) به عنوان معرف استفاده شود.

اگر از قبل فرمولی در بخش توضیحات وجود داشته باشد باید آن‌را حذف کنید. این کار را با حذف تمام کاراکترهای قبل از نماد | در رشته temp انجام می‌دهیم به طوری که بتوانیم فرمول جدیدی به این سلول اضافه کنیم.

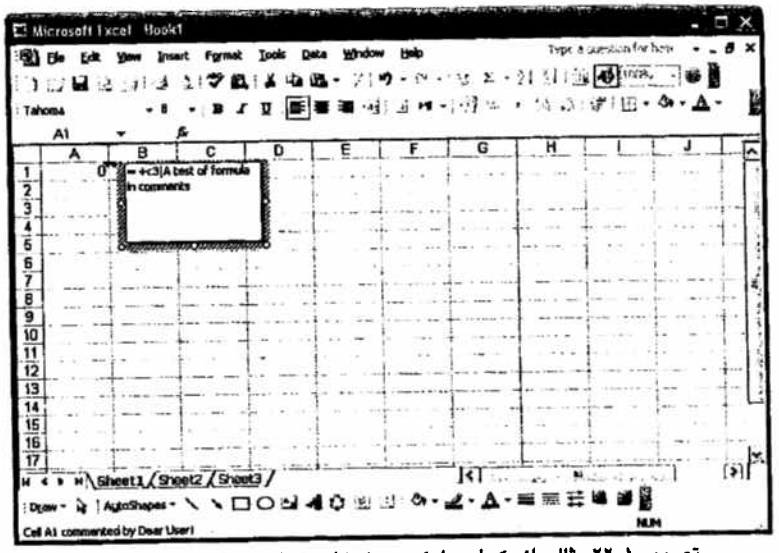
چون توضیحات مربوط به سلول در رشته temp وجود دارند کار ما در مرحله بعدی این است که توضیحات را از آن سلول حذف کنیم.

سپس کد کنترل می‌کند که آیا سلول فرمول دارد یا خیر و این کار با کنترل ویژگی HasFormula انجام می‌شود. اگر فرمولی وجود داشته باشد آن‌را در یک توضیح جدید اضافه می‌کنیم. در این توضیح، ابتدا فرمول و سپس نماد | برای نشان دادن خاتمه فرمول قرار می‌گیرد و در صورت دلخواه می‌توانیم پس از این نماد توضیحاتی نیز اضافه کنیم. به خاطر داشته باشید که اگر از قبل هیچ توضیحی وجود نداشته باشد مقدار temp برابر با Null در نظر گرفته می‌شود و بنابراین هیچ توضیحی نیز نشان داده نخواهد شد. اگر هیچ فرمولی وجود نداشت نیز می‌توانیم با کد زیر توضیح قدیمی را مجدداً اضافه کنیم:

```
If temp < > " "Then Range(addr).AddComment temp
```

اگر آن، یک رشته تهی باشد (چون از قبل هیچ توضیحی وجود نداشته است) آن‌گاه در نتیجه دستور If که در ابتدای این خط کد وجود دارد، هیچ توضیحی تنظیم نخواهد شد.

با کشیدن مکان‌نما روی مجموعه‌ای از سلول‌ها، انتخابی را روی صفحه گسترده انجام دهید و سپس کد را اجرا کنید. نتیجه باید شبیه شکل ۱-۲۲ باشد.



تصویر ۱-۲۲ مثالی از یک فرمول که در یک کادر توضیح نشان داده شده است.

سعی کنید فرمول را تغییر داده و کد را مجدداً اجرا کنید. قسمت فرمول در عبارت توضیحی تغییر خواهد کرد اما توضیح اصلی بدون تغییر باقی خواهد ماند.

فصل بیست و سوم

محاسبه یک مجموعه از سلول‌ها

اگر صفحه گسترده پیچیده‌ای در اختیار داشته باشیم، انجام محاسبات زمان زیادی را تلف می‌کنند به‌ویژه اگر چندین کاربرگ و فرمول‌های پیچیده‌ای نیز وجود داشته باشند. البته می‌توانیم با استفاده از Tools | Options انجام محاسبات را به صورت دستی تنظیم کنیم. اما این کار وقتی تنها می‌خواهیم یک برگه یا مجموعه‌ای از چند سلول را محاسبه کنیم از لحاظ زمانی مقرون به صرفه نیست. مایکروسافت اقدام به تعبیه متد range.calculate در مدل شئ Excel کرده است که البته از منوی صفحه گسترده قابل دسترسی نیست. برای استفاده از آن باید کد زیر را در یک ماژول بنویسیم:

```
Sub range_calculate ( )
For Each window In Windows
        For Each Worksheet In window.SelectedSheets
                For Each cell In Application.Selection
                        addr = Worksheet.Name & "!" & cell.Address
                        Range(addr).Calculate
                Next cell
        Next worksheet
Next window
End Sub
```

کاربر با کشیدن مکان‌نما روی مجموعه‌ای از سلول‌ها، آن‌ها را انتخاب می‌کند. البته می‌تواند این کار را از بین چند کاربرگ نیز انجام دهد.

کد، تمام پنجره‌های موجود در شئ Windows را همراه با تمام کاربرگ‌های موجود در شئ window.selectedsheets در یک حلقه مرور می‌کند تا برگه‌هایی که کاربر از بین آن‌ها سلول‌ها را انتخاب کرده است، پیدا کند .

سپس هر سلول مجموعه انتخابی نیز مرور می‌شود. متغیر addr، نام کاربرگ انتخاب شده را همراه با آدرس سلول انتخابی که با کاراکتر ! به آن چسبیده است، در خود نگه می‌دارد.

سپس سلول مورد محاسبه قرار می‌گیرد. در اصل تنها سلول‌های موجود در محدوده انتخابی که روی کاربرگ‌های گزینش شده قرار دارند در عملیات محاسبه، شرکت داده می‌شوند و این کار باعث صرفه‌جویی زیادی در زمان می‌شود.

سعی کنید مقداری داده همراه با چند فرمول روی یک صفحه گسترده قرار دهید. محاسبه را با استفاده از Tools | Options به صورت دستی تنظیم کنید و زبانه Calculation را انتخاب کرده و سپس دکمه رادیویی موجود را به Manual تنظیم کنید.

داده‌ها را تغییر داده و سپس همان مجموعه را انتخاب کنید البته چند سلول را که فرمول دارند در مجموعه انتخابی قرار ندهید. کد را اجرا کرده و نتایج را کنترل کنید. فرمول‌هایی که در داخل مجموعه انتخابی باشند مجدداً محاسبه می‌شوند اما آن‌هایی که بیرون مجموعه باشند بدون تغییر باقی می‌مانند.

فصل بیست و چهارم

وارونه کردن یک برچسب

اگر داده‌ها را از فایل یا برنامه دیگری به صفحه گسترده ارسال کنید، معمولاً در قالب موردنظر ما نخواهد بود. برای مثال، لیست نام‌ها در یک فایل دیگر ممکن است به صورت first name، last name باشد (مانند "Shepherd Richard") و ما بخواهیم آن‌ها در قالب last name، first name باشند (مانند "Richard Shepherd"). حال فرض کنید که هزاران نام از آن فایل به صفحه گسترده شما انتقال یابند. تکلیف این ناهماهنگی چه خواهد بود؟ آیا باید تک تک آن‌ها را به صورت دستی ویرایش کنیم؟ باور کنید من افرادی را دیده‌ام که این کار را می‌کنند اما عاقلانه‌تر آن است که این کار را با کدنویسی انجام دهیم و اکنون که از قابلیت‌های Excel VBA آگاه هستیم انجام این کار بسیار ساده است:

```
Sub reverse_label ( )

For Each window In Windows
        For Each Worksheet In window.SelectedSheets

                For Each cell In Application.Selection

                        addr = Worksheet.Name & "!" & cell.Address

                                If Len(Range(addr).Text) > 0 Then

                                        temp = Range(addr).Value

                                        x = InStr(temp," ")

                                        If x Then
temp = Mid(temp, x + 1) & " " &                 Left(temp, x – 1)

                                                Range(addr).Value = temp

                                        End If

                                End If

                Next cell
```

```
    Next worksheet
Next Window
End Sub
```

کاربر با کشیدن مکان‌نما روی مجموعه‌ای از سلول‌ها، آن‌ها را انتخاب می‌کند. البته می‌تواند این کار را از بین چند کاربرگ نیز انجام دهد.

کد، تمام پنجره‌های موجود در شئ Windows را همراه با تمام کاربرگ‌های موجود در شئ window.selectedsheets در یک حلقه مرور می‌کند تا برگه‌هایی که کاربر از بین آن‌ها سلول‌ها را انتخاب کرده است، پیدا کند .

سپس هر سلول در مجموعه انتخابی نیز مرور می‌شود. متغیر addr، نام کاربرگ انتخاب شده را همراه با آدرس سلول انتخابی که با کاراکتر ! به آن چسبیده است، در خود نگه می‌دارد. متن موجود در سلول مورد آزمایش قرار می‌گیرد تا اطمینان حاصل شود که در آن داده وجود دارد؛ در صورتی که چیزی در سلول وجود نداشته باشد کد عملیاتی انجام نمی‌دهد.

سپس متغیری با نام temp همراه با مقدار سلول بارگذاری می‌شود. با استفاده از تابع Instr، جستجویی برای کاراکتر فاصله (space) انجام می‌شود. کد فرض می‌کند که تنها یک کاراکتر فاصله بین نام کوچک و نام فامیل وجود دارد. آدرس آن کاراکتر در متغیر x بارگذاری می‌شود. البته اگر از کاراکتر فاصله برای جدا کردن نام کوچک و نام فامیل استفاده نمی‌کنید، می‌توانید تغییرات دلخواه را در کد اجرا کنید.

اگر مقدار x غیر صفر باشد (یعنی یک مقدار واقعی داشته باشد) به معنای آن خواهد بود که کاراکتر فاصله پیدا شده است. متغیر temp همراه با قسمتی از temp که پس از کاراکتر فاصله قرار دارد،بارگذاری می‌شود،سپس خود کاراکتر فاصله قرار می‌گیرد و در آخر نیز قسمتی از temp که قبل از کاراکتر فاصله بوده است قرار می‌گیرد. به مثال زیر توجه کنید:

```
temp = "Richard Shepherd"
x = InStr(temp, " ")
```

چون کاراکتر فاصله،کاراکتر هشتم در رشته است پس مقدار متغیر x برابر با 8 خواهد بود.

```
temp = Mid(temp, x + 1) & " " & Left(temp, x – 1)
```

در این‌جا، Mid(temp,x+1) مانند Mid(temp,9) عمل می‌کند. هر دوی آن‌ها به معنای این هستند که تمام کاراکترها از موقعیت کاراکتر نهم تا انتهای رشته انتخاب شوند. البته می‌توانستیم تعداد کاراکترهای انتخابی پس از کاراکتر نهم را هم مشخص کنیم که در آن صورت عملیات انتخاب تا انتهای

رشته صورت نمی‌گرفت. نتیجه این دو دستور، "Shepherd" خواهد بود. Left(temp,x-1) مانند نوشتن Left(temp,7) عمل می‌کند نتیجه هر دوی آن‌ها "Richard" خواهد بود (یعنی 7 کاراکتر از سمت چپ).

این دو قسمت با یک کاراکتر فاصله که در وسط آن‌ها قرار می‌گیرد به یکدیگر الحاق می‌شوند و نتیجه حاصل، "Shepherd Richard" خواهد بود. سپس رشته temp به سلول برگردانده می‌شود.

سعی کنید لیستی از نام‌ها را در محدوده‌ای از صفحه گسترده وارد کنید. در وسط بعضی از آن‌ها از کاراکتر فاصله استفاده کنید و در وسط بقیه از آن استفاده نکنید. این محدوده را با کشیدن مکان‌نما روی سلول‌هایش انتخاب کنید. کد را اجرا کنید و خواهید دید که هرجا از کاراکتر فاصله استفاده کرده‌اید، نام، وارونه خواهد شد.

این کد را برای شرایط دیگری هم می‌توان به کار گرفته و اصلاح کرد تا کاراکترهایی مانند کاما (،) یا (/) را به جای کاراکتر فاصله در نظر بگیرد.

فصل بیست و پنجم

چه کسی کارپوشه را ایجاد کرده است؟

توابع استاندارد Excel، اطلاعات ارزشمندی ارایه می‌کنند که می‌توان از آن‌ها در صفحه گسترده استفاده کرد اما یکی از چیزهایی که در این توابع در نظر گرفته نشده است، جزییات درباره فردی است که صفحه گسترده را ایجاد کرده و هم‌چنین تاریخ و زمان ساخت آن نیز در نظر گرفته نشده است. البته می‌توانید از منوی صفحه گسترده، File | Properties را انتخاب کنید تا این موارد را به دست آورید اما بسیار ساده‌تر خواهد بود که یک فرمول استاندارد داشته باشیم تا این اطلاعات را گرفته و مستقیماً روی صفحه گسترده نمایش دهد.

همچنان‌که در فصل ۳ دیدیم می‌توانیم توابع اختصاصی برای خود ایجاد کنیم پس بسیار ساده خواهد بود که تابعی بنویسیم تا به کاربر بگوید چه کسی کارپوشه را ایجاد کرده است:

```
Function WHO ( )
temp = "Created By"
Dim Workbook As Workbook
Set Workbook = Application.ActiveWorkbook
For Each property In Workbook.BuiltinDocumentProperties
        On Error Resume Next
        If property.Name = "Author" Then temp = temp & property.Value
        If  property.Name = "Creation date" Then temp = temp & " on " &
        Property.Value
Next property
WHO = temp
End Function
```

به یاد داشته باشید که این یک تابع است و زیرروال محسوب نمی‌شود پس کمی متفاوت از مثال‌های قبلی که دیده‌ایم عمل می‌کند. چون تنها داده‌هایی را از خصوصیت‌هایی که در Excel نگه داشته می‌شوند، گرفته و آن‌ها را به فرمولی روی صفحه گسترده بر می‌گردانیم، نیازی نیست تا پارامتری به این تابع انتقال یابد.

اولین کاری که باید انجام دهیم، ایجاد متغیری با نام temp است تا رشته "Created by" را در خود نگه دارد. این عبارت، قسمت نخست رشته برگشتی در صفحه گسترده خواهد بود و دیگر قسمت‌های آن رشته توسط خصوصیت‌های موجود پر خواهند شد.

متغیری با نام Workbook از نوع Workbook تعریف شده است. سپس این متغیر برای کارپوشه فعال که در حال حاضر در حالت انتخاب شده قرار دارد و فرمول در آن‌جا درج خواهد شد تنظیم می‌شود. این امر بدین دلیل ضروری است که اگر چندین کارپوشه بارگذاری شده بودند بتوانیم کارپوشه صحیح را انتخاب کرده و اطلاعات مربوط به آن‌را استخراج کنیم.

سپس کد تمام خصوصیت‌های موجود در مجموعه BuiltInDocumentProperties را در یک حلقه مرور می‌کند. این مجموعه در تمام برنامه‌های کاربردی Office مشترک است پس مجموعه BuiltInDocumentProperties حاوی پارامترهای بسیار زیادی است که بسیاری از آن‌ها در یک برنامه کاربردی Excel قابل استفاده نیستند. خصوصیت‌های اصلی، شبیه آن‌هایی هستند که در صورت انتخاب File | Properties، سپس کلیک روی زبانه Summary به نمایش درمی‌آیند؛ خصوصیاتی مانند Subject،Title و Author که تمام آن‌ها را می‌توان در نمونه کدی که قبلاً نشان دادیم به‌کار بگیریم. در این‌جا به‌کارگیری دستور On Error Resume Next ضروری است چون امکان نمایش بعضی خصوصیت‌ها وجود ندارد و آن‌ها باعث بروز خطا می‌شوند.

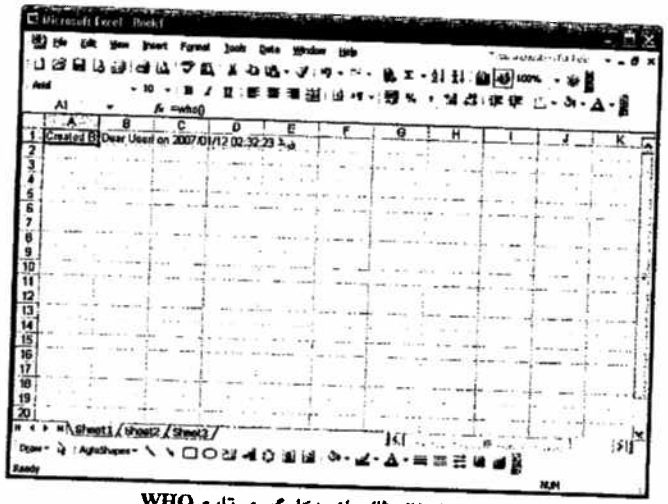
اگر نام خصوصیت، Author باشد آن‌گاه مقدار آن‌را به رشته temp خواهیم چسباند تا به صورت "Created by Richard Shepherd" خوانده شود. توجه کنید که وقتی رشته temp ایجاد شد، آخرین کاراکتر آن، کاراکتر فاصله بود، پس آن نام کاربری به درستی به "on" چسبیده است.

اگر نام خصوصیت creation date باشد آن‌گاه کلمه " on " را باید بچسبانیم و به کاراکتر فاصله قبل و بعد از on توجه کنیم و سپس مقدار این خصوصیت را قرار می‌دهیم.

سپس متغیر WHO با مقدار موجود در رشته temp بارگذاری می‌شود و به صفحه گسترده انتقال می‌یابد. برای آزمایش این قضیه لازم نیست کد را اجرا کنید بلکه تنها کافی است فرمول (=WHO()) را مانند همیشه در یکی از سلول‌های کاربرگ تایپ کنید. اگر روی نوار ابزار Formula روی آیکن Formula Paste کلیک کنیم، این فرمول در بخش User Defined Formula (فرمول‌های تعریف شده

توسط کاربر) ظاهر خواهد شد و می‌توانیم از آن مانند دیگر فرمول‌ها استفاده کنیم. اگر پارامتری همراه با آن درج کنیم، پیام‌های خطای Excel صادر خواهند شد.

وقتی کد را در یک ماژول وارد کردید، عبارت (=Who() را در یک سلول تایپ کنید. توجه کنید که هنوز هم باید از پرانتز استفاده کنید حتی اگر قرار نباشد پارامتری انتقال یابد. نتیجه باید شبیه تصویر ۱-۲۵ باشد.



تصویر ۱-۲۵ مثالی از به‌کارگیری تابع WHO

فصل بیست و ششم

ارزیابی یک سلول

گاهی اوقات در یک صفحه گسترده، سلولی حاوی کاراکترهای عددی و کاراکترهای حرفی است. برای مثال سلولی که حاوی RJS123 است، کاراکترها را به صورت متنی در خود نگه می‌دارد. حال ممکن است بخواهیم قسمت عددی این متن را از قسمت حرفی جدا کنیم به طوری که سلول تنها حاوی مقدار 123 باشد اما انجام این کار ساده نخواهد بود. تابع Value، یک برچسب را که حاوی داده‌های عددی است، به یک عدد واقعی تبدیل می‌کند اما اگر ابتدای آن برچسب حاوی کاراکترهای الفبایی باشد، جواب نخواهد داد. البته می‌توانیم یک تابع اختصاصی برای این کار بنویسیم. برای مثال اگر سلول دارای برچسب ART3478BC باشد ممکن است بخواهیم قسمت عددی آن یعنی 3478 را جدا کنیم. می‌توانیم از تابع Mid برای انجام این کار استفاده کنیم اما این تابع فرض می‌کند که ساختار حروف همیشه یکسان است. بنابراین اگر ما همیشه قبل از یک عدد چهار رقمی، سه کاراکتر حرفی و پس از آن نیز دو کاراکتر حرفی نداشته باشیم، آن‌گاه با هر بار تغییر باید کد مربوط به دستور Mid را نیز تغییر دهیم چون ساختار دستور Mid همیشه ثابت است. کد زیر، ابزار قابل اتکایی برای استخراج اعداد از برچسب‌ها در اختیار ما قرار می‌دهد:

```
Function EVAL(cell_ref As Object)
        Application.Volatile
        t = ""
        For Each cell In cell_ref
                temp = cell.Value

                For n = 1 to Len(temp)

                        If IsNumeric(Mid(temp, n, 1)) Then

                                t = t & Mid (temp, n, 1)

                        End If

                Next n
        Next cell
        EVAL = Val(t)

End Function
```

این تابع، متضاد یک زیرروال است و متفاوت از آن نیز عمل می‌کند. مجموعه سلول‌های انتخابی به فرم یک شئ با نام cell_ref به آن انتقال می‌یابند .

Application.volatile تضمین می‌کند که هرگاه هر کدام از سلول‌های موجود روی کاربرگ مجدداً مورد محاسبه قرار گرفتند، تابع نیز دوباره محاسبه شود. سپس کاربر می‌تواند مانند وقتی که از تابع SUM استفاده می‌کند، یک یا چند سلول را با کشیدن ماوس روی کاربرگ انتخاب کند. اگر تنها یک سلول انتخاب شده باشد، پیدا کردن مبدأ مقدار عددی کار ساده‌ای خواهد بود. برای مثال ABC1234 به صورت 1234 به نمایش در می‌آید. البته اگر چندین سلول انتخاب شده باشند این کار نتایج عجیب و غریبی به همراه خواهد داشت چون تمام مقادیر عددی به یکدیگر چسبانده می‌شوند.

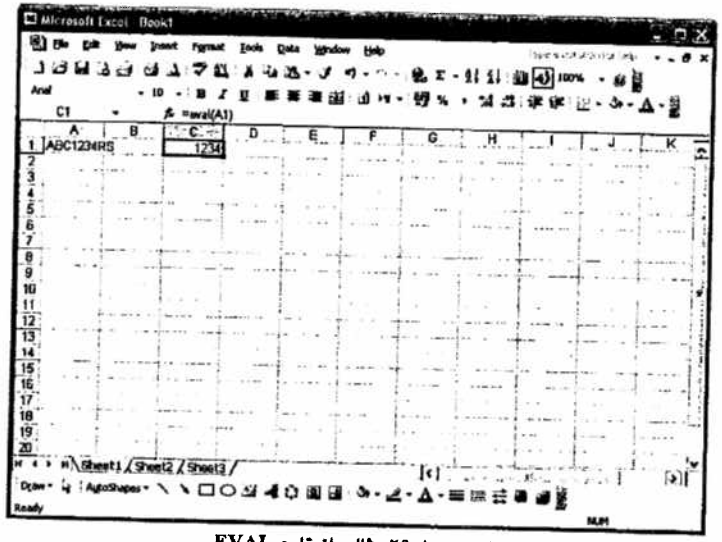
متغیری با نام t (که تهی قرار داده می‌شود) رشته‌ای عددی می‌سازد که برای سلول باز پس فرستاده می‌شود. سپس در یک حلقه تمام سلول‌های موجود در شئ cell_ref مرور می‌شوند. متغیری با نام temp بارگذاری شده و مقدار سلول در آن نگه داشته می‌شود.

سپس کد از حلقه For..Next استفاده می‌کند تا رشته‌ای را که معادل مقدار سلول است کاراکتر به کاراکتر بررسی کند این حلقه هر کاراکتر را با استفاده از تابع IsNumeric کنترل می‌کند تا ببیند آیا یک مقدار عددی است یا خیر. اگر عددی بود آن‌را به متغیر t می‌چسباند.

وقتی کار روی تمام سلول‌های موجود در مجموعه انتخابی تمام شد کد، مقدار t را با استفاده از تابع Val به دست آورده و این مقدار را با استفاده از متغیر EVAL بر می‌گرداند. برای آزمایش این قضیه نیازی به اجرای کد نیست تنها فرمولی مانند روال معمولی در صفحه گسترده درج کنید (مثلاً =EVAL(A1)).

اگر روی آیکن Formula Paste از نوار ابزار Formula کلیک کنیم این فرمول در بخش User Defined Formula (فرمول‌های تعریف شده توسط کاربر) ظاهر خواهد شد و می‌توانیم از آن مانند دیگر فرمول‌ها استفاده کنیم. اگر از پرانتزها استفاده نکنیم، پیام‌های خطای Excel صادر خواهند شد.

وقتی کد را در یک ماژول وارد کردید، عبارت ABC1234RC را در سلول A1 تایپ کنید و در سلول دیگری نیز عبارت =EVAL(A1) را قرار دهید. نتیجه خروجی در سلولی که حاوی فرمول است، 1234 به صورت یک مقدار عددی خواهد بود. فرمول شما باید شبیه تصویر ۱-۲۶ باشد.



تصویر ۱-۲۶ مثالی از تابع EVAL

اگر داده‌ها را از منبع دیگری مانند یک پایگاه داده دریافت می‌کنید یا می‌چسبانید ممکن است بخواهید از این تابع استفاده کنید. هم‌چنین اگر یک کد مرجع ارسالی داشته باشیم که شامل حروف و اعداد باشد، ممکن است بخواهیم از این تابع برای جدا کردن اعداد استفاده کنیم.

فصل بیست و هفتم

مرتب کردن کاربرگ‌ها به ترتیب الفبا

به موازات گسترش کارپوشه‌ها، تعداد کاربرگ‌ها نیز افزایش یافته و اغلب به طور تصادفی و غیر منظم، مرتب می‌شوند. البته می‌توانیم با استفاده از زبانه برگه‌ها، آن‌ها را کشیده و مرتب کنیم اما وقتی تعداد برگه‌ها بسیار زیاد باشد، این کار چندان جالب نخواهد بود. در این فصل کدی را مرور خواهیم کرد که می‌تواند به سرعت و به صورت کارآمد، کاربرگ‌های موجود را با توجه به نامی که برای هر کاربرگ روی زبانه‌های موجود در قسمت پایین صفحه مشخص شده است، مرتب کند. اگر نام کاربرگ‌ها الفبایی باشد برحسب حروف الفبایی مرتب می‌شوند و اگر این نام‌ها عددی باشند، بر حسب اعداد مرتب خواهند شد. نام‌های عددی نسبت به نام‌های الفبایی اولویت دارند و قبل از آن‌ها قرار می‌گیرند.

کد زیر را در یک ماژول وارد کنید:

```
Sub sortsheet ( )

For  i = 1 To Sheets.Count

        For j = 1 To Sheets.count -1

            If UCase$(Sheets(j).Name) > UCase$(Sheets(j + 1).Name) Then

                Sheets(j).Move after: =Sheets(j + 1)

            End If
        Next j
Next i
End Sub
```

این کد طوری طراحی شده است که تنها روی کارپوشه فعلی عمل می‌کند یعنی کارپوشه‌ای که در حال حاضر، انتخاب شده است.

این کد با استفاده از ویژگی Count و یک حلقه For..ext (تعداد دفعات اجرای حلقه بر مبنای مقدار موجود در متغیر Count تعیین می‌شود) مجموعه Sheets را مرور می‌کند. در قسمت بعدی یک حلقه تو در تو قرار دارد که بجز برگه پایانی، تمام برگه‌های مشابه را مرور می‌کند. در واقع کد، آزمایش

می‌کند تا ببیند آیا نام برگه فعلی از نام برگه بعدی بزرگ‌تر است یا خیر و چون بعد از برگه آخر، برگه‌ای قرار ندارد در عملیات شرکت داده نمی‌شود.

به خاطر آن که اختلاف بین حروف کوچک و بزرگ در نام برگه‌ها باعث بروز نتایج نادرست می‌شود، در اول کار از تابع Ucase برای تبدیل نام برگه‌ها به حروف بزرگ استفاده کرده‌ایم. اگر نام برگه فعلی از برگه بعدی بزرگ‌تر باشد از متد Move برای جابه‌جا کردن آن‌ها استفاده می‌کنیم و با تنظیم پارامتر after به برگه شاخص گذاری شده بعدی، این کار انجام می‌شود. کاربرگ‌های موجود در یک کارپوشه نمی‌توانند نام مشابه داشته باشند پس کد با مشکلی مواجه نخواهد شد. این روش را اصطلاحاً مرتب سازی حبابی می‌نامیم.

کاربرگ‌های موجود را با راست کلیک کردن روی زبانه Sheet در قسمت پایین و انتخاب Rename، تغییر نام دهید. از نام‌های تصادفی استفاده کرده و توجه کنید که ترتیب نام برگه‌ها به هم ریخته باشد. کد را اجرا کنید تا برگه‌ها همراه با محتویات انتقال یابند.

فصل بیست و هشتم

جای‌گزینی کاراکترها در یک رشته

کد این فصل، یک تابع ایجاد می‌کند که تمام موارد موجود از یک کاراکتر خاص در یک رشته را با کاراکتر خاص دیگری تعویض می‌نماید. برای مثال اگر سلول در صفحه گسترده ما وجود داشته باشد که حاوی متن "***Replacement String***" باشد می‌توانیم از فرمول Replace برای تعویض کاراکتر * با کاراکتر ! استفاده کنیم تا رشته به صورت "!!!Replacement String!!!" خوانده شود.

این تابع به‌ویژه برای داده‌هایی که از دیگر برنامه‌ها ارسال یا چسبانده می‌شوند، مفید است. گاهی اوقات داده‌های دریافتی شامل کاراکترهای خاصی هستند که لازم است همگی حذف شوند یا به کاراکتر دیگری تبدیل شوند.

چون این تابع، یک تابع عمومی است می‌تواند در داخل کد VBA نیز به کار گرفته شود. کد زیر را در یک ماژول وارد کنید:

```
Function REP(target As String, r As String, s As String)
Temp = ""
For n = 1 to Len(target)

        If Mid(target, n, 1) = r Then

                temp = temp & s

        Else

                temp = temp & Mid(target, n, 1)
        End If

REP = temp
Next n
End Function
```

این یک تابع است و متفاوت از زیرروال‌ها عمل می‌کند. سه پارامتر به آن انتقال می‌یابند. متغیر temp، پوچ (null) تنظیم شده است و در ادامه برای ساختن یک رشته جدید به کار می‌رود. سپس از یک حلقه For..Next برای حرکت کاراکتر به کاراکتر در target استفاده کرده‌ایم. تابع Mid هر کاراکتر را از

رشته جدا کرده و آن را آزمایش می‌کند تا ببیند آیا با کاراکتر مورد جستجو مساوی است یا خیر. با توجه به این که عملیات تطبیق نسبت به بزرگ و کوچک بودن حروف حساس است، کاراکترها باید دقیقاً عین هم باشند.

اگر تطبیقی به دست آمد، کاراکتری که توسط s نشان داده شده است به انتهای temp چسبانده می‌شود. اگر تطبیقی به دست نیامد، کاراکتر اصلی به temp چسبانده می‌شود. در آخر، REP که نشان‌دهنده تابع است، مقدار رشته‌ای متغیر temp را به دست آورده و مجدداً آن مقدار را در سلول قرار می‌دهد.

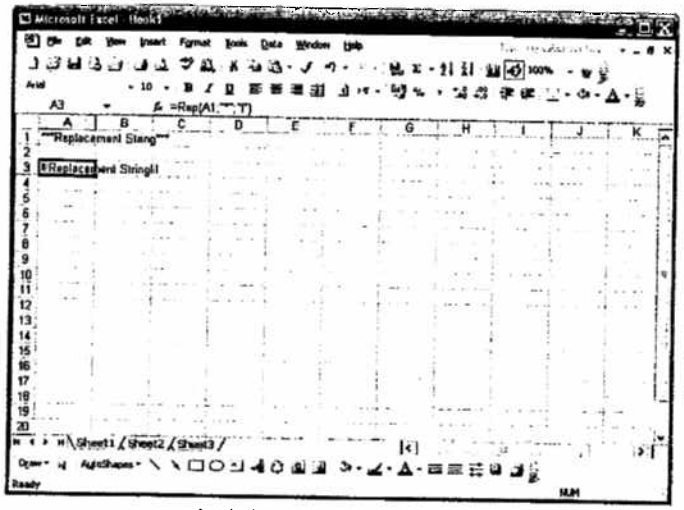
برای آزمایش این تابع لازم نیست کد را اجرا کنیم بلکه می‌توانیم فرمول را مانند حالت معمولی Excel در یکی از سلول‌های صفحه گسترده تایپ کنیم:

```
=Rep(A1,"*","!")
```

اگر روی آیکن Formula Paste از نوار ابزار Formula کلیک کنیم این فرمول در بخش User Defined Formula (فرمول‌های تعریف شده توسط کاربر) ظاهر خواهد شد و می‌توانیم از آن مانند دیگر فرمول‌ها استفاده کنیم. اگر پارامترهای صحیح را برای آن تعیین نکنیم، پیام‌های خطای Excel صادر خواهند شد. کاربرگ شما باید شبیه تصویر ۱-۲۸ باشد.

نکته: *یک روش ساده برای حذف یک کاراکتر خاص از یک رشته، این است که در دستور REP رشته جستجو را به صورت طبیعی مشخص کنیم اما به جای رشته جایگزین، یک رشته تهی را در نظر بگیریم:*

```
=Rep(A1,"*","")
```

تصویر ۱-۲۸ مثالی از تابع جای‌گزینی

فصل بیست و نهم

رویدادهای زمان‌بندی شده

وقتی یک برنامه کاربردی می‌نویسیم اغلب می‌خواهیم یک رویه بر مبنای یک فاصله زمانی منظم اجرا شود. برای مثال ممکن است بخواهیم برنامه الحاقی ذخیره سازی خودکار Excel را شبیه‌سازی کرده و هر 5 دقیقه، روال ذخیره‌سازی (save) را فراخوانی کرده یا هر 5 دقیقه، کارپوشه را مجدداً محاسبه کنیم. اگر کارپوشه بزرگ و پر از فرمول‌های پیچیده باشد باید از محاسبه دستی استفاده کنیم تا مطمئن شویم هر 5 دقیقه، تمام محاسبات به روز در آمده‌اند.

اگر از ویژوال بیسیک در ویندوز استفاده کرده باشید با کنترل timer (زمان‌سنج) در آن آشنا هستید. در این کنترل، کاری که باید انجام دهیم این است که فاصله زمانی تایمر (interval) را تنظیم کنیم و کد را برای رویداد تایمر بنویسیم. متأسفانه مشابه این کنترل در VBA وجود ندارد و اجرای یک رویداد زمان‌بندی شده روند پیچیده‌ای دارد. البته با استفاده از متد OnTime می‌توانیم این کار را انجام دهیم. کدی که در این متد قرار دارد، زمانی را در آینده مشخص می‌کند و هنگامی که زمان سیستم با زمان مشخص شده مساوی شد، زیرروالی را فرا می‌خواند. کد زیر را در یک ماژول بنویسید:

```
Private Sub time_set
Dim w As Workbook
Application.OnTime Now + TimeValue("00:05:00") , "time_set"
For Each w In Application.Workbooks
        w.Save
Next w
End Sub
```

این رویه فقط به یک بار فراخوانی نیاز دارد. می‌توانیم وقتی کارپوشه برای نخستین مرتبه بارگذاری می‌شود آن‌را از طریق رویداد Workbook_Open فرا بخوانیم.

در کد قبلی، ابتدا متغیر w به عنوان یک کارپوشه تعیین می‌شود. سپس دستور OnTime مشخص می‌کند که در عرض 5 دقیقه (مدت زمان تعیین شده توسط متد Now به علاوه مقدار زمانی 5 دقیقه)، همان روال time_set دوباره فراخوانده شود. سپس با استفاده از یک حلقه For..Each، تمام کارپوشه‌های بارگذاری شده، مرور و ذخیره می‌شوند. 5 دقیقه بعد، روال مجدداً فراخوانده می‌شود. در

این مرحله خود را طوری تنظیم می‌کند که 5 دقیقه بعد دوباره فراخوانده شده و سپس دوباره تمام کارپوشه‌ها را ذخیره کند.

تا این‌جای کار تنها از دو پارامتر در OnTime استفاده کرده‌ایم:

Procedure و EarlisetTime

می‌توانیم پارامتر LatestTime را نیز مشخص کنیم که آخرین زمانی است که رویه باید اجرا شود. وقتی زمان سیستم از آخرین زمانی که مشخص کرده‌ایم گذشت، رویه OnTime دیگر فراخوانده نخواهد شد.

مقدار پارامتر Schedule را می‌توانیم True یا False تعیین کنیم که باعث روشن یا خاموش شدن تابع OnTime می‌شود. اگر Schedule را False تعیین کنیم این تابع دیگر اجرا نخواهد شد.

هم‌چنین می‌توانیم آخرین زمانی را که (LatestTime) می‌خواهیم اجرای رویداد زمان‌بندی شده متوقف شود، مشخص کنیم:

```
Application.OnTime Now + TimeValue("00:05:00"), _
"time_set", TimeValue("22:00:00")
```

در این مثال، رویه time_set هر 5 دقیقه یک بار تا ساعت 10:00 P.M فراخوانده می‌شود. برای توقف اجرای اتوماتیک رویه، مقدار پارامتر Schedule را False قرار می‌دهیم:

```
Application.OnTime Now + TimeValue("00:05:00"), "time_set",  , False
```

اگر قصد نوشتن ماکروهای VBA برای صنعت Shareware دارید، این یک روش بسیار جالب برای تعریف کادرهای پیغام "nag" است که کاربر را مجبور می‌کند محصول شما را ثبت کند.

فصل سی‌ام

جمع کردن خودکار ماتریسی از وسط اعداد

مسأله‌ای که اکثر کاربران صفحه گسترده به نحوی با آن سروکار دارند، جمع کردن خودکار ماتریسی از اعداد است. یعنی ما یک مجموعه بزرگ از مقادیر داشته باشیم که یک ماتریس از اعداد را شکل دهند و لازم باشد در انتهای هر ستون و پایان هر ردیف، جمع ستون‌ها و ردیف‌ها را داشته باشیم و در گوشه سمت راست- پایین نیز مجموع کلی قرار گیرد. فرمول‌نویسی جداگانه برای هر ردیف و ستون کار پر زحمتی است و امکان وقوع خطا نیز در آن بسیار زیاد است.

کد زیر، روی مجموعه انتخابی کاربر عمل کرده و تمام مجموع‌ها را ایجاد می‌کند حتی اگر روی چندین کاربرگ کار کنیم:

```
Sub matrix_total ( )

If Appication.Selection.Count = 1 Then msgBox "Select a range" : Exit Sub

Dim rr As Range
Dim coord(4) As String
temp = Application.Selection.Address & "$"
Set rr = ActiveWindow.RangeSelection
Coord(0) = rr.Column
Coord(2) = rr.Column + rr.Columns.Count – 1
Coord(1) = rr.Row
Coord(3) = rr.Row + rr.Rows.Count – 1

For Each Window In Windows
    For Each worksheet In window.SelectedSheets
        con = 1

        For n =coord(0) To coord(2)
            Formulastr = convert_asc(VAL(coord(0)) + con – 1 ) & _
            coord(1) & ".." & convert_asc(VAL(coord(0)) + con – 1) _
             & coord(3)

                Worksheet.Cells(VAL(coord(3)) + 1, n).Formula = _
"=sum(" & formulastr & ")"
            con =con  + 1
        Next n
```

```
        con = 1
        for n = coord(1)  To coord(3) + 1
                formulastr = convert_asc(VAL(coord(0))) & (coord(1)  + _
                 con – 1) & ".."   & convert_asc(VAL(coord(2))) & _
                 (coord(1) + con -1 )

                 Workheet.Cells(n,  VAL(coord(2)) +1).Formula = _
"=sum(" & formulastr & ")"
                 con = con +1
        Next
    Next Worksheet
Next Window
End Sub
```

این کد تا حدی پیچیده است چون برنامه مجبور است ابتدا مختصات گوشه سمت چپ- بالا و سپس مختصات گوشه سمت راست- پایین محدوده انتخابی را مشخص کند. اگر تنها یک سلول انتخاب شده باشد روال انتخاب مختصات با مشکل مواجه می‌شود و چون نتیجه بی معنی حاصل می‌شود، در حالت انتخاب تک سلولی اصلاً کد اجرا نمی‌شود.

اگر تعداد سلول‌ها بیش از یکی باشد، آرایه‌ای ایجاد می‌شود تا مختصات به دست آمده برای مجموعه انتخابی را برای خود نگه دارد. متغیری با نام temp، آدرس واقعی مجموعه انتخاب شده را در خود ذخیره می‌کند. مقدار این متغیر با استفاده از کاراکتر $ همیشه در قالب مطلق است ($B$4:$D$6). در مواردی که قرار است عملیات جستجو انجام شود، یک کاراکتر $ دیگر نیز اضافه می‌شود.

متغیری با نام rr ایجاد می‌شود تا دامنه انتخابی را در خود ذخیره کند. از این متغیر می‌توانیم با استفاده از ویژگی‌های column و row ، مختصات چهار گوشه مجموعه انتخابی را مشخص کنیم.

سپس ماکرو از طریق یک حلقه، مجموعه Windows و تمام کاربرگ‌های موجود در آن را مرور می‌کند تا برگه‌های انتخاب شده را مشخص کند سپس در برگه انتخاب شده، از یک حلقه For..Next برای کار روی ستون‌های انتخابی آرایه (عناصر 0 و 2) استفاده می‌کنیم. element(0) حاوی شماره ستون آغازین و element(2) حاوی شماره ستون آخر است. متغیر con روال تغییر شماره ستون را حفظ کرده و با هر بار اجرای حلقه، یک واحد به مقدار آن اضافه می‌شود.

در این مرحله، برای هر ستون باید فرمولی به کار گرفته شود تا مقادیر موجود در آن‌را با یکدیگر جمع کند. متغیری با نام formulastr ، مختصات ستون را در خود نگه می‌دارد. برای مثال این متغیر می‌تواند مقدار D3...D7 را در خود ذخیره کند. این متغیر از تابعی با نام convert_asc استفاده می‌کند که عملیات تبدیلی را که قبلاً انجام شده است، لغو کرده و اعداد را دوباره به حروف ستونی تبدیل می‌کند.

کد تابع convert_asc به شرح زیر است:

```
Private Function convert_asc(target As Integer) As String
high = Int(target / 26)
low = target Mod 26
temp = " "
If high > 0 Then temp = Chr(high + 64)
temp = temp & Chr(low + 64)
convert_asc = temp
End Sub
```

عددی که نشان دهنده شماره ستون است، به صورت یک عدد صحیح (integer) انتقال می‌یابد. با توجه به این که شماره ستون‌ها، دو رقمی است این شماره به دو بخش تقسیم شده و قسمت بالایی آن یعنی بخشی که نشان دهنده رقم نخست شماره ستون است انتقال می‌یابد. اگر شماره ستون یک رقمی بود نیز این عدد برابر 0 خواهد بود.

قسمت پایین از تابع Mod استفاده می‌کند تا باقی مانده را به دست آورد که نشان دهنده ستون دوم است. به عدد قسمت بالا، 64 واحد اضافه می‌شود و تابع Chr آن را دوباره به یک حرف تبدیل می‌کند. اهمیت عدد 64 این است که کد اسکی کاراکتر A برابر 65 است. قبلاً سیستمی داشتیم که در آن، 1 نشان دهنده ستون A بود حال اگر 65 واحد به 1 اضافه کنیم به عدد 65 می‌رسیم که کد اسکی حرف A است. وقتی الگوریتم به ستون‌های دوحرفی مانند AA می‌رسد کار از این هم پیچیده‌تر می‌شود. متغیرهای بالا و پایین، عدد را با تقسیم آن بر 26 برای قسمت بالایی (حرف نخست ستون) و سپس استفاده از Mod برای قسمت پایینی (حرف دوم ستون)، عدد را به دو بخش حرفی تقسیم می‌کند. به قسمت پایینی، 64 واحد اضافه می‌شود و تابع Chr آن را مجدداً به یک حرف تبدیل می‌کند. سپس این کاراکتر به کاراکتر حرفی بالایی الحاق می‌شود پس می‌توانیم از طریق حروف به ستون دلخواه ارجاع کنیم.

به رویه اصلی برمی‌گردیم. سعی می‌کنیم تا برای هر ستون در دامنه انتخاب شده، فرمول مناسب به دست آوریم. اگر کاربر محدوده D4...F6 را انتخاب کرده باشد، باید سه فرمول SUM را به صورت SUM(D4..D6)، SUM(E4..E6) و SUM(F4..F6) بنویسیم.

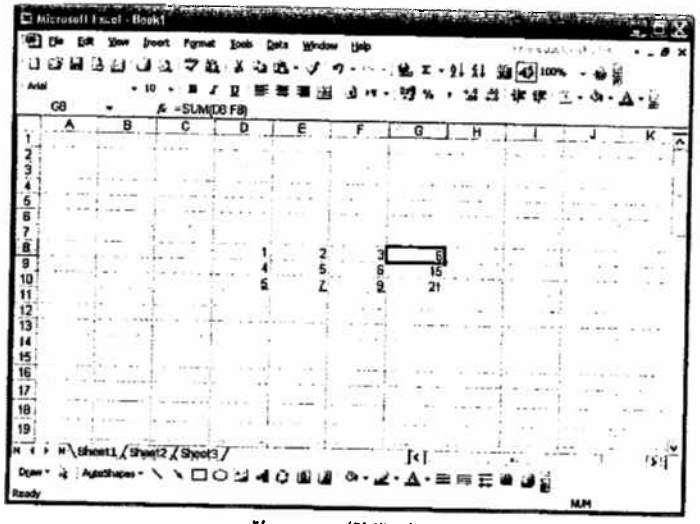
قسمت نخست فرمول، حرف مربوط به ستون است که توسط نقطه شروع آرایه مشخص می‌شود. سپس قسمت نهایی یعنی con اضافه شده و 1 کسر می‌شود. در واقع این مقدار به حروف تبدیل شده است. قسمت بعدی فرمول، شماره ردیف فوقانی است که توسط عنصر نخست آرایه مشخص شده است. از دو نقطه (..) برای جدا کردن قسمت شروع و پایان آدرس استفاده کرده‌ایم.

اکنون لازم است فرمول SUM مستقیماً در کاربرگ در زیر مجموعه سلول‌های انتخاب شده و در همان ستونی‌که فرمول، به آن استناد می‌کند قرار گیرد. برای مثال فرمول SUM(D4..D6) باید در سلول D7 قرار گیرد. با استفاده از ردیف آخر که از طریق عنصر سوم و اضافه کردن عدد 1 به آن مشخص می‌شود و با استفاده از n به جای ستون (چون حلقه For..Next روی شماره‌های ستون عمل می‌کند)، می‌توانیم به سلول مورد نظر ارجاع کنیم. کلمه sum= و دو پرانتز به formulastr چسبانده شده‌اند و یک فرمول معتبر برای جمع کردن مقادیر ستونی در مجموعه انتخاب شده، در ردیف زیر آن قرار گرفته است. این عملیات با استفاده از حلقه For..Next برای تمام ستون‌های موجود در مجموعه انتخاب شده تکرار شده است.

مقدار متغیر con، افزایش می‌یابد و نتیجه این است که در این مرحله تمام ستون‌ها یک حاصل جمع از مقادیر خود دارند. بعد با استفاده از یک حلقه دیگر For..Next که بر مبنای عناصر اول و سوم آرایه تنظیم شده است (ردیف بالایی و ردیف پایینی محدوده انتخابی کاربر) به سراغ ردیف‌ها می‌رویم. توجه کنید که به مقدار نهایی، یک واحد اضافه شده است (تا در گوشه سمت راست-پایین مجموعه انتخابی حاصل جمع کلی به دست آید) و این کار به شیوه قبل انجام شده است:

کپی آرایه که در متغیر coord نگه داشته می‌شود این بار برای حروف نام ستون به کار گرفته می‌شود چون آن‌ها در این آرایه به مقادیر عددی تبدیل نشده‌اند. سپس سلول‌هایی که در سمت راست ماتریس قرار دارند با فرمول SUM پر می‌شوند تا حاصل جمع ردیف‌ها را مشخص کنند و در سلول سمت راست- پایین نیز مجموع کلی قرار می‌گیرد.

حال در این ماتریس در سمت راست ردیف‌ها و آخرین سلول ستون‌ها، حاصل جمع ردیف‌ها و ستون‌ها را می‌بینید و در خانه سمت راست- پایین از مجموعه انتخابی نیز حاصل جمع کلی مشاهده می‌شود. یک مثال از جمع ماتریسی در تصویر ۱-۳۰ نشان داده شده است.



تصویر ۱-۳۰ نتایج جمع ماتریسی

فصل سی و یکم

فرمول‌های مطلق و نسبی

همان‌طور که می‌دانید فرمول‌ها را می‌توان با استفاده از آدرس دهی مطلق و نسبی سلول‌ها وارد کرد. برای مثال آدرس A1 در یک فرمول، نسبی تلقی می‌شود یعنی اگر فرمول را به محل جدیدی کپی کنیم، آدرس سلول در فرمول به نسبت محل جدید تغییر می‌کند. البته اگر از یک آدرس مطلق مانند $A$1 استفاده کنیم، همیشه به سلول A1 ارجاع خواهد کرد و کپی کردن آن در هرجای صفحه گسترده نیز تغییری در این وضعیت ایجاد نخواهد کرد.

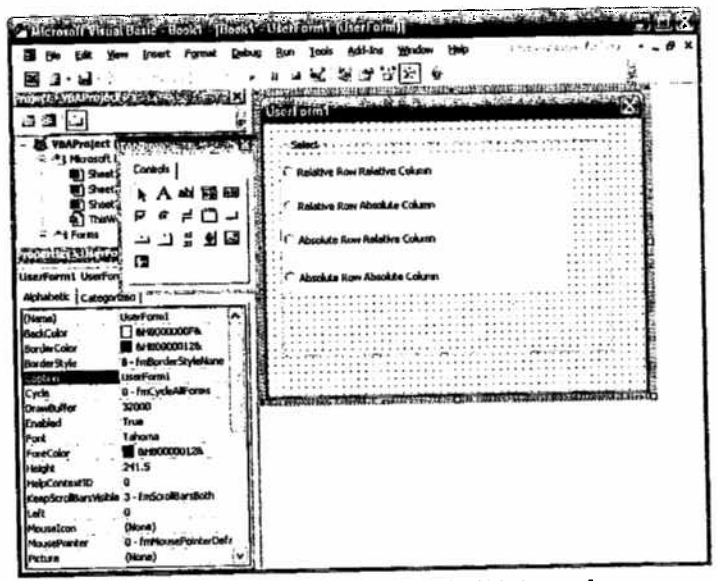
اگر قرار باشد فرمول یک سلول را از حالت نسبی به مطلق تغییر دهیم کار چندان سختی نخواهد بود اما اگر قرار باشد مجموعه بزرگی از فرمول‌ها را تغییر دهیم چه وضعیتی خواهیم داشت؟ تغییر یک به یک این فرمول‌ها به صورت دستی کار بسیار سختی خواهد بود و درصد بروز خطا را نیز افزایش می‌دهد.

می‌توانیم کدی بنویسیم تا با استفاده از ارجاع‌های مطلق در یک رویه بر مبنای انتخاب کاربر، فرمول‌ها به طور خودکار تغییر کنند و حتی می‌توانیم درباره چگونگی ظاهر شدن فرمول‌های مطلق و نسبی در صفحه گسترده، امکان انتخاب را به کاربر بدهیم. برای مثال می‌توانیم امکان انتخاب به کاربر دهیم تا هم ستون و هم ردیف مطلق باشند($A$1)، ستون نسبی و ردیف مطلق باشد (A$1) ،ستون مطلق و ردیف نسبی باشد ($A1) و یا هر دوی آن‌ها نسبی باشند (A1).

برای ارایه امکان انتخاب به کاربر درباره این قضیه، لازم است یک UserForm درج کنیم. هم‌چنین لازم است یک متغیر سراسری با نام canc (مخفف Cancel) تعریف کنیم تا کنترل کند آیا کاربر روی دکمه Cancel فرم کلیک کرده است یا خیر. برای این کار باید کد زیر را در بخش declarations ماژول خود قرار دهیم:

```
Global canc as Integer
```

می‌توانید مثال تصویر ۱-۳۱ را ببینید.



تصویر ۱-۳۱ ایجاد یک UserForm برای ارایه رابط GUI

به کارگیری یک مجموعه دکمه رادیویی که در کنترل frame قرار گرفته‌اند، امکان این را می‌دهد تا کاربر راحت‌تر ببیند چه عملیاتی در حال انجام است. دو دکمه فرمان یعنی OK و Cancel نیز اضافه شده‌اند. با تنظیم خصوصیت Value یکی از دکمه‌های رادیویی به مقدار True می‌توانیم آن را به صورت پیش‌فرض در حالت انتخاب شده قرار دهیم. این کار باعث می‌شود دیگر نگران این نباشیم که کاربر فراموش کند یکی از دکمه‌ها را انتخاب کند.

از جعبه ابزار می‌توانیم کنترل‌های مورد نیاز را با ماوس کشیده و به صفحه فرم انتقال دهیم. می‌توانیم عنوان (Caption) دکمه‌های رادیویی و فرم را با تغییر خصوصیت Caption هر کدام از کنترل‌ها عوض کنیم. اگر برای انجام این کار به مشکلی برخوردید می‌توانید به بخش ۹ مراجعه کنید.

لازم است روی دکمه‌های Command کدنویسی کنیم در غیر این صورت این دکمه‌ها کاری انجام نمی‌دهند. برای کدنویسی روی یک دکمه Command، روی آن دابل کلیک کنید تا پنجره کدنویسی آن باز شود که در حالت پیش‌فرض، نشان دهنده رویداد Click برای آن رویداد است. کد بعد را وارد کنید:

```
Private Sub CommandButton1_Click ( )
```



```
        UserForm1.Hide
        canc = 0
End Sub

Private Sub CommandButton2_Click ( )
        UserForm1.Hide
        canc = 1
End Sub
```

در هر مورد، UserForm را با استفاده از متد Hide پنهان می‌کنیم که باعث می‌شود فرم برای کاربر به صورت نامریی درآمده و اجرای برنامه به کدی که فرم را فراخوانده است، بازگردد.

متغیر canc که به عنوان یک متغیر سراسری تعریف شده است در صورت کلیک کاربر روی OK دارای مقدار 0 و در صورت کلیک کاربر روی Cancel دارای مقدار 1 خواهد بود. با کمک این مقادیر است که می‌توانیم از داخل فرم مشخص کنیم کاربر روی کدام دکمه کلیک کرده است. کد اصلی برای روال Absolute و Relative را باید به صورت زیر در یک ماژول وارد کنیم:

```
Sub conv_formula ( )
UserForm1.Show
If canc = 1 Then Exit Sub
If UserForm1.OptionButton1.Value = True Then act = xlRelative
If UserForm1.OptionButton2.Value = True Then act = xlRelRowAbsColumn
If UserForm1.OptionButton4.Value = True Then act = xlAbsolute
If UserForm1.OptionButton3.Value = True Then act = xlAbsRowRelColumn
For Each window In Windows

        For Each Worksheet In window.SelectedSheets

                For Each cell In Application.Selection

                 addr = WorkSheet.Name & "!" & cell.Address

                 If Range(addr).HasFormula = True Then Range(addr).Formula = _
                 Application.ConvertFormula(Formula:=Range(addr).Formula, _
                 fromreferencestyle:=xlA1, toreferencestyle:=xlA1, toabsolute:=act)

                Next cell

        Next worksheet
  Next window
End Sub
```

کار نخستی که کد انجام می‌دهد، نشان دادن UserForm ساخته شده ما و نمایش آن با استفاده از متد Show است. این کد سپس روی فرم به اجرای خود ادامه می‌دهد و به انتظار می‌نشیند تا زمانی که یکی از دو دکمه (OK یا Cancel) کلیک شوند. کاربر با توجه به سلیقه خود یکی از دکمه‌های رادیویی را انتخاب کرده و روی OK کلیک می‌کند که این کار باعث پنهان شدن فرم می‌شود و اجرای کد اصلی دوباره از سر گرفته می‌شود. مقدار متغیر canc کنترل می‌شود تا ببینیم آیا کاربر روی Cancel کلیک کرده است یا خیر. اگر مقدار canc برابر با 1 باشد آن‌گاه مشخص می‌شود که دکمه‌Cancel کلیک شده است و کافی است از یک عبارت Exit Sub استفاده کنیم تا مطمئن شویم هیچ اتفاقی رخ نمی‌دهد. البته اگر مقدار متغیر canc برابر 0 باشد به معنای آن است که کاربر روی OK کلیک کرده و خواهان ادامه اجرای ماکرو است.

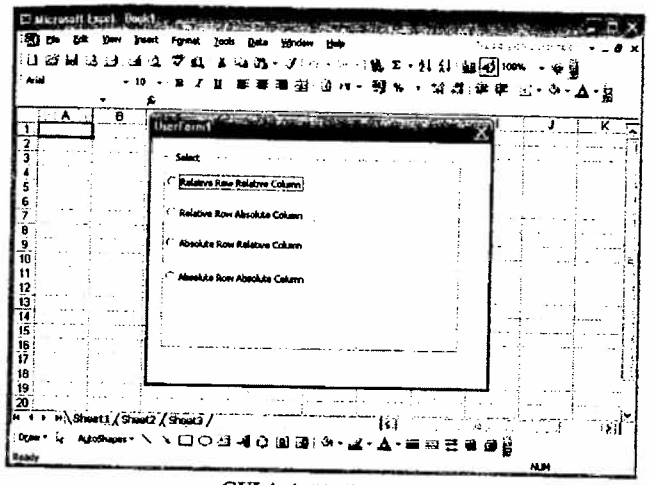
چهار خط بعدی کد مشخص می‌کند که کاربر کدام دکمه رادیویی را انتخاب کرده است. توجه کنید که تنها یکی از دکمه‌های رادیویی را می‌توانیم انتخاب کنیم چون به محض آن‌که روی یک دکمه رادیویی دیگر کلیک کنیم، مقدار True به خود گرفته و مقدار دکمه انتخاب شده نخست False می‌شود.

بسته به این که کدام دکمه رادیویی مقدار True داشته باشد، متغیری با نام act که مقدار ثابتی دارد بارگذاری می‌شود. ثابت‌های تعریف شده در این‌جا قبلاً در VBA تعریف شده‌اند و با متد ConvertFormula کار می‌کنند.

سپس کد تمام پنجره‌های موجود در مجموعه Windows را همراه با کار برگ‌های موجود در برگه‌های انتخاب شده آن پنجره‌ها در یک حلقه مرور کرده و در آخر از طریق یک حلقه دیگر، تمام سلول‌های انتخاب شده را نیز مرور می‌کنند.

متغیری با نام addr بارگذاری می‌شود که مقدار آن، نام کاربرگ انتخابی و آدرس سلول است. سپس کد کنترل می‌کند که آیا سلول مورد نظر فرمول دارد یا خیر. سپس فرمول با استفاده از متد ConvertFormula و مقدار ثابت ارایه شده توسط دکمه‌های رادیویی، تبدیل می‌شود.

کد را اجرا کنید تا UserForm شبیه تصویر ۲-۳۱ ظاهر شود.



تصویر ۲-۳۱ رابط GUI

یکی از دکمه‌های رادیویی را انتخاب کرده و روی OK کلیک کنید؛ فرمول‌های انتخاب شده را کنترل نمایید. در هر جا که مقتضی باشد، کاراکتر $ اضافه شده است.

فصل سی و دوم

رنگ کردن یک در میان ردیف‌ها و ستون‌های صفحه گسترده

گاهی اوقات وقتی ردیف‌های متعدد داده روی صفحه گسترده وجود دارند،خواندن آن‌ها مشکل می‌شود به‌ویژه وقتی که این داده‌ها خیلی طولانی باشند یا روی کاغذ چاپ شوند. یک راه حل این قضیه، رنگ کردن یک در میان ردیف‌هاست.

این کار ممکن است به نظر پیچیده آید اما همان‌طور که در فصل ۹ بررسی شد، کادرهای محاوره‌ای داخلی در برنامه‌های Office وجود دارند که کارهای مختلفی انجام می‌دهند. یکی از این کادرهای محاوره‌ای، Color Selection است که با کمک آن فرآیند رنگ آمیزی و سایه زنی بسیار ساده می‌شود. با کمک این کادر حتی لازم نیست یک UserForm طراحی کنید تا کاربر، رنگ دلخواه خود را از آن انتخاب کند چون تمام چیزهایی که لازم دارید در این کادر محاوره‌ای وجود دارد.

گرچه برای نگه داشتن کنترل Common Dialog نیز نیازی به یک UserForm خالی نداریم اما یک UserForm درج کرده و سپس کنترل Common Dialog را روی آن قرار می‌دهیم. می‌توانیم کنترل را روی هر جای فرم که می‌خواهیم قرار دهیم.

کد زیر را در یک ماژول وارد کنید:

```
Sub shade ( )
Userform3.commonDialog1.CancelError = True
Userform3.commonDialog1.Flags = &H1&
On Error GoTo errhandler
Userform3.commonDialog1.Action = 3
For Each window In Windows

        For Each Worksheet In window.SelectedSheets

          addr = Worksheet.Name & "!" & Selection.Address

          For counter =1 To Application.Selection.Rows.Count

                If counter Mod 2 = 1 Then
                  Range(addr).Rows(counter).Interior.Color = _
```

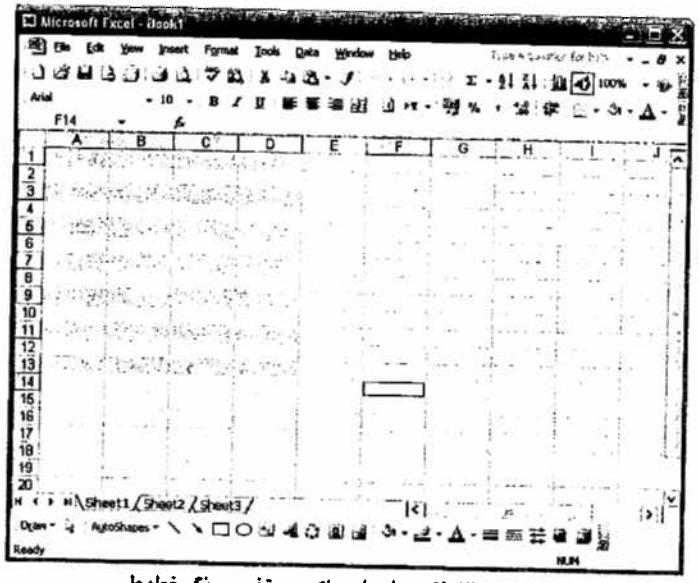
```
        Userform3.commonDialog1.Color

        End If

    Next counter
Next worksheet
Next window
Exit Sub
errhandler:
Exit Sub

End Sub
```

برای نمایش Common Dialog ، ابتدا باید مقدار خصوصیت CancelError را True تنظیم کنیم تا وقتی کاربر روی دکمه Cancel کلیک می‌کند، یک پیغام خطا ایجاد شده و عملیات Cancel به درستی انجام شود.

مقدار خصوصیت Flags را H1 (مقدار هگزادسیمال برای 1) تنظیم کرده و روال On Error را به یک روال برطرف کردن خطا مرتبط کنید. اگر مقدار خصوصیت Flags را 1 تنظیم نکرده باشیم، کادر محاوره‌ای Color Selection به نمایش در نخواهد آمد. در عمل آن‌چه روال برطرف کردن خطا انجام می‌دهد این است که وقتی کاربر روی دکمه Cancel کلیک می‌کند، یک پیغام خطا ایجاد می‌شود و روال برطرف کردن خطا باعث خروج از زیر روال مربوطه می‌شود.

چون در محدوده انتخابی کاربر تنها با ردیف‌ها سر و کار داشته و کاری به سلول‌ها نداریم، پیچیدگی روال برنامه نسبت به مثال‌های قبلی کمتر است. این روال ابتدا تمام پنجره‌های موجود در مجموعه Windows و سپس تمام کاربرگ‌های موجود در شئ SelectedSheets را در یک حلقه مرور می‌کند. سپس متغیری با نام addr بارگذاری می‌شود که مقدار آن، نام کاربرگ و آدرس انتخابی است. یک حلقه For..Next با استفاده از خصوصیت rows.count ، تمام ردیف‌های موجود در محدوده انتخابی کاربر را مرور می‌کند. با استفاده از تابع Mod ، شماره هر ردیف کنترل می‌شود تا ببینیم آیا زوج است یا فرد. اگر باقی مانده برابر با 1 باشد آن‌گاه شماره ردیف فرد است و آن ردیف باید رنگ‌آمیزی شود. ردیفی که باید رنگ شود در کنترل Common Dialog با استفاده از خصوصیت interior.color مشخص شده است.

روی صفحه گسترده،چند سلول را انتخاب کرده و سپس کد را اجرا کنید. اگر محدوده A1..D13 را انتخاب کرده باشید، نتیجه باید شبیه تصویر ۱-۳۲ باشد.



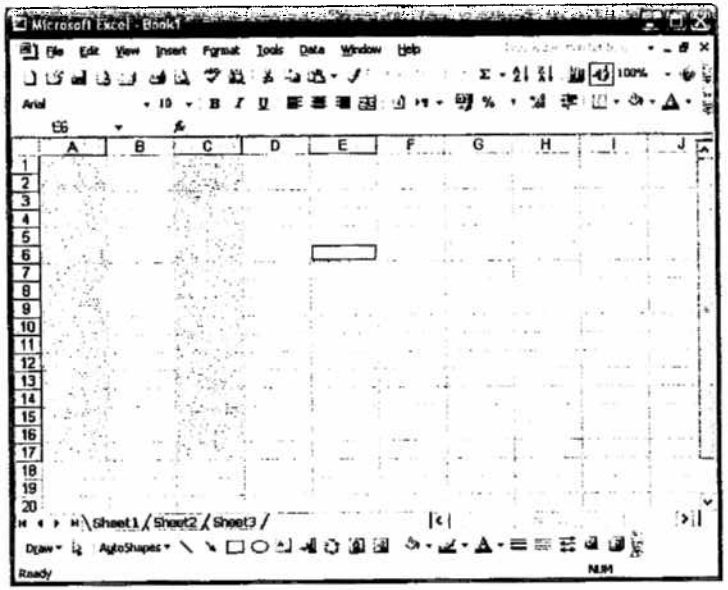
تصویر ۱-۳۲ نتیجه اجرای ماکروی تغییر رنگ خطوط

با کمی تغییر و اصلاح می‌توانیم کد را طوری تنظیم کنیم که به جای ردیف‌ها، ستون‌ها به صورت یک در میان رنگ شوند:

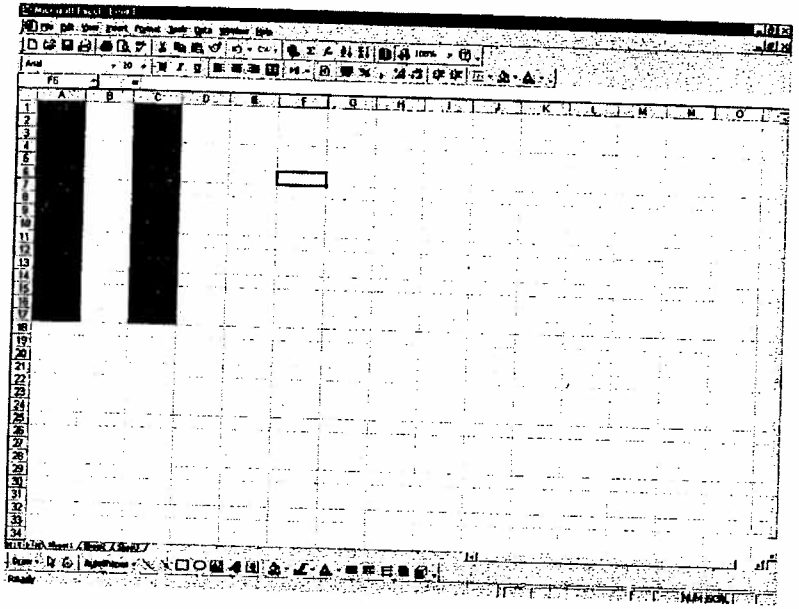
```
Sub shade1 ( )
UserForm3.CommonDialog1.CancelError = True
UserForm3.CommonDialog1.Flags = &H1&
On Error GoTo errhandler
UserForm3.CommonDialog1.Action = 3
For Each window In Windows

        For Each Worksheet In window.SelectedSheets

          addr = Worksheet.Name & "!" & Selection.Address

          For counter =1 To Application.Selection.Columns.Count

                If counter Mod 2 = 1 Then
```

```
                Range(addr).Columns(counter).Interior.Color = _
UserForm3.CommonDialog1.Color

            End If

        Next counter
        Next worksheet
Next window
Exit Sub
```

این کد بسیار شبیه کد قبلی عمل می‌کند. تنها بخشی که تغییر کرده ، حلقه For..Next است که این‌بار با استفاده از خصوصیت column.count ، ستون‌ها را کنترل می‌کند. این کد را اجرا کنید و در صورتی که محدوده A1..C17 را انتخاب کرده باشیم، خروجی شبیه تصویر ۲-۳۲ خواهد بود.



تصویر ۲-۳۲ نتیجه اجرای ماکروی تغییر رنگ ستون‌ها



تصویر ۳-۳۲

فصل سی و سوم

رنگ کردن سلول‌هایی که حاوی فرمول هستند

همیشه کار ساده‌ای نیست که ببینیم روی صفحه گسترده‌های پیچیده، کدام سلول حاوی اعداد واقعی و کدام سلول حاوی فرمول است. مثالی که در این بخش خواهید دید، به شما نشان خواهد داد که چگونه سلول‌های فرمول دار را بر مبنای انتخاب کاربر در یک صفحه گسترده و با توجه به رنگ دلخواه کاربر، می‌توان رنگ آمیزی کرد. کاربر با کنترل CommonDialog، رنگ دلخواه خود را انتخاب می‌کند. این کنترل یک کادر نمودار رنگ دارد که می‌توان از آن، رنگ دلخواه را انتخاب کرد. اگر مثال‌های بخش قبل را با دقت مرور کرده باشید حتماً از کنترل CommonDialog روی UserForm استفاده کرده‌اید.

اگر UserForm ندارید، یکی از آن‌ها را درج کرده و سپس کنترل CommonDialog را روی آن قرار دهید. کنترل را می‌توانید در هر جای فرم که می‌خواهید قرار دهید.

کد زیر را در یک ماژول وارد کنید:

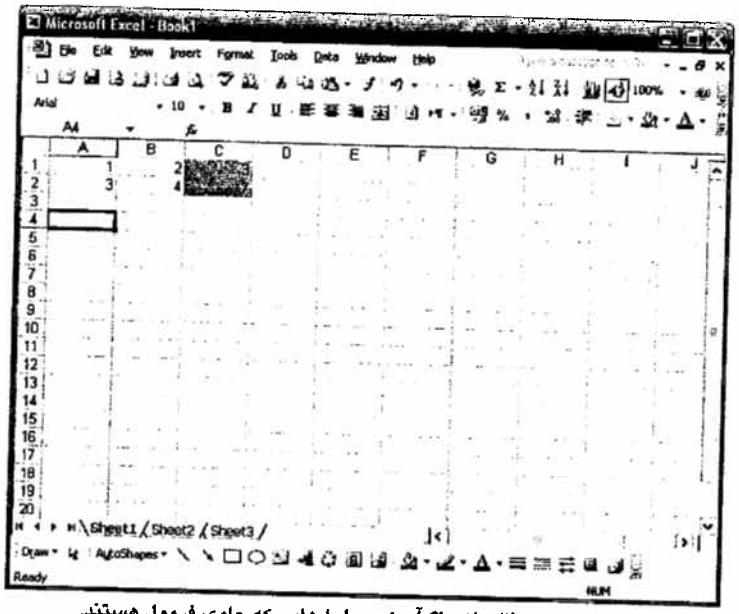
```
Sub col_cell ( )
UserForm3.CommonDialog1.CancelError = True
UserForm3.CommonDialog1.Flags = &H1&
On Error GoTo errhandler1
UserForm3.CommonDialog1.Action = 3
For Each window In Windows

        For Each Worksheet In window.SelectedSheets

                For Each cell In Application.Selection

                  addr = Worksheet.Name & "!" & cell.Address

                If    Range(addr).HasFormula  Then  Range(addr).Interior.Color  =
UserForm3.CommonDialog1.Color

                Next Cell
```

```
        Next worksheet
Next window
Exit Sub
Errhandler1:
Exit Sub

End Sub
```

برای نمایش Common Dialog،ابتدا باید مقدار خصوصیت CancelError را True تنظیم کنیم تا وقتی کاربر روی دکمه Cancel کلیک می‌کند، یک پیغام خطا ایجاد شده و عملیات Cancel انجام شود. مقدار خصوصیت Flags را H1 (مقدار هگزادسیمال برای ۱) تنظیم کرده و روال On Error را به یک روال برطرف کردن خطا مرتبط کنید. اگر مقدار خصوصیت Flag را ۱ تنظیم نکرده باشیم، کادر محاوره‌ای Color Selection به نمایش در نخواهد آمد. در عمل آن‌چه روال برطرف کردن خطا انجام می‌دهد این است که وقتی کاربر روی دکمه Cancel کلیک می‌کند، یک پیغام خطا ایجاد می‌شود و روال برطرف کردن خطا باعث خروج از زیر روال مربوطه می‌شود.

وقتی کاربر، از کادر محاوره‌ای رنگی انتخاب می‌کند، کد از طریق یک حلقه، تمام پنجره‌های موجود در مجموعه Windows و سپس تمام کاربرگ‌های موجود در مجموعه SelectedSheets را مرور می‌کند. سپس تمام سلول‌های موجود در محدوده انتخابی مرور می‌شوند. متغیری با نام addr، نام کاربرگ را همراه با آدرس سلول که با کاراکتر ! به هم چسبیده‌اند، در خود نگه می‌دارد. سپس کد کنترل می‌کند تا ببیند آیا آن سلول حاوی فرمول است یا خیر. اگر فرمول داشته باشد، با استفاده از کادر CommonDialog و ویژگی interior.color ، رنگ انتخابی کاربر را روی ستون مذکور اعمال می‌کند.

مجموعه چند سلول را که حاوی ترکیبی از اعداد و فرمول‌ها هستند از صفحه گسترده خود انتخاب کرده و سپس کد را اجرا کنید نتیجه کار باید شبیه تصویر ۱-۳۳ باشد.



تصویر ۱-۳۳ مثالی از رنگ‌آمیزی سلول‌هایی که حاوی فرمول هستند.

فصل سی و چهارم

جمع کردن چند سلول با ارجاع به یک سلول اصلی

همچنان که در فصل‌های قبلی این کتاب دیدید می‌توانیم توابع اختصاصی ایجاد کنیم تا به عنوان فرمول در صفحه گسترده به کار گرفته شوند. این توابع طوری عمل می‌کنند انگار که خود مایکروسافت آن‌ها را نوشته‌است. در این فصل به شما نشان می‌دهیم که چگونه تابع پیچیده‌تری نسبت به قبل بسازیم. وقتی کاربران در حال جمع کردن داده‌های موجود در یک ستون یا ردیف هستند، گاهی اوقات می‌خواهند تنها مقادیر خاصی را در عملیات شرکت دهند. این مقادیر معمولاً بر مبنای خصوصیت خاصی از سلول مانند خصوصیات قلم یا رنگ پس زمینه انتخاب می‌شوند.

گاهی اوقات افراد از من می‌پرسند که : آیا می‌توان اعداد Italic یا Bold را با هم جمع کرد؟ من به آن‌ها می‌گویم که در Excel روش خاصی برای این کار در نظر گرفته نشده است اما با توجه به کارآیی VBA می‌توانیم یک فرمول اختصاصی برای انجام این کار بنویسیم. ساختار دستوری انجام این کار روی صفحه گسترده به صورت زیر است:

SUMCELLSBYREF (range,reference,attribute)

- range ، مجموعه‌ای از سلول‌هاست که به شیوه‌ای مشابه با تابع استاندارد SUM تعریف شده‌اند. مقدار آن را با وارد کردن فرمول و سپس کشیدن مکان‌نما روی سلول‌های مورد نظر مشخص می کنیم.
- reference، یک ارجاع تک سلولی است که بر حسب ویژگی‌هایی که می‌خواهیم در عملیات جمع شرکت داده شوند (مثلاً آن‌هایی که Italic هستند) تنظیم می‌شود.
- attribute، رشته‌ای است که حاوی یکی از مقادیر زیر است:

جدول ۱-۳۴

ویژگی	شرح
Color	تمام سلول‌هایی را که رنگ قلم آن‌ها مشابه سلول مرجع است، با هم جمع می‌کند.

ویژگی	شرح
Italic	تمام سلول‌هایی را که خصوصیت Italic آن‌ها مشابه سلول مرجع است، با هم جمع می‌کند.
Bold	تمام سلول‌هایی را که خصوصیت Bold قلم آن‌ها مشابه سلول مرجع است، را با هم جمع می‌کند.
Size	تمام سلول‌هایی را که خصوصیت Italic قلم آن‌ها مشابه سلول مرجع است، با هم جمع می‌کند.
Underline	تمام سلول‌هایی را که خصوصیت Underline قلم آن‌ها مشابه سلول مرجع است، با هم جمع می‌کند.
Subscript	تمام سلول‌هایی را که خصوصیت Subscript قلم آن‌ها مشابه سلول مرجع است، با هم جمع می‌کند.
Superscript	تمام سلول‌هایی را که خصوصیت Superscript قلم آن‌ها مشابه سلول مرجع است، با هم جمع می‌کند.

کد زیر تمام این کارها را انجام می‌دهد. آن را در یک ماژول وارد کنید:

```
Public Function SUMCELLSBYREF(cells_to_sum As Object, r As Object, p _
As String)
Application.Volatile
total = 0
For Each cell In cells_to_sum

        If p = "bold" And cell.Font.Bold = r.Font.Bold Then

                total = total + cell.Value

        End If

        If p = "color" And cell.Font.Color = r.Font.Color Then

                total = total + cell.Value

        End If
        If p = "italic" And cell.Font.Italic = r.Font.Italic Then

                total = total + cell.Value
```



```
        End If

If p = "name" And cell.Font.Name = r.Font.Name Then

                total = total + cell.Value

        End If

If p = "size" And cell.Font.Size = r.Font.Size Then

                total = total + cell.Value

        End If

        If p = "underline" And cell.Font.Underline = r.Font.Underline Then

                total = total + cell.Value

        End If

        If p = "subscript" And cell.Font.Subscript = r.Font.Subscript Then

                total = total + cell.Value

        End If

        If p = "superscript" And cell.Font.Superscript = r.Font.Superscript Then

                total = total + cell.Value

        End If
Next cell
SUMCELLSBYREF = total
End Function
```

برای این‌که بتوانیم از داخل صفحه گسترده به این تابع دسترسی پیدا کنیم باید آن را به صورت یک تابع Public تعریف کنیم. مانند ساختار دستوری قبلی که نشان دادیم، سه پارامتر باید انتقال یابند:

جدول ۲-۳۴

نام	نوع	شرح
cells_to_sum	Object	مجموعه سلول‌هایی که قرار است با هم جمع شوند.
r	Object	آدرس سلول مرجع
p	String	نوع ویژگی، مثل size یا bold

دستور Application.Volatile اجازه می‌دهد تا مجموعه‌ای از سلول‌ها به صورت دینامیکی از داخل فرمول انتخاب شوند. این دستور مانند تابع SUM عمل می‌کند که در آن می‌توانستیم زمانی که داخل فرمول بودیم، مکان‌نما را روی سلول‌های مورد نظرمان بکشیم تا انتخاب شوند. مقدار متغیری را با نام total که حاوی مقدار حاصل جمع کلی است، برابر 0 قرار می‌دهیم. با استفاده از یک حلقه For Each...Next، تمام سلول‌های موجود در شئ cells_to_sum یک بار مرور می‌شوند.

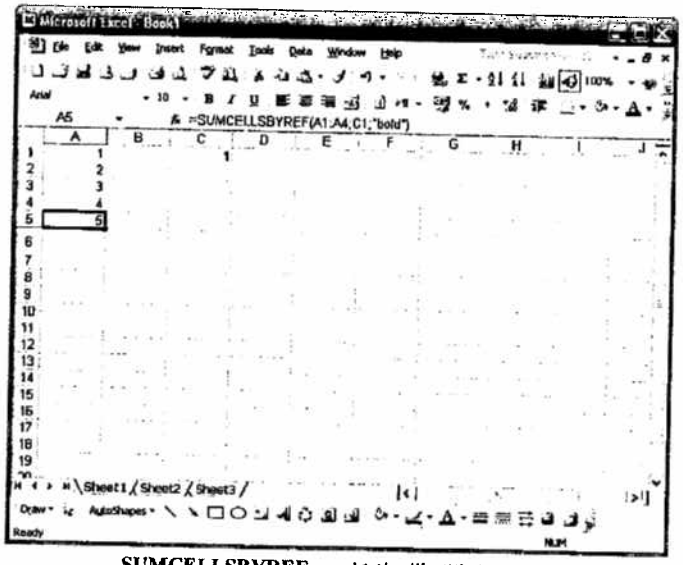
دستورات شرطی برای هر کدام از مقادیری که ممکن است p قبول کند ایجاد شده‌اند. برای مثال اگر مقدار p برابر با "bold" باشد آن‌گاه کد، این شرط را آزمایش می‌کند که آیا خصوصیت bold در سلول مورد نظر با خصوصیت bold در سلول مرجع مساوی است یا خیر. اگر این دو خصوصیت مشابه هم باشند آن‌گاه مقدار سلول به متغیر total اضافه می‌شود.

وقتی تمام سلول‌ها آزمایش شدند آن‌گاه مقدار متغیر total به متغیر SUMCELLSBYREF داده می‌شود و این مقدار، مقدار برگشتی به صفحه گسترده خواهد بود.

برای آزمایش این قضیه لازم نیست کد را اجرا کنید بلکه می‌توانید مانند حالت معمولی فرمول را برای یکی از سلول‌ها تایپ کنید:

=SUMCELLSBYREF (A1..A4,C1,"bold")

متوجه خواهید شد که اگر در نوار ابزار Formula روی آیکن Formula Paste کلیک کنیم، فرمول در بخش User Defined Formula ظاهر خواهد شد و می‌توانید از آن مانند دیگر فرمول‌ها استفاده کنید. اگر هر کدام از پارامترها را فراموش کنید آن‌گاه پیغام‌های خطای استاندارد Excel صادر خواهند شد. می‌توانید در تصویر ۱-۳۴ یک مثال از این عملیات را ببینید.



تصویر ۱-۳۴ مثالی از کاربرد SUMCELLSBYREF

فصل سی و پنجم

تغییر سراسری مجموعه‌ای از مقادیر

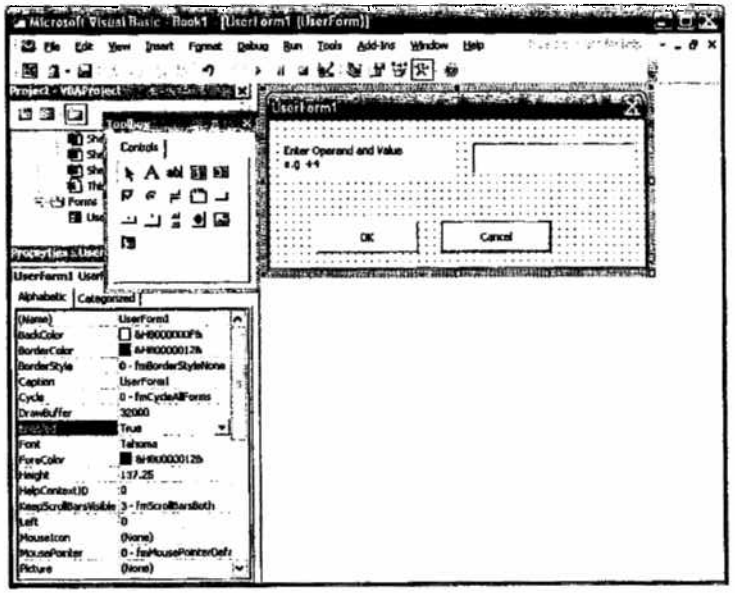
وقتی مقدار زیادی داده روی صفحه گسترده وجود داشته باشد ممکن است بخواهیم با تنظیم یک مقدار یا درصد، مجموعه‌ای از اعداد را تغییر دهیم. برای مثال ممکن است بخواهیم تمام اعداد موجود در مجموعه انتخابی به میزان دو واحد افزایش یابند. می‌توان مقدار سلول‌ها را در Excel، به‌طور جداگانه تغییر داد، اما نمی‌توان یک عملیات سراسری روی مجموعه‌ای از سلول‌ها اجرا کرد.

با کمک کدی که در این فصل خواهید دید، امکان به کارگیری یک فاکتور سراسری (مثلاً اضافه کردن عدد 10 به تمام سلول‌ها یا کسر 5 درصد از تمام سلول‌ها) به کاربر داده می‌شود. برای اجرای این عملیات ابتدا باید یک UserForm ساخته شود تا به کاربر امکان وارد کردن یک فاکتور داده شود. با انتخاب Insert | UserForm از منوی کد، یک UserForm جدید درج کنید.

در فرم لازم است کنترل‌های زیر را درج کنیم:

یک کنترل Lable برای این‌که شرح مقدار ورودی را نشان دهد؛ یک کادر متنی که ورودی کاربر را می‌گیرد و دو دکمه Command که برای OK و Cancel استفاده می‌شوند.

عنوان فرم را می‌توانیم با کلیک روی خود فرم و سپس تغییر خصوصیت Caption در پنجره Properties عوض کنیم. کنترل‌ها را از جعبه ابزار بر روی فرم بکشید تا UserForm شبیه تصویر ۱-۳۵ شود.



تصویر ۱-۳۵ تعریف یک UserForm برای تنظیم مقادیر سراسری

لازم است برای دکمه‌های OK و Cancel کدنویسی کنیم چون در غیر این صورت این دکمه‌ها کاری انجام نخواهند داد. هم‌چنین لازم است یک متغیر سراسری با نام canc در بخش declarations ماژول تعریف شده باشد تا مقداری را که مشخص کننده کلیک کاربر روی OK یا Cancel است، در خود نگه دارد.

تمام هدف UserForm، گرفتن یک رشته پارامتری است که قرار است روی مجموعه‌ای از سلول‌ها اعمال شود. اگر بخواهیم تمام اعداد به میزان سه واحد افزایش یابند باید 3+ را در کادر متنی وارد کنیم:

```
Global canc As Integer
```

کد زیر را در رویداد Click برای دکمه‌های فرمان OK و Cancel وارد کنید. این کد را در ماژول قرار ندهید، بلکه دوبار روی دکمه OK کلیک کنید تا به پنجره کدنویسی برای رویداد Click در دکمه OK وارد شوید:

```
Private Sub CommandButton1_Click ( )
```



```
        UserForm2.Hide
        canc = 0
End Sub

Private Sub CommandButton2_Click ( )
        UserForm2.Hide
        canc = 1
End Sub
```

هر کدام از این روال‌ها، UserForm را با استفاده از متد Hide پنهان می‌کنند. این به معنای آن است که اجرا به کد اصلی که فرم را نمایش می‌داد انتقال یافته است. اگر کاربر روی OK کلیک کند مقدار canc برابر با 0 (False) تنظیم می‌شود و در غیر این صورت مقدار آن برابر با 1 (True) تنظیم می‌شود. با توجه به این‌که متغیر canc یک متغیر سراسری است می‌توانید از هرجا به مقدار آن دسترسی پیدا کنید. سپس کد زیر را در یک ماژول وارد کنید:

```
Sub change_val ( )
Userform2.Show
If canc = 1 Then Exit Sub
op = UserForm2.TextBox1.Text
Dim addr As String
For Each window In Windows

    For Each Worksheet In window.SelectedSheets

        For Each cell In Application.Selection

          addr = Worksheet.Name & "!" & cell.Address

          On Error Resume Next
          If Range(addr).Value <> ""And IsNumeric(Range(addr).Value) and
Range(addr).HasFormula = False Then

              Range(addr).Value = "=" & Val(Range(addr).Value) & op

              Range(addr).Copy

              Range(addr).PasteSpecial xlPasteValues

        End If

        Next cell
            Next worksheet
```

```
Next window
End Sub
```

اولین کاری که این کد انجام می‌دهد، نمایش UserForm ساخته شده است تا به کاربر اجازه دهد اطلاعات دلخواهش را در کادر متنی وارد کرده و روی دکمه‌های OK یا Cancel کلیک کند.

پس از این‌که فرم با توجه به یکی از رویدادهای Click، پنهان شد کد، اقدام به بررسی مقدار متغیر canc می‌نماید تا مشخص کند آیا دکمه Cancel کلیک شده است یا خیر. اگر مقدار متغیر canc برابر I بود (یعنی دکمه Cancel کلیک شده است) آن‌گاه تنها به یک دستور Exit Sub نیاز خواهیم داشت.

اگر روی OK کلیک شده باشد، مقداری که وارد کادر متنی شده است در متغیر op ذخیره می‌شود.

متغیری با نام addr را از نوع String تعریف می‌کنیم. سپس کد، تمام پنجره‌های موجود در مجموعه Windows و تمام کاربرگ‌های موجود را در یک حلقه مرور می‌کند. متغیر addr در حالی بارگذاری می‌شود که مقدار آن برابر با نام کاربرگ فعال و آدرس سلول جاری است که با کاراکتر ! به هم الحاق شده‌اند. سپس مقدار سلول تعریف شده توسط addr کنترل می‌شود تا مشخص شود که خالی نیست یعنی یک مقدار عددی در آن قرار دارد و البته نباید تهی نیز باشد.

اگر یک عملوند و یک مقدار عددی را روی یک سلول اعمال کنیم باید از قبل یک مقدار عددی در آن سلول وجود داشته باشد و فرمولی نیز در آن سلول نباشد چون مقدار فرمول به خاطر تغییر در سلول‌های عددی به صورت خودکار به روز درآورده می‌شود.

اگر تمام شرایط تحقق پیدا کند آن‌گاه رشته‌ای به عنوان مقدار سلول در آن قرار می‌گیرد که در ابتدای آن علامت = است سپس مقدار فعلی سلول همراه با متغیر op که جزییات چگونگی تغییر سلول را در خود دارد(مثلا "%90*")، قرار می گیرند.

در یک مجموعه از سلول‌های صفحه گسترده، داده‌هایی را همراه با چند فرمول قرار دهید و سپس کل مجموعه را انتخاب کنید. کد را اجرا کرده و در کادر متنی موجود روی UserForm، مقدار %90* را وارد کنید. روی OK کلیک کنید تا تمام سلول‌ها دارای 90% مقدار قبلی خود شوند. سلول‌های فرمول‌دار بدون تغییر باقی می‌مانند اما بر مبنای سلول‌های دیگر، مقدار واقعی 90% را نشان می دهند.

برای برگرداندن مقادیر قبلی به سلول‌ها کافی است برعکس این کار را انجام دهیم: مقدار 90% را در کادر متنی وارد کنید تا همه چیز به حالت اول بازگردد.

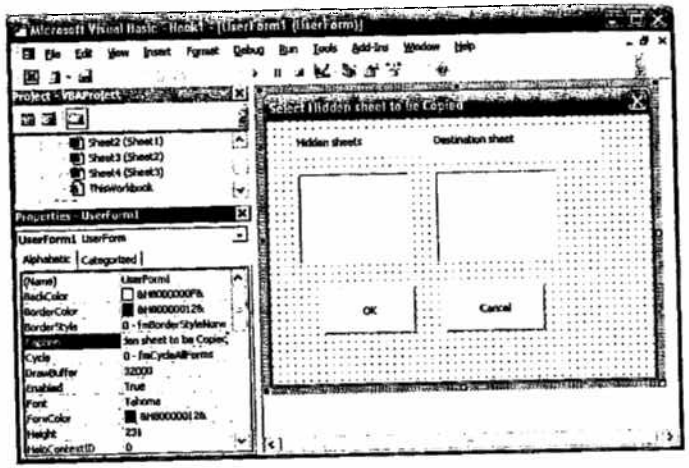
فصل سی و ششم

نمایش برگه‌های پنهان شده بدون وارد کردن کلمه عبور

در این فصل، قدرت واقعی VBA به شما نشان داده خواهد شد. اگر یک کاربرگ را پنهان کرده (Hidden) وبرای آن کلمه عبور تعیین کنید فرض بر این است که اجازه نخواهید داشت آن را ببینید. برنامه‌هایی در بازار وجود دارند که قفل کلمه‌های عبور را می‌شکنند اما نسخه‌های جدید Excel طوری کلمه عبور را پنهان می‌سازند که این برنامه‌ها نمی‌توانند قفل آن را بشکنند. با دانستن این مطالب ممکن است شما فرض کنید که کاربرگ‌های ایمنی در اختیار دارید.

اما در مدل شئ Excel، مایکروسافت یک راه فرار برای این قضیه در نظر گرفته است. درست است که هیچ مرجعی برای دیدن کلمه‌های عبور وجود ندارد اما می‌توانیم وارد مجموعه Worksheets شده و کاربرگ‌های پنهان را پیدا کنیم. وقتی نام چنین کاربرگی مشخص شد می‌توان به سادگی محتویات آن را در یک کاربرگ مریی کپی کرده و همه چیز را دید.

در این کد نیازمند آن هستیم تا یک UserForm جدید درج کنیم تا رابطی بین کد و کاربر ایجاد شود. این کار را از منوی VBE با انتخاب Insert | UserForm انجام می‌دهیم. UserForm باید شبیه تصویر ۱-۳۶ باشد.



تصویر ۱-۳۶ طراحی یک UserForm برای مرور برگه‌های پنهان

این فرم دارای دو کادر لیست (ListBox) است که یکی از آن‌ها برگه‌های پنهان و دیگری برگه‌های مقصد (که مریی هستند)را نشان می‌دهند. کاربر، یک برگه پنهان شده را انتخاب می‌کند سپس در لیست مریی روی یک کاربرگ دیگر به عنوان مقصد انتقال اطلاعات کاربرگ پنهان کلیک خواهد کرد.

لازم است در UserForm از دو کنترل Label استفاده کنیم تا عنوان کادرهای فهرست را در آن‌ها درج نماییم تا کاربر متوجه شود هر لیست چه چیزی نشان می‌دهد. برای عملیات‌های OK و Cancel هم نیازمند دو دکمه Command هستیم. با کلیک روی فرم و تغییر خصوصیت Caption در پنجره Properties می‌توانیم عنوان فرم را هم تغییر دهیم.

در این مثال،UserForm متفاوت از مثال‌های قبلی است چون این فرم پیش از به نمایش درآمدن باید با اطلاعاتی پر شود. اگر این کار را انجام ندهیم، کاربر تنها دو کادر لیست خالی را خواهد دید که زیاد مفید نخواهند بود.

برای نوشتن کد این عملیات، روی فرم دابل کلیک کنید تا وارد پنجره کد نویسی UserForm شوید. اولین قسمت کد را باید در رویداد UserForm Activate بنویسیم. این رویداد هر وقت فرم برای بار نخست به نمایش درمی‌آید، فراخوانده می‌شود. روی ماژول، ابتدا در منوی بازشوی سمت چپ - بالا، UserForm و در منوی بازشوی سمت راست - بالا، Activate را انتخاب کرده و کد زیر را وارد کنید:

```
Private Sub UserForm_Activate ( )
```



```
ListBox1.Clear
ListBox2.Clear
For Each Worksheet In ActiveWorkbook.Worksheets
        If Worksheet.Visible = False Then

          ListBox1.AddItem Worksheet.Name

        End If
Next
For Each Worksheet In ActiveWorkbook.Worksheets
        If Worksheet.Visible = True Then

          ListBox2.AddItem Worksheet.Name

        End If
Next
End Sub
```

اولین کاری که کد انجام می دهد، اطمینان از خالی بودن کادرهای لیستی با استفاده از متد Clear است.

سپس کد تمام کاربرگ‌های موجود در ActiveWorkbook Worksheets را از طریق یک حلقه مرور می‌کند. توجه کنید که در اینجا تنها از کارپوشه فعال استفاده کرده و کاری به دیگر کارپوشه‌ها که احتمالاً بارگذاری شده‌اند نداریم. اگر از تمام کارپوشه‌ها استفاده کنیم ممکن است مشکلاتی در نمایش دادن آن‌ها در کادرهای لیستی به وجود آید و این امر منجر به بروز مشکلاتی در آدرس دهی شود.

کد با کنترل خصوصیت Visible مشخص می‌کند که آیا کاربرگ پنهان شده است یا خیر. اگر مقدار خصوصیت Visible، False باشد کد از متد AddItem در کادر لیست استفاده می‌کند تا نام کاربرگ را به لیست اول اضافه کند.

سپس مجدداً به سراغ کاربرگ‌ها آمده و کنترل می‌کند که آیا کاربرگ بعدی، مریی است یا خیر. اگر کاربرگی مریی باشد، کد با کمک متد AddItem نام آن را به لیست دوم اضافه می‌کند. وقتی کد، UserForm را باز می‌کند کاربر دو لیست را خواهد دید که در یکی از آن‌ها تمام برگه‌های پنهان و در دیگری تمام برگه‌های مریی نشان داده شده‌اند.

هم‌چنین لازم است برای دکمه‌های OK و Cancel هم کدنویسی کنیم. برای این کار یک متغیر سراسری با نام canc را در بخش Declarations تعریف می کنیم:

```
Global canc as Integer
```

سپس می‌توانیم کد زیر را در ماژول UserForm اضافه کنیم:

```
Private Sub CommandButton1_Click ( )
```

```
UserForm1.Hide

canc = 0

End Sub

Private Sub CommandButton2_Click ( )

UserForm1.Hide

canc = 1

End Sub
```

با کمک این کد، هم فرم پنهان می‌شود و هم اجرای کد مجدداً به بخشی که فرم از آنجا نمایش داده شد باز می‌گردد.

برای برقراری ارتباط با کاربر لازم است کد زیر را به ماژول اضافه کنیم:

```
Sub hidden_sheets ( )
UserForm1.Show
If canc = 1 Then Exit Sub
s1 = UserForm1.ListBox1.Text
s2 = UserForm1.ListBox2.Text
If s1 = "" Or s2 = "" Then Exit Sub
Range(s1 & "!a1.iv65536").Copy
Range(s2 & "!a1").PasteSpecial
Range(s2 & "!a1").Select
End Sub
```

عملیاتی که این کد انجام می‌دهد بسیار ساده است. در خط اول کد با استفاده از متد Show ، UserFormبه نمایش درمی‌آید. این، به کاربر اجازه می‌دهد تا نام تمام برگه‌های پنهان را ببیند و پس از انتخاب یکی از آن‌ها، محل کپی کردن محتویات آن‌ها را هم مشخص کند.

وقتی فرم پنهان شده و اجرای کد به روال hidden_sheets انتقال یافته باشد آن‌گاه مقدار متغیر canc کنترل می‌شود. اگر کاربر روی Cancel کلیک کرده باشد، اجرای رویه خاتمه یافته و چیزی رخ نمی‌دهد. اما اگر کاربر روی OK کلیک کرده باشد، رویه hidden_sheets اجرا می‌شود.

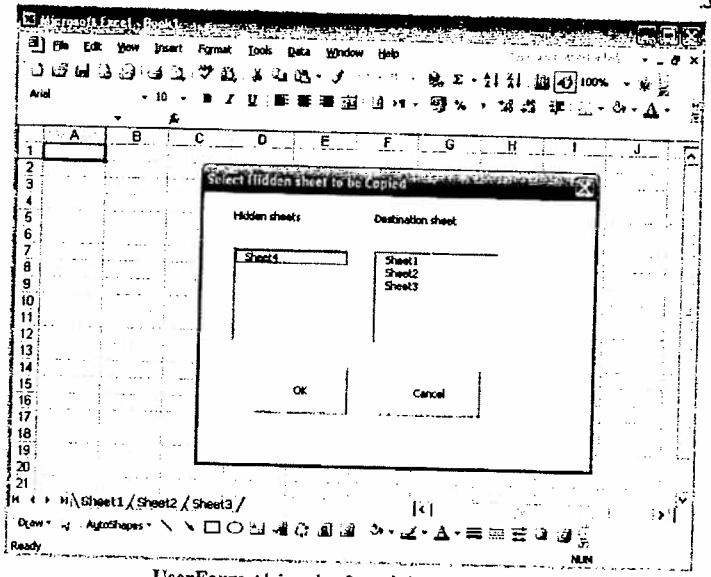
دو متغیر به نام‌های s1 و s2 بارگذاری می‌شوند که مقادیر موجود در آن‌ها،گزینه انتخابی از کادرهای لیستی است. اگر در یکی از لیست‌ها یا هر دوی آن‌ها انتخابی صورت نگرفته باشد، اجرای رویه متوقف شده و اتفاقی هم نخواهد افتاد.

کل برگه پنهان شده با استفاده از متد Copy به حافظه Clipboard انتقال می‌یابد و کلمه a1.iv65536 با استفاده از کاراکتر ! به نام برگه متصل می‌شود. این امر باعث می‌شود اطمینان حاصل کنیم که تمام سلول‌های برگه کپی شده‌اند. سپس با استفاده از متد PasteSpecial، اطلاعات موجود در Clipboard برگه مقصد چسبانده می‌شود.

در کد مثالی ما، سلول A1 به عنوان مقصد است. پس سلول A1 روی برگه مقصد انتخاب می‌شود. این کار باعث می‌شود آن برگه به عنوان برگه فعال درآید.

برای آزمایش این عملیات، برگه‌ای را در یک کارپوشه ساخته و داده‌هایی روی آن قرار دهید. سپس با استفاده از Format | Sheet | hide از منوی صفحه گسترده، آن برگه را پنهان کنید. اکنون با استفاده از Tools | Protection | Protect Workbook تمام کارپوشه را با یک کلمه عبور محافظت کنید.

اگر بخواهید برای ظاهر شدن برگه پنهان شده تلاش کنید، امکان موفقیت نخواهید داشت. در حال حاضر در منوی صفحه گسترده، گزینه Format | Sheet | Unhide غیر فعال شده است و بدون دانستن کلمه عبور امکان مشاهده آن برگه وجود ندارد. کد را اجرا کنید تا صفحه‌ای مانند تصویر ۲-۳۶ را ببینید.



تصویر ۲-۳۶ نمایش برگه‌های پنهان UserForm

برگه پنهان را انتخاب کرده و سپس برگه‌ای را به عنوان مقصد مشخص کنید. اگر روی OK کلیک کنید محتویات برگه پنهان در برگه مقصد ظاهر می‌شود. دقت کنید که این رویه، خالی بودن برگه مقصد را کنترل نمی‌کند و تمام داده‌های موجود در آن برگه کاملاً از بین خواهند رفت.

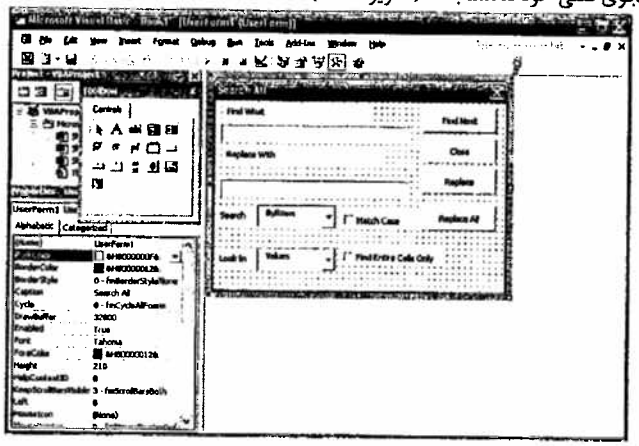
فصل سی و هفتم

جستجوی چندین برگه و کارپوشه

Excel یک امکان جستجو دارد که با انتخاب Edit | File از منوی صفحه گسترده می‌توانیم به آن دسترسی داشته باشیم اما این متد فقط کاربرگ فعلی را جستجو می‌کند. می‌توان یک گروه از کاربرگ‌ها را انتخاب کرده و در آن‌ها به جستجو بپردازیم. اما ابتدا باید گزینش انجام دهیم در ضمن نمی‌توانیم در تمام کارپوشه‌ها، جستجو انجام دهیم. به کارگیری امکانات جستجوی استاندارد می‌تواند پرزحمت و پردردسر باشد اما می‌توانیم کدی بنویسیم که عملیات جستجوی مخصوص و دلخواه ما را انجام دهد و بدون این‌که مجبور باشیم ابتدا آن‌ها را انتخاب کنیم، در چندین کاربرگ یا کارپوشه جستجو کنیم.

به دلیل انعطاف‌پذیری مدل شئ Excel، تمام خصوصیات و امکانات جستجو در آن تعبیه شده است. تمام آنچه نیاز داریم این است که کاری کنیم تا آن‌ها روی کاربرگ‌ها و کارپوشه‌ها جواب دهند.

اولین چیزی که لازم داریم، یک UserForm برای برقراری ارتباط با کاربر است. چون می‌خواهیم همان عملیات و کارآیی امکانات جستجوی موجود را در اختیار داشته باشیم، UserForm باید کاملاً نزدیک به فرم جستجوی فعلی خود Excel باشد (تصویر ۱-۳۷).



تصویر ۱-۳۷ طراحی UserForm برای امکانات Search

چهار دکمه فرمان وجود دارند که هر کدام به ترتیب برای Replace, Close, Next و Replace All به کار می‌روند. برای متن جستجو و متن جایگزین، کادرهای متنی برای وارد کردن متن مذکور وجود دارد و برچسب‌هایی (Label) هم برای شرح عملکرد کادرهای متنی وجود دارند.

دو Combo Box هم برای انتخاب نوع جستجو و دو برچسب برای شرح عملکرد این کادرها وجود دارند. در آخر هم دو کادر انتخابی (check box) برای تطبیق بزرگ و کوچک بودن حروف یا جستجو در کل سلول‌ها نیز ارایه شده‌اند.

کنترل‌ها از جعبه ابزار روی فرم کشیده شده‌اند. خصوصیت Caption فرم را با کلیک روی آن در پنجره Properties می‌توان تغییر داد.

لازم است در ماژول، یک متغیر سراسری با نام canc ایجاد کنیم تا عملیات کاربر را بر مبنای این‌که روی کدام دکمه فرمان کلیک کرده است در خود نگه دارد. این متغیر در بخش Declarations ماژول قرار می‌گیرد:

```
Global canc As Integer
```

چون چهار دکمه فرمان وجود دارند، متغیر canc باید دامنه بزرگ‌تری از مقادیر را نسبت به مثال‌های قبلی در خود نگه دارد.

اکنون لازم است کد زیر را به ماژول UserForm اضافه کنیم تا اجرا شود. اگر روی هر کدام از کنترل‌های موجود روی فرم دوبار کلیک کنیم وارد پنجره کدنویسی فرم خواهیم شد. کد به شرح زیر است:

```
Private Sub CommandButton1_Click ( )
FindDialog.Hide
canc = 0
End Sub

Private Sub CommandButton2_Click ( )
FindDialog.Hide
canc =1
End Sub

Private Sub CommandButton3_Click ( )
FindDialog.Hide
canc = 2
End Sub

Private Sub CommandButton3_MouseMove(ByVal Button As Integer, _
ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single)
ComboBox2.Visible = False
```



```
Label3.Visible = False
End Sub

Private Sub CommandButton4_Click ( )
FindDialog.Hide
canc = 4
End Sub

Private Sub CommandButton4_MouseMove(ByVal Button As Integer, _
ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single)
ComboBox2.Visible = False
Label3.Visible = False
End Sub

Private Sub TextBox2_MouseMove(ByVal Button As Integer, _
ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single)
ComboBox2.Visible = False
Label3.Visible = False
End Sub

Private Sub UserForm_Activate ( )

ComboBox1.Clear

ComboBox1.AddItem "ByRows"
ComboBox1.AddItem "ByColumns"

ComboBox2.Clear

ComboBox2.AddItem "Formulas"
ComboBox2.AddItem "Values"
ComboBox2.AddItem "Comments"

End Sub

Private Sub UserForm_MouseMove(ByVal Button As Integer, _
ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single)
ComboBox2.Visible = True
Label3.Visible = True
End Sub

Private Sub UserForm_Terminate ( )
canc = 1
End Sub
```

این UserForm نسبت به مثال‌های قبلی این کتاب، کمی پیچیده‌تر است چون رویدادهای بسیار زیادی در آن به کار گرفته شده است. هر رویه را به نوبت بررسی خواهیم کرد:

Command Button1، نشان دهنده دکمه Find Next است. وقتی روی این دکمه کلیک کنیم، UserForm(FindDialog) پنهان می‌شود و اجرا به رویه‌ای برمی‌گردد که فرم را به نمایش در آورده است. مقدار متغیر سراسری canc برابر صفر تنظیم می‌شود.

Command Button2 نشان دهنده یک دکمه Close است. وقتی روی این دکمه کلیک کنیم، UserForm(FindDialog) پنهان می‌شود و اجرا به رویه‌ای بازمی‌گردد که فرم را به نمایش در آورده است. مقدار متغیر سراسری canc برابر 1 تنظیم می‌شود.

Command Button3 نشان دهنده دکمه Replace است. وقتی روی این دکمه کلیک کنیم، UserForm(FindDialog) پنهان می‌شود و اجرا به رویه‌ای بازمی‌گردد که فرم را به نمایش در آورده است. مقدار متغیر سراسری canc برابر 2 تنظیم می‌شود.

Command Button4 نشان دهنده دکمه Replace All است. وقتی روی این دکمه کلیک کنیم، پنهان می‌شود و اجرا به رویه‌ای بازمی‌گردد که فرم را به نمایش در آورده است. مقدار متغیر سراسری canc برابر 3 تنظیم می‌شود.

برای Command Button3، رویداد Mouse Move نیز روی آن دکمه به کار گرفته شده است چون وقتی از گزینه Replace استفاده می‌شود، پارامترهای LookIn (که در تصویر ۱-۳۷ نشان داده شده است) مورد نیاز نخواهد بود. این پارامترها توسط ComboBox2 و Lable3 نشان داده شده‌اند. این کد مقدار متغیرهای مربی ComboBox2 و Lable3 را False تنظیم می‌کند تا آن‌ها را روی فرم نبینیم. همین امر برای رویداد Mouse Move در Command Button4 (که Replace All را نشان می‌دهد) نیز رخ می دهد.

هم‌چنین اگر کاربر متنی را در TextBox2 وارد کند، رویداد Mouse Move باعث ناپدید شدن ComboBox2 و Lable3 از فرم می‌شود. پارامترهای LookIn توسط ComboBox و Lable تعریف می‌شوند و این پارامترها برای Replace مورد نیاز نیستند.

وقتی UserForm فعال می‌شود، کادرهای کمبو لازم است برای انتخاب کاربر، مقادیری در لیست‌های خود داشته باشند. این امر توسط رویداد UserForm_Activate انجام می‌شود و متد AddItem، گزینه‌هایی را به لیست‌های ComboBox اضافه می‌کند. ابتدا با استفاده از متد Clear، کادرها پاک می‌شوند.

هر جا رویدادهای ماوس Move روی دکمه‌های فرمان و کادرهای متنی استفاده شده باشند تا کنترل‌های خاص کم‌رنگ شوند (غیر فعال شوند)، رویداد ماوس Move روی فرم با تنظیم مقدار خصوصیت Visible برابر با True، باعث می‌شود آن‌ها مجدد مریی شوند.

در آخر اگر فرم بسته شود، رویداد UserForm_Terminate فرا خوانده شده و باعث می‌شود مقدار متغیر canc برابر با 1 تنظیم شود و این به معنای آن است که دکمه Close کلیک شده است و هیچ عملیات دیگری رخ نخواهد داد.

سپس لازم است کد زیر را در همان ماژولی که دیگر متغیرهای سراسری را در آن‌جا درج کرده‌ایم، وارد کنیم:

```
Sub findsheet ( )
flag = 0
sflag = 1
temp = ""
Dim a As Range, s As Worksheet, w As Workbook

FindDialog.TextBox1.Text = ActiveCell.Value
  Set a = ActiveCell

For Each w In Workbooks
        Set a = w.Sheets(1).Range("iv65535")

        For n = 1 To w.Worksheets.Count
Loopa:

        If sflag = 1 Then FindDialog.Show

        sflag = 0

        If canc = 1 Then FindDialog.Hide : canc = 0 : Exit Sub

        On Error Resume Next

        sstr = FindDialog.TextBox1.Text

        rep = FindDialog.TextBox2.Text
                If canc = 2 Then
```

```
            a.Replace sstr, rep
            sflag = 1
            GoTo loopa
            End If

        On Error Resume Next

        If FindDialog. ComboBox2.Text = "Formulas" Then li = -4123

        If FindDialog. ComboBox2.Text = "Values" Then li = -4163

        If FindDialog. ComboBox2.Text = "Comments" Then li = -4144

        If FindDialog. ComboBox1.Text = "ByRows" Then so = 1

        If FindDialog. ComboBox1.Text = "ByColumns" Then so = 2

        mc = FindDialog.CheckBox1.Value

        la = FindDialog.CheckBox2.Value

        If la = True Then lat = 1 Else lat = 2
        If canc = 4 Then

                        p = w.Sheets(n).Range("a1", "iv65535")._
Replace(sstr rep, lat, so, mc)
        End If
        If a.Address = "" Then Set a = Sheets(n).Range("iv65535")

        Set a = w.Sheets(n).Range("a1", "iv65535"). _
Find(sstr, a, li, lat, so, ,mc)
        If a.Adress = "" Then sflag = 0 Else sflag = 1

        If InStr(temp, w.Name & Sheets(n).Name & a.Address) Or _
        a.Address = "" Then sflag = 0: GoTo bypass

        w.Activate

        Sheets(n).Activate

        a.Activate

        temp = temp & " " & w.Name & Sheets(n).Name  & a.Address

        GoTo loopa
```



```
bypass:

        temp = ""
  Next n
Next w
MsgBox "No Further occurrences of " & sstr, vbInformation
Exit Sub

End Sub
```

اولین کاری که این کد انجام می‌دهد راه‌اندازی متغیرهای flagها، رشته‌های موقت، دامنه(range)، کارپوشه و کاربرگ است.

کادر متنی روی UserForm برای "Find what"، به عنوان مقدار سلول فعال در نظر گرفته می‌شود. این سلولی است که در حال حاضر وقتی UserForm به نمایش درمی‌آید، مکان‌نما روی آن قرار دارد و به طور پیش فرض، مقدار آن سلول در کادر متنی "Find what" را دارد. متغیر a که از نوع range تعریف شده است به عنوان سلول فعال تنظیم شده است.

کد با استفاده از حلقه For Each..Next، تمام کارپوشه‌ها را مرور می‌کند. متغیر a، آخرین سلول در نخستین کاربرگ شاخص گذاری شده در کارپوشه قرار می‌گیرد.

سپس کد از حلقه For Each..Next برای مرور تمام کاربرگ‌ها در کارپوشه استفاده می‌کند. در این مرحله از یک برچسب با نام loopa استفاده می‌شود تا در شرایط خاص، کد بتواند به آن ارجاع کند.

اگر متغیر sflag برابر 1 باشد، UserForm جستجو به نمایش درمی‌آید، سپس کاربر می‌تواند انتخاب خود را درباره چیزی که جستجو گردد و چیزی که جایگزین آن شود وارد کند.

بسته به این‌که کدام عملیات انجام شود، یکی از چهار مقدار فوق‌الذکر به متغیر canc اختصاص می‌یابد.

متغیر sflag مساوی صفر تنظیم می‌شود. این متغیر تعیین می‌کند که آیا UserForm به نمایش در آید یا خیر. این امر وابسته به شرایط خاص است و در آغاز، نمی‌خواهیم فرم، دوباره به نمایش درآید مگر آن‌که متن جستجو وارد شده باشد که در این صورت مقدار متغیر sflag برابر صفر است.

اگر canc برابر 1 باشد به معنای آن است که کاربر روی دکمه Close کلیک کرده است سپس UserForm پنهان می‌شود، مقدار متغیر canc برابر صفر تنظیم می‌شود و اجرای زیر روال بدون انجام عملیاتی به پایان می‌رسد.

سپس دو متغیر برای نگهداری مقادیر "Find what" و "Replace whit" روی UserForm تعریف می‌شوند. متغیر sstr مقدار "Find what" و متغیر rep مقدار "Replace whit" را در خود نگه می‌دارد.

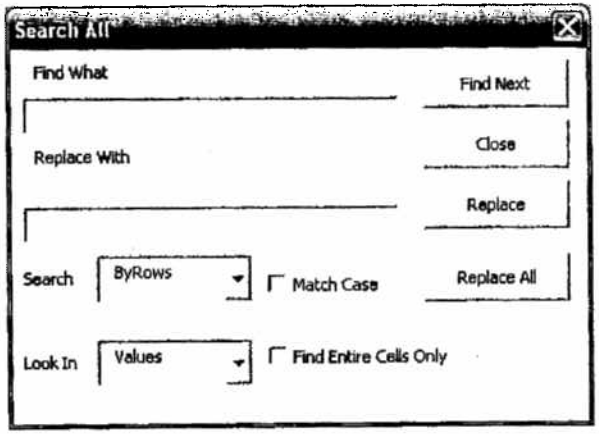
اگر کاربر روی دکمه Replace روی فرم، کلیک کند مقدار متغیر canc برابر 2 تنظیم می‌شود. این به معنای آن است که کاربر، یک مورد از رشته جستجو را پیدا کرده است و می‌خواهد آن را با رشته

جایگزین تعویض کند. این کار با استفاده از متد Replace بر مبنای دامنه‌ای که در متغیر a مشخص شده است انجام می‌شود. مقدار متغیر sflag برابر با 1 تنظیم می‌شود که به معنای نمایش مجدد UserForm است و کد به سراغ برچسب loopa می‌رود که دوباره UserForm را نشان می‌دهد.
اگر کاربر روی دکمه Replace یا Close کلیک نکرده باشد آن‌گاه احتمالاً گزینه‌های دیگری در عملیات جستجوی کاربرگ دخیل هستند. در این مرحله، گزینه‌های دیگر جستجوی UserForm به کار گرفته می‌شوند این‌کار با استفاده از مقادیر ComboBoxها و کادرهای انتخابی انجام می‌شود:

- بر مبنای مقدار متن ComboBox2 که انتخاب مورد جستجو را در خود جای داده است مقدار متغیر li تنظیم می‌شود. این متغیر برحسب مقادیر استفاده شده در متدهای Find و Replace تنظیم می‌شود.
- بر مبنای مقدار متن ComboBox1 که انتخاب انجام جستجو برمبنای ردیف‌ها یا ستون‌ها را در خود جای داده است، متغیر با نام so تنظیم می‌شود.
- مقدار Checkbox1 که حساسیت جستجو نسبت به بزرگ و کوچک بودن حروف را مشخص می‌کند، در متغیری با نام mc قرار می‌گیرد.
- مقدار Checkbox2 که نشان می‌دهد آیا عبارت جستجو باید کل سلول را در برگیرد یا خیر، در متغیری با نام la قرار می‌گیرد. اگر مقدار la، true باشد مقدار متغیر lat برابر 1 تنظیم می‌شود در غیر این صورت برابر 2 تنظیم می‌شود.
- اگر کاربر روی دکمه Replace All کلیک کند، متغیر canc دارای مقدار 4 خواهد بود و مقدار هر چیزی در کاربرگ با استفاده از متغیرهایی که اطلاعات گرفته شده از UserForm را نگه می‌دارند، عوض می‌شود.
- اگر شئ Range که در متغیر a نگه داشته می‌شود هیچ آدرسی نداشته باشد مقدار آخرین سلول صفحه گسترده، در آن قرار می‌گیرد.
- شئ Range در متغیرa، به عنوان گزینه بعدی یافت شده رشته جستجو روی کاربرگ تنظیم می شود و این‌کار با استفاده از متغیرهایی انجام می شود که از UserForm گرفته شده است. متد Find با حرکت از آخرین گزینه یافت شده روی کاربرگ، عمل می‌کند پس نیازی به آپدیت دامنه جستجوی واقعی نیست.

در آخر کار، دیگر موردی از رشته جستجو یافت نمی شود و آدرس شئ Range برابر Null تنظیم می‌شود. وقتی این امر رخ دهد، مقدار متغیر sflag برابر صفر تنظیم می‌شود که به معنای آن است که

دیگر UserForm به نمایش در نیاید در غیر این صورت مقدار این متغیر برابر 1 خواهد بود تا UserForm به نمایش درآید.
حال از متغیری با نام temp استفاده می‌کنیم تا مسیر تمام موارد یافت شده از رشته جستجو را حفظ کند. مسأله‌ای که در رابطه با متد استاندارد Find وجود دارد این است که روی یک کاربرگ اجرا می‌شود، نهایتاً یک آدرس Nul ایجاد می‌کند(به معنای آن‌که دیگر موردی یافت نمی‌شود) و سپس مجدداً کار جستجو را از سلول A1 روی همان کاربرگ از سر می‌گیرد. برای آن‌که متد Find به کاربرگ بعدی برود و جستجو را از آن‌جا آغاز کند، کمی کدنویسی مورد نیاز است تا متد Find بتواند مسیر آن‌چه یافته است را حفظ کند و آن مسیر را در صورت نیاز به کاربرگ جدید انتقال دهداگر آدرس سلول در فرم کارپوشه، کاربرگ و سلول از قبل در متغیر رشته‌ای temp باشد (که به معنای آن است که متد Find به سلول A1 برگشته و جستجو دوباره آغاز شده است) یا آدرس شئ Range در متغیر A برابر Null باشد آن‌گاه مقدار متغیر sflag برابر صفر تنظیم می‌شود(به معنای عدم نمایش UserForm) و اجرا به برچسبی که bypass نامیده می‌شود انتقال یابد.
اگر یک نتیجه صحیح یافت شده باشد آن‌گاه کارپوشه، کاربرگ و سلول، همگی با استفاده از متد Activate فعال می‌شوند. این به معنای آن است که مکان‌نما به محل مورد بعدی متن مورد جستجو انتقال می‌یابد.
نام کارپوشه، نام کاربرگ و آدرس سلول همگی به متغیر temp اضافه می‌شوند. به طوری که بتوانیم کنترل کنیم آیا جستجو روی یک کاربرگ تکمیل شده است یا خیر. سپس اجرا به loopa برمی‌گردد و در آن‌جا UserForm به نمایش درآمده و کاربر می‌تواند مورد بعدی را جستجو کند.
برچسب bypass اجازه می‌دهد تا کد، متغیر temp را پاک کند و به کاربرگ بعدی انتقال یابد. سپس در کاربرگ می‌گردد تا یک مورد از رشته جستجو را پیدا کند؛ در آن قسمت مجدداً UserForm به نمایش درمی‌آید تا کاربر در صورت دلخواه، رشته جستجو راتعریف کند.
وقتی تمام کارپوشه‌ها و کاربرگ‌ها مرور شدند، یک کادر پیغام به نمایش درمی‌آید که حاوی عبارت"No further instances of the search string"(دیگر موردی از رشته مورد نظر یافت نمی‌شود).
کد را اجرا کنید. این، زیرروال findsheet است. صفحه نمایش ما باید شبیه تصویر ۲-۳۷ باشد.



تصویر ۲-۳۷ مثالی از امکانات Search All

عملیات باید دقیقاً مانند متد نرمال Find در منوی صفحه گسترده Excel عمل کند اما خصوصیت اضافی جستجو در صفحه گسترده‌ها و کاربرگ‌ها را داشته باشد.

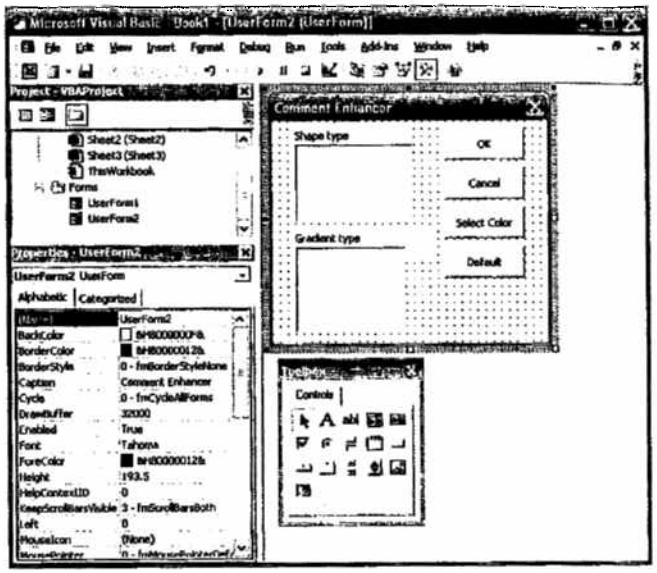
فصل سی و هشتم

ایجاد افکت در توضیحات

حتماً با توضیحات (Comments) در صفحه گسترده Excel آشنا هستید. منظور کادرهای زردرنگی است که به یک سلول اختصاص می‌یابد و وقتی ماوس را روی آن سلول ببریم، باز می‌شود. توضیحات عموماً برای ارایه اطلاعات درباره آن سلول به کار می‌روند. در فصل ۲۲ درباره اضافه کردن فرمول‌ها به توضیحات مطالبی یاد گرفتیم.

هم‌چنین می‌توانیم کدی بنویسیم تا ظاهر کادرهای توضیحات را اصلاح کند– برای مثال، مجموعه بزرگی از اشکال وجود دارد که برای کادرهای توضیحات ما قابل استفاده هستند. این، یک مثال بسیار خوب از قادر بودن به انجام کارهای مختلف در VBA از طریق مدل شئ Excel است که نمی‌تواند از طریق منوی نرمال Excel انجام شود: کل کد در پشت صحنه وجود دارد اما کاربر امکان دستیابی به آن را ندارد.

افکت‌ها روی صفحه گسترده بسیار جذاب به نظر می‌رسند به‌ویژه اگر هرگز از قبل، آن‌ها را ندیده باشیم. کدی که افکت‌ها را اضافه می‌نماید فرض می‌کند کاربر قبلاً گزینشی از سلول‌ها را از محل‌هایی که سلول‌های حاوی توضیحات هستند انجام داده است. پیش از هر چیز لازم است یک فرم رابط کاربر ایجاد کنیم (تصویر ۱-۳۸).



تصویر ۱ – ۳۸

این فرم شامل دو کادر لیست است تا به کاربر اجازه دهد نوع شکل و نوع درجه را انتخاب کند. نوع شکل (shape type) به کاربر اجازه می‌دهد از یک مجموعه اشکال از قبل تعریف شده، یکی را انتخاب کند (مانند مکعب یا بالون). نوع درجه (gradient type) به کاربر اجازه می‌دهد تا درجه‌بندی رنگ و این‌که از مرکز یا گوشه بالایی قرار بگیرد، انتخاب کند.

دو کنترل برچسب برای تعریف کادرهای لیستی وجود دارد. به علاوه، چهار دکمه فرمان وجود دارند که گزینه‌های کاربری آن عبارتند از:

OK : استفاده از تنظیمات توضیحات

Cancel: کاری انجام نمی‌دهد.

Select Color: انتخاب رنگی که کاربر می‌خواهد.

Default: برگرداندن تنظیمات پیش فرض به توضیحات

کنترل Common Dialog نیز اضافه شده است چون ساده‌ترین روش برای آن، این است که به کاربر اجازه انتخاب رنگ‌ها را بدهد. برای دیدن چگونگی استفاده از این کنترل به فصل ۱۰ مراجعه کنید.

نام فرم (caption) با کلیک روی عنوان فرم و ویرایش خصوصیت caption در پنجره Properties قابل تغییر است.

سپس لازم است این کد را به فرم اضافه کنیم تا تمام کنترل‌ها به کار بیفتند:

```
Private Sub CommandButton1_Click( )
UserForm5.Hide
canc = 0
End Sub

Private Sub CommandButton2_Click( )
UserForm5.Hide
canc = 1
End Sub

Private Sub CommandButton3_Click( )
UserForm5.CommonDialog1.CancelError = True
UserForm5.CommonDialog1.Flags = &H1&
On Error GoTo errhandler2
UserForm5.CommonDialog1.Action = 3
col = UserForm5.CommonDialog1.Color
errhandler2:
Exit Sub
End Sub


Private Sub CommandButton4_Click( )
UserForm5.Hide
canc = 2
End Sub

Private Sub UserForm_Initialize( )
ListBox1.AddItem "msoShape32PointStar"
ListBox1.AddItem "msoShape24PointStar"
ListBox1.AddItem "msoShape16PointStar"
ListBox1.AddItem "msoShape8PointStar"
ListBox1.AddItem "msoShapeBalloon"
ListBox1.AddItem "msoShapeCube"
ListBox1.AddItem "msoShapeDiamond"
ListBox2.AddItem "msoGradientDiagonalDown"
ListBox2.AddItem "msoGradientDiagonalUp"
ListBox2.AddItem "msoGradientFromCenter"
ListBox2.AddItem "msoGradientFromCorner"
ListBox2.AddItem "msoGradientFromTitle"
```

```
ListBox2.AddItem "msoGradientHorizontal"
ListBox2.AddItem "msoGradientMixed"
ListBox2.AddItem "msoGradientVertical"
End Sub
```

اگر قبلاً متغیر سراسری با نام canc را معرفی نکرده باشیم باید آن‌را ایجاد کنیم. این متغیر، رکوردی از عملیات‌های کاربر را نگه می‌دارد و به همین‌خاطر می‌تواند به دیگر ماژول‌ها انتقال یابد. هم‌چنین لازم است یک متغیر سراسری با نام col نیز تعریف کنیم تا انتخاب رنگ کاربر را در خود نگه دارد. لازم است این متغیر در بخش Declaration ماژول قرار گیرد:

```
Global canc As Integer
Global canc As Long
```

به‌جز دکمه Color (دکمه ۳) تمام رویدادهای Click مربوط به چهار دکمه فرمان مشابه یکدیگر هستند. وقتی کاربر روی OK، Cancel یا Default کلیک کند،UserForm پنهان شده و مقدار canc طوری تنظیم می‌شود تا نشان دهد کاربر روی کدام دکمه کلیک کرده است(OK برابر صفر،Cancel برابر 1 و Default برابر 2 است).

دکمه Color (دکمه۳) متفاوت است چون باید یک کادر محاوره به کاربر نشان دهد تا به کاربر اجازه دهد یک رنگ انتخاب کند.

کد، خصوصیت CancelError را True تنظیم می‌کند که به معنای آن است که اگر کاربر روی Cancel کلیک کند یک خطا ایجاد می‌شود که کد می‌تواند از آن برای کنترل عملیات Cancel استفاده کند.

خصوصیت Flags برابر H1 تنظیم شده و یک وضعیت خطا رخ می‌دهد. هدف از تنظیم flag، نمایش صحیح کادر محاوره‌ای Color Option است. اگر با کلیک کاربر روی دکمه Cancel، یک خطا ایجاد شود اجرا به errhandler2 می‌پرد و اجرا از زیرروال خارج می شود.

خصوصیت Action برابر 3 تنظیم می‌شود که کادر محاوره‌ای Color Selection را نشان می‌دهد و اجازه می‌دهد کاربر، رنگی را انتخاب کند و سپس آن رنگ در متغیر سراسری col درج می‌شود.

وقتی UserForm بارگذاری می‌شود، کادرهای لیستی باید مقداردهی شوند. این کار با استفاده از متد AddItem از شئ ListBox روی رویداد From Intialize انجام می‌شود. این مثال، نام‌های ثابت واقعی را به کادرهای لیستی اضافه می‌کند اما می‌توانیم آن را اصلاح کنیم تا از نام‌های مختلف یا نام‌های دوستانه‌تر نیز استفاده کند.

نام‌های ثابت که به ListBox1 اضافه شده‌اند، گزینش کوچکی از تمام شکل‌های موجود را نشان می‌دهند. اگر از Object Browser (F2 در پنجره کد نویسی) استفاده کنیم و روی msoAutoShapeType جستجو کنیم، یک دامنه جامع از شکل‌ها را پیدا خواهیم کرد که می‌توانیم آن‌ها را به کار ببریم. متأسفانه، این امکان وجود ندارد که با استفاده از یک حلقه For..Each، نام‌های ثابت را مرور کنیم سپس لازم است تا آن‌ها را خط به خط در کادر لیستی قرار دهیم. این امر به آن خاطر است که آن‌ها شئ (object) نیستند و مقادیر ثابت (contant) هستند، بنابراین قسمتی از یک collection (مجموعه) محسوب نمی‌شوند.

سپس لازم است رویه‌ای برای نمایش UserForm و نشان دادن عکس العمل نسبت به انتخاب‌های کاربر بنویسیم. کد زیر را در یک ماژول قرار دهید:

```
Sub comment_enhance ( )
Dim param As Long, grad As Long
UserForm5.Show
If canc = 1 Then Exit Sub
Select Case UserForm5.ListBox1.Text
Case " msoShape32PointStar"

        param = msoShape32PointStar
Case " msoShape24PointStar"

        param = msoShape24PointStar
Case " msoShape16PointStar"

        param = msoShape16PointStar
Case " msoShape8PointStar"

        param = msoShape8PointStar
Case " msoShapeBalloon"

        param = msoShapeBalloon
Case " msoShapeCube"

        param = msoShapeCube
Case " msoShapeDiamond"

        param = msoShapeDiamond

End Select

Select Case Userform5.ListBox2.Text
```

```
Case  " msoGradientDiagonalDown"

        grad = msoGradientDiagonalDown
Case " msoGradientDiagonalUp"

        grad = msoGradientDiagonalUp
Case " msoGradientFromCenter"

        grad = msoGradientFromCenter
Case " msoGradientFromCorner"

        grad = msoGradientFromCorner
Case " msoGradientFromTitle"

        grad = msoFromTitle
Case " msoGradientHorizontal"

        grad = msoGradientHorizontal
Case " msoGradientMixed"

        grad = msoGradientMixed
Case " msoGradientVertical"

        grad = msoGradientVertical
End Select

If canc = 2 Then
For Each window In Windows

        For Each Worksheet In window.Selectedsheets

                For Each cell In Application.Selection
                        addr = Worksheet.Name &  "!" & cell.Address
                        On Error Resume Next
                        temp = ""
temp = Range(addr).Comment.Text
Range(addr).Comment.Delete
If temp <> "" Then
        Range(addr).AddComment(temp)
End If
                Next cell
        Next worksheet
Next window
Exit Sub
End If
```



```
For Each window In Windows

        For Each Worksheet In window.Selectedsheets

                For Each cell In Application.Selection
                        addr = Worksheet.Name &  "!" & cell.Address
                        On Error Resume Next
Range(addr).Comment.Shape.Fill.OneColorGradient grad, 1, 1

Range(addr).Comment.Shape.Fill.BackColor.RGB = col
Range(addr).Comment.Shape.AutoShapeType = param
                Next cell
        Next worksheet
Next window
Exit Sub
```

در آغاز، دو متغیر راه اندازی شده‌است، یکی به نام param نامیده می‌شود که انتخاب شکل کاربر را نگه می‌دارد و دیگری grad که انتخاب درجه‌بندی رنگ کاربر را در خود نگه می‌دارد. سپس UserForm به نمایش درآمده و کاربر، انتخاب شکل، رنگ و درجه بندی رنگ را انجام می‌دهد.

سپس کد، متغیر canc را کنترل می‌کند تا ببینید آیا مقدار آن 1 است یا نه. مقدار 1 نشان دهنده این است که کاربر در فرم روی دکمه Cancel کلیک کرده است. اگر چنین باشد تنها یک Exit Sub مورد نیاز خواهد بود و به چیزی دیگر احتیاج نخواهد داشت.

اگر کاربر روی دکمه‌ای غیر از Cancel کلیک کند لازم است مشخص کنیم کاربر چه انتخابی انجام داده است. متأسفانه، کادرهای لیستی، فقط رشته‌های متنی نام‌های ثابت‌ها را در خود نگه می‌دارند و انتخاب‌های کاربر باید با استفاده از متد Case (برمبنای ListBox1) به یک مقدار ثابت واقعی تبدیل می‌شود. متغیر param بر مبنای متن انتخاب شده در کادر لیستی، با مقدار ثابت واقعی تنظیم شده است. همین امر برای کادر لیستی دوم هم رخ می‌دهد و مقدار متغیر grad برابر مقدار ثابت انتخاب شده، تنظیم می‌شود.

سپس، کد آزمایش می‌کند که آیا مقدار متغیر canc دارای مقدار 2 است یا خیر. اگر چنین باشد، کاربر روی دکمه Default کلیک کرده است، که تمام توضیحات را به شکل و رنگ پیش فرض برمی‌گرداند. اگر Default انتخاب شده باشد، کد، تمام پنجره‌ها در مجموعه Windows را در یک حلقه مرور می‌کند و سپس تمام کاربرگ‌ها را در شئ sheet انتخاب شده، مرور می‌کند.

برای تنظیم مجدد توضیحات در سلول‌های انتخاب به حالت پیش فرض، لازم است چندین مرحله کدنویسی انجام دهیم چون هیچ دستور خاصی برای انجام این کار در مدل شئ Excel وجود ندارد. باید

از متن مورد توضیح، یک کپی داشته باشیم، توضیح را پاک کنیم، سپس با استفاده از متن کپی شده، آن را دوباره ایجاد نماییم. توضیح جدید، شکل و رنگ پیش فرض را می‌گیرد.

کد، از طریق هر سلول در مجموعه انتخاب شده کار می‌کند و متغیر addr را با نام برگه و آدرس سلول انتخاب شده بارگذاری می‌کند (این دو با کاراکترا به هم الحاق شده‌اند).

دستور On Error Resume Next به کار گرفته شده است چون روشی در مدل شئ Excel وجود ندارد تا آزمایش کنیم آیا به سلول، عبارت توضیحی الحاق شده است یا خیر.

مقدار متغیر temp برابر Null تنظیم شده است، سپس این متغیر با متنی که از عبارت توضیحی برای یک سلول خاص مشخص شده است بارگذاری می‌شود. در هنگام مرور سلول‌هایی مانند این در یک حلقه، بسیار مهم است که متغیر temp را با تنظیم مقدار آن برابر با یک رشته تهی، پاک کنیم در غیر این صورت، این خطر وجود دارد که مطالب و نکته‌ها از طریق دامنه انتخاب شده، جایگزین شوند.

توضیح، با استفاده از متد Delete حذف شده و آزمایش می‌شود تا ببینیم آیا در متغیر temp، متنی وجود دارد یا خیر. یعنی ببینیم آیا توضیحی در آن سلول وجود دارد یا خیر. اگر چنین باشد، توضیح دوباره با استفاده از متد AddComment با تمام تنظیمات پیش فرض اضافه می‌شود و به کاربر اعلان می‌کند توضیح مجدداً تنظیم شده است.

وقتی تمام سلول‌ها برای گزینه‌های پیش فرض، از طریق یک حلقه مرور شدند، از رویه خارج می‌شویم و دیگر عملیاتی رخ نمی‌دهد.

اگر هنوز کد اجرا می‌شود، تنها به این دلیل است که کاربر روی OK کلیک کرده است و این یعنی لازم است تغییراتی روی توضیحات انجام شود.

باز هم کد، پنجره‌های موجود در مجموعه Windows و کاربرگ‌های موجود در شئ windows.selectedsheets را در یک حلقه مرور می‌کند تا تمام برگه‌های انتخاب شده را پیدا کند.

سپس کد، هر سلول را در انتخاب برنامه کاربردی در یک حلقه مرور کرده و متغیر addr را با نام کاربرگ و آدرس سلول بارگذاری می‌کند.

On Error Resume Next دوباره به کار گرفته شده است تا سلول‌های بدون توضیحات را پوشش دهد.

ابتدا، درجه بندی (gardiant) با استفاده از متد Fill.OneColorGrdiant و آن متد به مقدار متغیری که در grad نگه داشته می‌شود، تنظیم شده است. اگر قصد تغییر رنگ توضیحات را داشته باشیم بسیار مهم است تا پیش از هر چیز، درجه بندی را تنظیم کنیم.

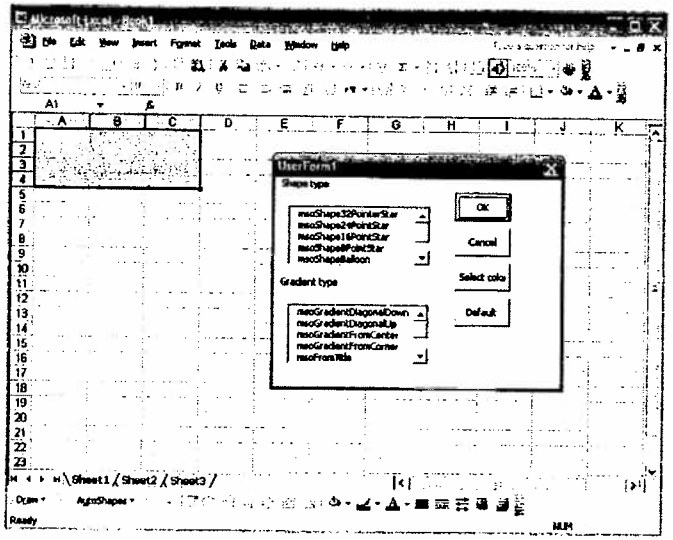
سپس با تنظیم خصوصیت BackColor.RGB برابر با متغیر سراسری col، رنگ تنظیم می‌شود. ما از یک متغیر سراسری استفاده می‌کنیم به طوری که بتوانیم از دیگر ماژول‌ها به مقدار آن دسترسی داشته باشیم.

در آخر، شکل توضیح با تنظیم خصوصیت AutoShapeType برابر با متغیر param تنظیم می‌شود. بسیار مهم است که این سه دستورالعمل آخر را به ترتیب انجام دهیم چون در غیر این صورت ممکن است نتایج نادرست به دست آوریم.

آزمایش کردن نتیجه

با کلیک راست کردن روی سلول و انتخاب Insert Comment از منوی بازشو، توضیحاتی را به سلول‌های صفحه گسترده اضافه کنید. چند سلول که سلول‌های توضیح دار را نیز در خود جای داده‌اند، انتخاب کنید.

کد را اجرا کنید تا صفحه‌ای شبیه تصویر ۲-۳۸ ببینید.



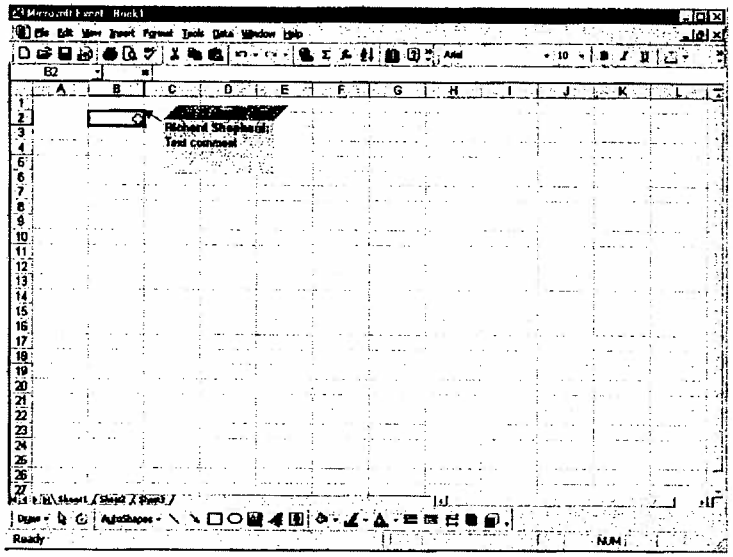
تصویر ۲ – ۳۸

یک شکل و یک نوع درجه بندی انتخاب کنید. روی دکمه Color کلیک کنید و یک رنگ را از کادر محاوره‌ای بازشده انتخاب کنید. روی OK کلیک کرده، سپس نگاهی به توضیحات بیندازید. حال آن‌ها باید شبیه تصویر ۳-۳۸ باشند. البته نتیجه کار وابسته به انتخاب‌های شما نیز هست.

در هنگام استفاده از این رویه، کاربر مجبور نیست هم یک شکل و یک رنگ و یک درجه بندی را انتخاب کند. البته اگر رنگ تغییر کند، درجه بندی نیز باید انتخاب شود در غیر این صورت، تغییر رنگ انجام نمی‌شود.

گاهی اوقات در هنگام تغییر شکل توضیحات، شکل به درستی نمی‌تواند تغییر اندازه دهد تا مقدار متن موجود در خود را نشان دهد و مقداری از متن، نمایش داده نمی‌شود. این امر معمولاً در مورد شکل‌های مدور (مانند ستاره) رخ می‌دهد. با تنظیم خصوصیات Height و Width در شکل‌های توضیحات به شرح زیر می‌توانیم روی این قضیه کار کنیم:

```
Range(addr).Comment.Shape.Height = 100
Range(addr).Comment.Shape.Width = 100
```



تصویر ۳-۳۸

فصل سی و نهم

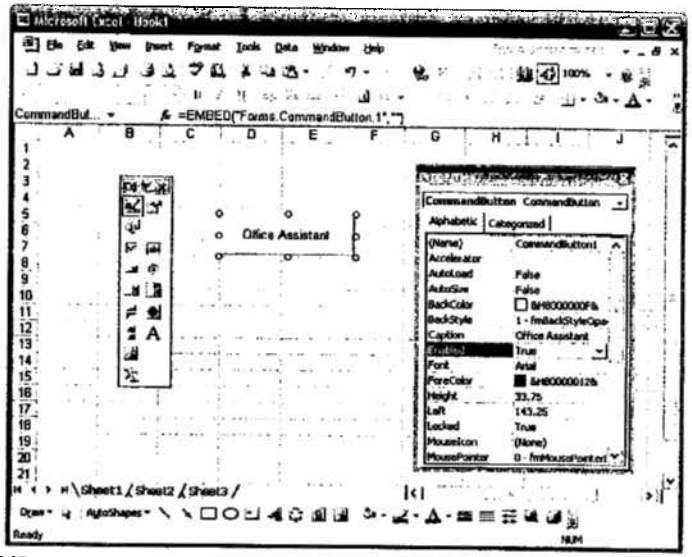
یک راه جایگزین برای کادرهای پیغام

در فصل ۵، کاربرد کادرهای پیغام را برای برقراری ارتباط با کاربر به عنوان یک روش حرفه‌ای برای ایجاد پیغام‌های ساده بررسی کردیم. با این وجود، اگر روش جالب‌تر و سرگرم کننده‌تری برای نمایش اطلاعات و گزینه‌های ساده را بخواهیم، می توانیم Office Assistant متعلق به برنامه Excel را طوری برنامه‌ریزی کنیم تا پیغام‌های حبابی و گرافیکی مختلفی را نمایش دهد و می‌توانیم گزینه‌های مورد نظر را طوری در بالن قرار دهیم تا کاربر بتواند یکی از آنها را انتخاب کند.

برای آنهایی که با Office Assistant آشنا هستند باید بگوییم این دستور، یک کاراکتر متحرک است که در هنگام صدور فرمان Help (فشار دادن F1) در برنامه‌های زیر مجموعه Office ظاهر می‌شود. شکل معمول‌آن، یک گیره کاغذ است که چشم دارد البته کاراکترهای دیگری را هم می‌توانیم انتخاب کنیم. Office Assistant بسیار شبیه کادرهای پیغام ساده عمل می‌کند. متأسفانه مجبوریم یک انیمیشن از Office Assistant را در برنامه خود اجرا کنیم اما گزینه‌های متنوعی هم به عنوان جایگزین وجود دارند، کاراکترهایی که ممکن است تا به حال ندیده باشید.

نکته قابل توجه در این درس آن است که برای عملکرد این مثال باید Office Assistant در قسمت Options فعال شده باشد در غیر این صورت، کد، هیچ چیزی به عنوان خروجی ایجاد نخواهد کرد.

بهترین حالت این است که کد، از طریق یک دکمه فرمان که در صفحه گسترده تعبیه شده است اجرا شود. برای قرار دادن دکمه فرمان روی یک صفحه گسترده، از منوی صفحه گسترده، View | Toolbars | Control Toolbox را انتخاب کنید. این باعث می‌شود جعبه ابزار آیکن‌های کنترل‌ها فعال شود. آیکن Command Button را پیدا کنید و آن را روی صفحه گسترده بکشید. اگر از Office Xp استفاده می‌کنید، ممکن است به ترسیم این آیکن روی صفحه گسترده نیاز داشته باشید. روی دکمه راست کلیک کنید و از منوی بازشو، Properties را انتخاب کنید. خصوصیت Caption را به "OfficeAssistant" تغییر دهید (تصویر ۱-۳۹).



تصویر ۱-۳۹ قرار دادن یک دکمه فرمان بر روی صفحه گسترده برای فراخوانی Office Assistant

روی دکمه جدید دابل کلیک کنید تا وارد پنجره کد نویسی رویداد Click شوید. کد زیر را در رویداد وارد کنید:

```
Private Sub CommandButton1_Click ( )
Dim ball As Balloon
Set ball = Application.Assistant.NewBalloon
With ball
        .Heading = "A demonstration of Office Assistant"

        .Text = "Select an option or press a button"

        .Icon = msoIconTip

        .Button = msoButtonSetAbortRetryIgnore

        .Labels(1).Text = "Option1"
                        .Labels(2).Text = "Option2"

        .Animation = msoAnimationGetArtsy
```



```
        .BalloonType = msoBalloonTypeButtons
End With
Result = ball.Show
If result = 1 Then MsgBox "You pressed Option1"
If result = 2 Then MsgBox "You pressed Option2"
If result = msoBalloonButtonAbort Then MsgBox "You pressed Abort"
If result = msoBalloonButtonRetry Then MsgBox "You pressed Retry"
If result = msoBalloonButtonIgnore Then MsgBox "You pressed Ignore"

End Sub
```

متغیری با نام ball به عنوان یک بالن (حباب گفتگو که از Office Assistant به‌وجود می‌آید)، تعریف شده است. سپس متد NewBalloon، این متغیر را به یک بالن جدید تنظیم می‌کند چون باید یک بالن جدید در بالای بالن فعلی برای Office Assistant تنظیم شود.

دستور With به ما اجازه می‌دهد تا بالن را به صورت اختصاصی و دلخواه در آوریم. خصوصیت Heading، متن پررنگی است که بالای بالن ظاهر می‌شود. خصوصیت Text، بدنه کلی متن بالن را نگه می‌دارد. خصوصیت Icon اجازه انتخاب دو آیکن یا هیچ‌یک از آن‌ها را می‌دهد. خصوصیت Batton نشان‌دهنده دکمه‌های پایین بالن است. این خصوصیت چند مقدار ثابت دارد که بستگی به این دارد که شما می خواهید چه دکمه‌هایی به نمایش درآیند.

می‌توانیم با استفاده از مجموعه Lables، دکمه‌های اختصاصی خود را ایجاد کنیم. شاخص، خود برچسب و ترتیب نمایش آن ارجاع می‌کند اما این نیز، مقدار برگشتی در صورت کلیک کاربر روی آن است. در این مورد، Lables(1) برابر Option1 است و مقدار برگشتی برابر 1 است. Lables(2) نیز Option2 است و مقدار برگشتی آن برابر 2 است.

خصوصیت Animation نشان می‌دهد ظاهر Office Assistant چگونه است. در این مثال، گزینه آن، msoAnimationGetArtsy است که یک نمای مدرن به Office Assistant می‌دهد.

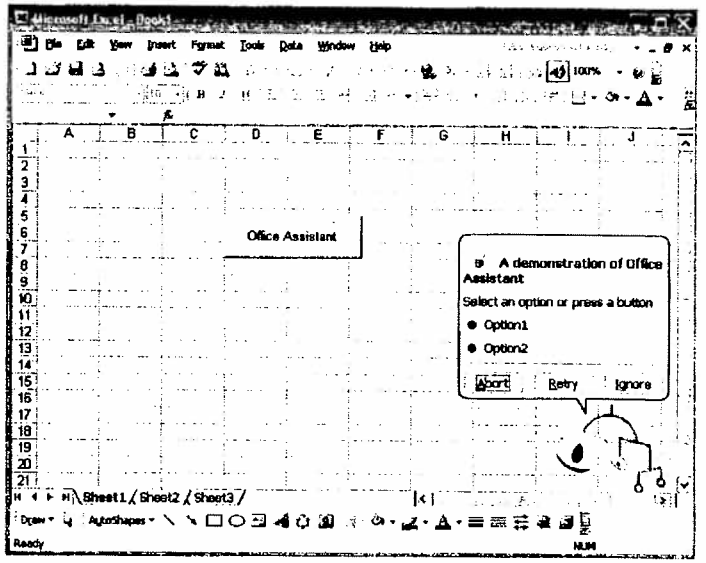
خصوصیت BalloonType برابر msoBalloontypeButtons تنظیم شده است.

سپس Balloon و انیمیشن Office Assistant با استفاده از متد Show به نمایش در می‌آیند. متغیر Result، مقادیر دکمه‌های کلیک شده را برمی‌گرداند.

سپس دستور With پایان یافته و یک سری دستور If به کار گرفته شده است تا به مقدار موجود در متغیر result جواب دهد و یک کادر پیغام با عبارت مناسب در آن به نمایش درآید.

وقتی کد را وارد کردید به صفحه گسترده برگردید. در پنجره Toolbox روی آیکن سمت چپ- بالا کلیک کنید (که متن توضیحی آن، "Exit Design Mode" است). پنجره جعبه ابزار و پنجره

Properties را ببندید. روی دکمه Office Assistant کلیک کنید تا نتیجه‌ای مانند تصویر ۲-۳۹ ظاهر شود.



تصویر ۲-۳۹ شرح برنامه‌نویسی برای Office Assistant

فصل چهلم

کار با شکل‌ها

حتماً می‌دانید که می‌توانیم شکل‌هایی را در صفحه گسترده درج کنیم. در کل،اشیایی مانند نمودارها و توضیحات، شکل هستند. ممکن است قبلاً از منوی AutoShapes روی نوار ابزار Drawing برای درج یک شکل در صفحه گسترده و حتی کار با اشیا سه بعدی استفاده کرده باشید اما آن‌چه نمی‌دانید این است که VBA، مجموعه اشکال عظیمی را در اختیار ما می‌گذارد که از منوهای Excel قابل دسترسی نیستند.

اشکال را می‌توان از طریق VBA دستکاری کرد تا دامنه کاربرد آن‌ها گسترده‌تر شود. رابط کاربر در Excel برای کسب این نتایج و تأثیرات جواب می‌دهد اما در VBA، این رابط راحت‌تر است و با سادگی بیشتری می‌توانید روی آن کار کنید. با به کارگیری شکل‌های نامتعارف و جدید و تأثیرات رنگ‌بندی و افکت‌های سه بعدی می‌توانیم کاری کنیم که صفحه گسترده ما، جالب‌تر به نظر آید.

کد زیر را در یک ماژول قرار دهید:

```
Sub test_shad()
Dim w As Worksheet
Set w = worksheets("sheet1")
Dim myshape As Shape
Set myshape = w.Shapes.AddShape(msoShapeBalloon, 90, 90, 90, 40)
myshape.TextFrame.Characters.Text = "Test Message"
myshape.TextFrame.Characters.Font.Bold = True
With myshape.ThreeD

        .Visible = True

        .Depth = 40

        .ExtrusionColor.RGB = RGB(255, 100, 255)

        .Perspective = False
        .PresetLightingDirection = msoLightingTop

End With

Myshape.BottomRightCell = ""
```

End Sub

در این‌جا، متغیری با نام w به عنوان یک کاربرگ معرفی شده، سپس در مجموعه Worksheets برابر با sheet1 قرار گرفته است. متغیری با نام Myshape هم به عنوان یک شکل تعریف شده است و با استفاده از دستور Set، شکل جدیدی را که به کاربرگ اضافه شده است، در آن قرار داده‌ایم. شکل اضافه شده، یک بالن (کادر گفتگو) است و پارامترهایی برای خصوصیات Width, Top Height, و Left تنظیم شده‌اند. با استفاده از شئ TextFrame، متن این شکل را برابر"Test Message" تنظیم کرده‌ایم و قلم نیز به صورت ضخیم (bold) تنظیم شده است.

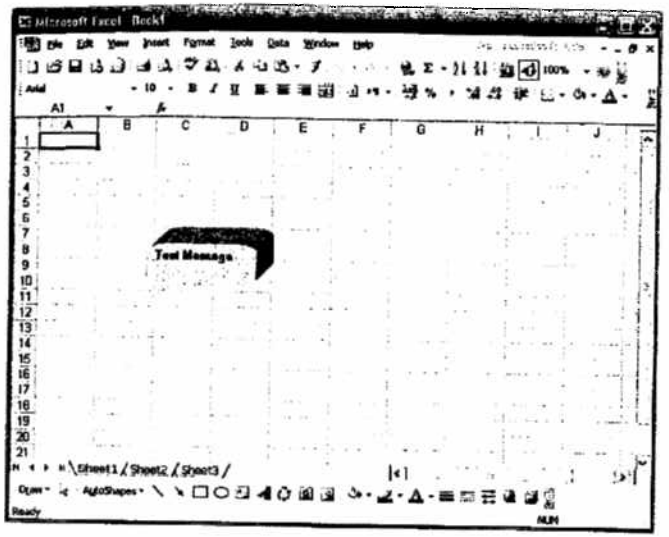
با استفاده از دستور With امکان تنظیمات سه بعدی روی شئ میسر می‌شود. خصوصیات Visible را True قرار می‌دهیم تا بتوانیم قسمت سه بعدی شئ را ببینیم. عمق افکت سه بعدی را ۴۰ و رنگ قسمت برجسته را برابر با مقدار RGB رنگ بنفش تنظیم می‌کنیم. قسمت برجسته، افکت سه بعدی در پشت شکل واقعی است.

خصوصیت Perspective را False تنظیم می کنیم. این خصوصیت نشان می‌دهد آیا افکت سه بعدی به یک نقطه کم رنگ شونده مشترک می‌رسد یا لبه‌های قسمت برجسته، راست و مستقیم خواهد بود.

جهت روشن شکل را از سمت بالا تنظیم کرده‌ایم البته می‌توان آن را طوری تنظیم کرد که با استفاده از مقادیر ثابت موجود، نور از دیگر لبه‌های شکل نیز تابیده شود. با استفاده از Object Browser (زدن دکمه F2 هنگامی که در VBA هستیم) و جستجوی msoPresetLightDirection می‌توانیم این مقادیر ثابت را ببینیم.

در آخر، سلول پایین سمت راست شکل، دارای یک مقدار تهی تنظیم می‌شود. این امر ممکن است غیر ضروری به نظر آید اما به ما اطمینان می‌دهد که شکل واقعاً روی کاربرگ ظاهر خواهد شد. اگر این‌کار را انجام نداده و کد را اجرا کنیم و تا زمانی که با ماوس، کاربرگ را بالا و پایین نبریم، شکل را نخواهیم دید.

اگر کد را اجرا کنیم، نتیجه کار شبیه تصویر ۱-۴۰ خواهد بود.



تصویر ۱-۴۰ مثالی از یک شکل تعریف شده VBA در Excel

شکل‌ها را می‌توان برای ماکروهای فعال هم به کار برد. کد زیر را به ماژولی که خود درج کرده‌اید اضافه کنید:

```
Sub text_message()
        MsgBox "My shape message"
End Sub
```

این کد، یک کادر پیغام ساده را نمایش می‌دهد. اکنون خط زیر را به ماکروی Shape اضافه کنید:

```
myshape.OnAction = "text_message"
```

این خط، خصوصیت OnAction را طوری تنظیم می‌کند تا زیرروالی را که با نام text_message ایجاد شده است، فرا بخواند. وقتی روی شکل کلیک کنیم، کد ما اجرا خواهد شد.

با کلیک روی قسمت برجسته شکل و فشار دادن دکمه Delete، شکل را پاک کرده و دوباره ماکرو را اجرا کنید. وقتی مکان‌نما را روی شکل جدید ببریم دارای یک آیکن دست خواهد بود. روی شکل کلیک کنید تا کادر پیغام ظاهر شود.

فصل چهل و یکم

تبدیل کد VBA به یک برنامه الحاقی (Add-In)

در تمام این کتاب، نمونه‌ها و مثال‌ها را به صورت ماکروهایی ساختیم که همگی از طریق پنجره کدنویسی یا از طریق یک دکمه فرمان در صفحه گسترده اجرا می‌شدند. اگر این ماکروها را برای استفاده شخصی یا کاربران قوی بنویسیم، این کار ایرادی ندارد، چون خود دقیقاً می‌دانیم چه می‌خواهیم و چگونه از آن‌ها استفاده کنیم.

اما اگر مخاطبان ماکرو زیاد باشند تکلیف چیست؟ دیگر کاربران ممکن است اصلاً درباره عملکرد ماژول‌ها چیزی ندانند و حتی دوست نداشته باشند کد شما را ببینند.

جواب این است که باید یک برنامه الحاقی ایجاد کنیم. احتمالاً با برنامه‌های الحاقی شخص ثالث (Third – Party) در Excel آشنا هستید. اگر از منوی صفحه گسترده، Tools | Add-In را انتخاب کنیم تعدادی از این برنامه‌های الحاقی را می‌توان دید. خوشبختانه تبدیل کد به یک فایل الحاقی کار نسبتاً ساده‌ای است.

اولین چیزی که لازم داریم، یک رابط کاربر صحیح برای فراخوانی تمام رویه‌ها است. بهترین روش انجام این کار، اضافه کردن یک منوی شخصی به ساختار منوی Excel است. این امر در فصل ۱۱ تشریح شد.

قصد داریم یک برنامه الحاقی با نام Magic ایجاد کنیم.

فرض کنید تمام کد مثال‌های قبلی را وارد کرده و UserFormهای مقتضی را هم درج کرده‌ایم. اکنون باید کد زیر را برای ساختار منو اضافه کنیم. اگر کد را در فایل‌های مختلف وارد کرده‌اید می‌توانید برای جمع کردن همه آن‌ها در یک ماژول، از دستور Copy و Paste استفاده کنید. اگر تمام مثال‌ها را اجرا نکرده‌اید، می‌توانید برای یک مثال خاص، دستور نوار منو را اضافه نکنید. ساده‌ترین روش انجام این کار، Rem کردن خطوطی است که آن‌ها را لازم ندارید. این کار با اضافه کردن کاراکتر (') به جلوی خط مذکور انجام می‌شود. سپس این خط کد، به رنگ سبز در آمده و به جای انجام عملیات به عنوان یک توضیح در نظر گرفته می‌شود. برای مثال اگر می‌خواهید گزینه Calculate Range را حذف کنید باید دستورات را Rem کنید:

```
.Controls.Add(Type:=msoControlButton).Caption = "Calculate Range"
        .Controls("Calculate Range").OnAction = "range_calculate"
```

این کد باعث حذف خطی از کد می‌شود که گزینه مربوط را روی منو ایجاد می‌کند.

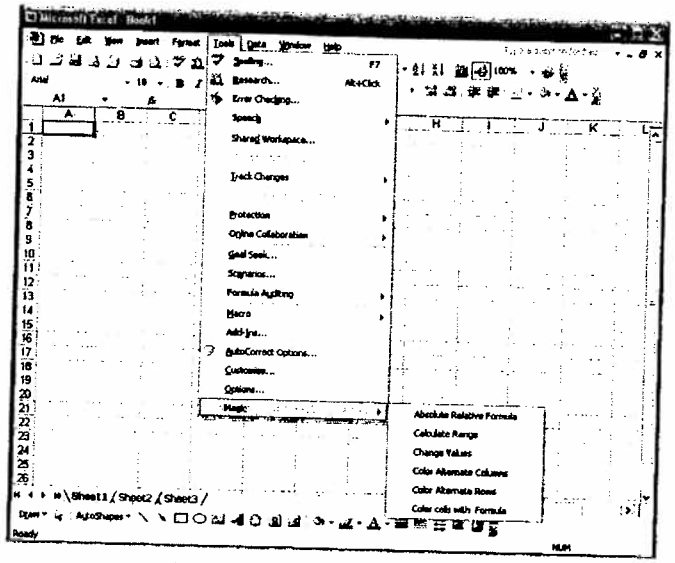
```
Sub menu ( )

Dim newsubitem As Object

commandBars("Worksheet menu bar") _
.Controls("Tools").Controls.Add(Type:= msoControlPopup).Caption = _
"Magic"

Set newsubitem = CommandBars("worksheet menu bar") _
.Controls("tools").Controls("magic")

With newsubitem

        .Controls.Add(Type:=msoControlButton). Caption = _
        "Absolute Relative Formula"

        .Controls("Absolute Relative Formula").OnAction = _
  "conv_formula"

        .Controls.Add(Type:=msoControlButton).Caption = _
"Calculate Range"

        .Controls("Calculate Range").OnAction = _
"range_calculate"

        .Controls.Add(Type:=msoControlButton).Caption = _
"Change Values"
        .Controls("Change Values").OnAction = "change_val"

        .Controls.Add(Type:=msoControlButton).Caption = _
        "Color Alternate Columns"

        .Controls("Color Alternate Columns").OnAction = _
          "shadel"

              .Controls.Add(Type: =msoControlButton).Caption = _
          "Color Alternate Rows"

              .Controls("Color Alternate Rows").OnAction = _
          "shade"
```



```
        .Controls.Add(Type:=msoControlButton).Caption = _
        "Color cells with  Formula"

              .Controls("Color Cells with Formula").OnAction = _
        "col_cell"
              .Controls.Add(Type:=msoControlButton).Caption = _
        "Contents to Label"

              .Controls("Contents to Label").OnAction = _
        "contents_to_label"

              .Controls.Add(Type:=msoControlButton).Caption =_
        "Copy Hidden Sheets"

      .Controls("Copy hidden Sheets").OnAction = _
  "hidden_sheets"

        .Controls.Add(Type:=msoControlButton).Caption = _
        "Enter Formula as Notes"

      .Controls("Enter Formula as Notes").OnAction = _
  "note"

      .Controls.Add(Type:=msoControlButton).Caption = _
"Label to Number"

      .Controls("Label to Number").OnAction = _
"label_to_number"

      .Controls.Add(Type:=msoControlButton).Caption = _
"Matrix Total"

      .Controls("Matrix Total").OnAction = _
"matrix_total"

      .Controls.Add(Type:=msoControlButton).Caption = _
"Reverse Label"

      .Controls("Reverse Label").OnAction = _
"reverse_label"

.Controls.Add(Type: =msoControlButton).Caption= _
  "Search"

.Controls("Sort Sheets").OnAction = "findsheet"
```

```
.Controls.Add(Type:=msoControlButton).Caption= _
   "Sort Sheets"

.Controls("Sort Sheets").OnAction ="Sortsheet"

.Controls.Add(Type:=msoControlButton).Caption= _
   "Transpose"

   .Controls("Transpose"). OnAction="transpose"

End With

End Sub
```

در آغاز متغیری با نام newsubitem به عنوان یک object تعریف می‌شود. سپس یک عنصر منوی جدید به نوار منوی Tools اضافه می‌شود. البته توجه کنید که به صورت یک عنصر استاندارد اضافه نمی‌شود بلکه به صورت بازشو (type=msoControlPopup) اضافه می‌شود و این به معنای آن است که در زیر آن می‌توانیم یک زیر منو داشته باشیم. این زیر منو تمام روال‌ها را در خود نگه می‌دارد.
متغیر newsubitem برابر با این عنصر منو قرار می‌گیرد. با استفاده از دستور With، عناصر مستقل منو به زیر منو اضافه می‌شوند. هر کدام از آن‌ها به صورت یک دکمه کنترلی (type=msoControlButton) اضافه می‌شوند و خصوصیت OnAction برابر با نام زیرروال تنظیم می‌شود.
کد قبلی را در یک ماژول وارد کرده و آن را اجرا کنید. گزینه‌ای با نام Magic در منوی Tools ایجاد خواهد شد. اگر آن را انتخاب کنیم یک زیرروال خواهیم دید (تصویر ۱-۴۱).



تصویر ۱-۴۱ مثال ساخت زیر منوی باز شو

وقتی این منو ایجاد شد، به کدی برای حذف آن نیز احتیاج خواهیم داشت. کاربر برنامه الحاقی ما ممکن است نخواهد این منو را روی برنامه Excel خود داشته باشد، پس باید راهی برای حذف آن داشته باشد. این امر به سادگی با استفاده از دستور Delete، انجام می‌شود.

```
Sub remove_menu ( )
        CommandBars("Worksheet menu bar") .Controls("Tools"). _
Controls("magic").Delete
End Sub
```

از نقطه نظر کاربر، این روال‌ها چگونه عمل می‌کنند؟ وقتی برنامه الحاقی نصب شد، لازم است ساختار منو ظاهر شود و وقتی برنامه الحاقی حذف شد باید ساختار منو نیز حذف شود. خوشبختانه رویدادهای داخلی برای کنترل این امر وجود دارند. ساختار درختی Project، باعث باز شدن Microsoft Excel Objects می‌شود. روی ThisWorkbook دابل کلیک کنید. این شیئی است که برای کارپوشه در برنامه الحاقی ما در نظر گرفته شده است. پیش فرض آن، رویداد Workbook_Open است اما برای یک برنامه

الحاقی، می‌خواهیم وقتی برنامه الحاقی نصب شد همه چیز رخ دهد نه این‌که وقتی کارپوشه باز می‌شود.

از لیست باز شو در گوشه سمت راست- بالا از پنجره ماژول استفاده کرده و آن‌را از بالا به پایین مرور کنید تا زمانی که AddinInstall و AddinUninstall را ببینید. روی AddinInstall کلیک کرده و کد زیر را وارد کنید:

```
Private Sub Workbook_AddinInstall ( )
        Call menu
End Sub
```

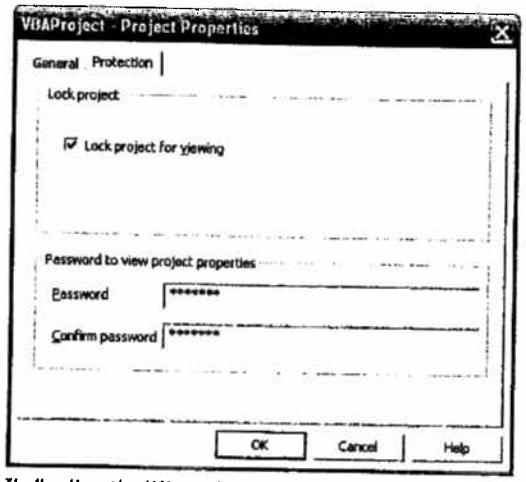
این روال، زیرروال menu را که قبلاً کد نویسی شده است، فرا می‌خواند. هر جا برنامه الحاقی نصب شود، زیر روال menu فرا خوانده شده و ساختار منوی جدید اضافه می‌شود. یک خصوصیت ایمن در این‌جا این است که برنامه الحاقی تنها یک بار می‌تواند نصب شود. این به معنای آن است که منو تنها یک بار ظاهر می‌شود.

در منوی بازشو در گوشه سمت راست- بالا از پنجره ماژول روی AddinUninstall کلیک کرده و کد زیر را اضافه کنید:

```
Private Sub Workbook_AddinUninstall ( )
        Call remove_menu
End Sub
```

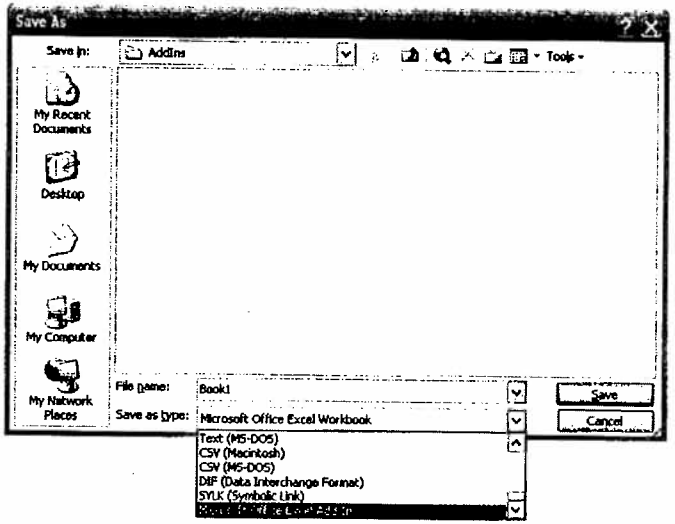
این کد، زیر روال remove_menu را فرا می خواند که باعث حذف منوی Magic می شود. این امر وقتی رخ می دهد که کاربر برنامه الحاقی را پاک کند و با این کد، هیچ اثری از آن باقی نمی‌ماند.

اکنون تقریباً یک برنامه الحاقی را کامل کرده‌ایم. مرحله بعدی، نام‌گذاری آن برنامه است به طوری‌که وقتی کاربر آن را نصب می‌کند بتواند به راحتی آن را شناسایی کند. برای انجام این کار | Tools VBAProject Properties را انتخاب کنید تا فرمی را که VBAProject Properties نامیده می‌شود، ببینید. نام پروژه را وارد کنید. دقت کنید که حتماً یک نام مناسب وارد کنید.



تصویر ۲-۴۱ وارد کردن کلمه عبور برای محافظت از برنامه الحاقی

ممکن است نخواهیم کاربران کد ما را ببینند یا آن را تغییر دهند. برای جلوگیری از این قضیه باید روی زبانه Protection کلیک کنیم (تصویر ۲-۴۱). کادر Lock Project for viewing را کنترل کرده و دو بار کلمه عبور دلخواه خود را وارد کنید. روی OK کلیک کنید. هرگز تحت هیچ شرایطی کلمه عبور را فراموش نکنید، چون راهی برای برگرداندن فایل وجود ندارد. تمام آن‌چه باقی می‌ماند این است که پروژه را به یک فایل الحاقی تبدیل کنیم. روی صفحه گسترده کلیک کرده و File | Save As را از منوی صفحه گسترده انتخاب کنید. در منوی Save As Type، گزینه‌ای برای ذخیره کردن برنامه به صورت Microsoft Excel Add-In وجود دارد (تصویر ۳-۴۱). فیلتر Add-In را انتخاب کرده و روی OK کلیک کنید. حال فایل به جای پسوند XLS با پسوند XLA ذخیره می‌شود.



۳-۴۱ ذخیره کردن فایل به صورت Add-in

از برنامه Excel خارج شده و دوباره آن را اجرا کنید تا برنامه الحاقی را آزمایش کنید. از Tools | Add-In روی منوی صفحه گسترده استفاده کنید تا برنامه الحاقی نصب شود. در حالت معمولی باید آن را در فهرست ببینیم و اگر چنین نبود از دکمه Browse برای پیدا کردن آن استفاده کنیم. تصویر ۴-۴۱ نشان دهنده پنجره Add-Ins و دکمه Browse است.

روی OK کلیک کنید تا فایل الحاقی نصب شود. از منوی صفحه گسترده Tools | Magic را انتخاب کنید تا گزینه‌های جدید منو را مانند تصویر ۱-۴۱ را ببینید.

تصویر ۴-۴۱ نصب برنامه الحاقی (Add-In)

توجه کنید که وقتی از منوی صفحه گسترده، منوی Window را انتخاب می‌کنیم هیچ نشان و علامتی از حضور برنامه الحاقی در آن‌جا نیست. این به دلیل آن است که کارپوشه متعلق به برنامه کاری، تنها به صورت مجازی وجود دارد. آن را می‌توان توسط کد VBA استفاده کرد اما نمی‌توان آن را روی صفحه نمایش ظاهر کرد.

کلیدهای ALT-F11 را فشار دهید و نگاهی به صفحه کد VBA بیندازید. شیئی در ساختار درختی Project با نام Magic.xla خواهید دید که نشان دهنده برنامه الحاقی است. اگر از کلمه عبور برای محافظت آن استفاده کرده باشیم در هنگام باز کردن آن، باید کلمه عبور را وارد کنیم.

هنوز هم می‌توانیم کد VBA را ویرایش کنیم، تغییراتی در آن ایجاد کنیم، گزینه‌های دیگر منو به آن اضافه کنیم و فرم‌های جدید درج کنیم. با وجود این اکنون که آن کد تبدیل به یک برنامه الحاقی شده است، تنها روش ذخیره کردن تغییرات، استفاده از File | Save Magic.xla از منوی کد است.

برنامه الحاقی را پاک کنید. از Tools | Add-Ins در منوی صفحه گسترده استفاده کرده و برنامه الحاقی را در فهرست پیدا کنید. تیک کنار کادر انتخابی را برداشته و روی OK کلیک کنید. اکنون Tools را از منوی صفحه گسترده انتخاب کنید و خواهید دید که گزینه منوی Magic ناپدید شده است.

لیست کتابهای انتشارات دیباگران تهران

نام کتاب	مؤلف / مترجم
۱- مهندسی نرم‌افزار با بهره‌گیری از UML	مؤلف: مهندس امیر مهدی هدایت‌فر
۲- آموزش گام به‌گام Word 2003	مترجم: مسعود پاک‌نظر
۳- خودآموز PHP در 24 ساعت	مترجمان: مهندس علی ناصح، محمد ناصح
۴- ابزارهای رسانه‌ای دیجیتال	مترجم: مهندس رامین مولاناپور
۵- درسنامه سازمان و مدیریت تخصصی بیمارستان(۱)	مؤلف: علی محمد مصدق راد
۶- درسنامه سازمان و مدیریت تخصصی بیمارستان(۲)	مؤلف: علی محمد مصدق راد
۷- آموزش گام به‌گام AutoCAD 2004 مقدماتی	مترجم: مهندس مرتضی متواضع
۸- آموزش گام به‌گام AutoCAD 2004 پیشرفته	مترجم: مهندس مرتضی متواضع
۹- آموزش VHDL	مترجم: مهندس فرزان گیتی
۱۰- آموزش گام به‌گام Office System 2003	مترجم: مهندس علی اکبر متواضع
۱۱- مرجع کامل برنامه‌نویسی Visual Basic.NET	مترجم: مهندس رامین مولاناپور
۱۲- حسابداری مالی با استفاده از نرم‌افزار پاپاپای	مؤلف: مهندس مهرداد اسماعیلی
۱۳- دوربین‌ها و عیب‌یابی تصاویر دیجیتالی	مترجم: منیره آبخو
۱۴- مبانی و فناوری کامپیوتر	مؤلف: مهندس مجتبی الله‌وردی
۱۵- واژه‌پرداز Word 2002	مؤلفان: مهندس مرتضی متواضع، افسون آذین
۱۶- پست الکترونیکی، اینترنت و ویروس‌های کامپیوتری	مؤلفان: مهندس مرتضی متواضع، مهندس مجتبی ا... وردی
۱۷- استانداردسازی فناوری	مؤلف: ابراهیم طلایی
۱۸- سیستم عامل مقدماتی DOS, Windows XP	مؤلفان: مهندس مجتبی ا... وردی، افسون آذین
۱۹- صفحه گسترده Excel 2002	مؤلف: مهندس علی‌اکبر متواضع
۲۰- زمین‌شناسی	مؤلف: امیر میرآخوری
۲۱- CIW :Java Script Fundamental	افست
۲۲- CIW: Design Methodology and Technology Volume I	افست
۲۳- CIW: Design Methodology and Technology Volume II	افست
۲۴- آموزش حرفه‌ای Macromedia Flash MX برای ویندوز میکنتاش	مترجم: نیما عربشاهی
۲۵- ارایه مطالب PowerPoint 2002	مؤلفان: مهندس مرتضی متواضع، افسون آذین
۲۶- عیب‌یابی صفحات وب	مترجمان: مهندس سید امیرحسین رضوی، مهندس ملیحه دهقان
۲۷- عیب‌یابی در Excel 2003	مترجم: مهندس مرتضی متواضع
۲۸- فناوری اطلاعات و ارتباطات ICT (پایه اول راهنمایی)	مؤلف: مهندس مرتضی متواضع
۲۹- فناوری اطلاعات و ارتباطات ICT (پایه دوم راهنمایی)	مؤلف: مهندس مرتضی متواضع
۳۰- راهنمای آموزشی Macromedia FreeHand MX	مترجم: مهندس رامین مولاناپور
۳۱- تدوین و ویرایش صوت با CoolEdit pro 2	مترجم: مهندس رامین مولاناپور

نام کتاب	مؤلف / مترجم
۳۲- CIW: Java programming Fundamentals	افست
۳۳- CIW: E-Commerce strategies and practices volume I	افست
۳۴- CIW: XML Document Design student Guide	افست
۳۵- اینترنت 2004	مؤلف: مهندس رامین مولاناپور
۳۶- Adobe Photoshop برای عکاسان دیجیتال	مترجم: مهندس رامین مولاناپور
۳۷- CIW: Building Data Base Client Application	افست
۳۸- CIW: E-Commerce strategies and Practices volume II	افست
۳۹- CIW: Java server pages	افست
۴۰- CIW: Java servlets	افست
۴۱- گواهینامه بین‌المللی کاربری کامپیوتر و اینترنت IC3 (جلد اول)	مترجم: مهندس علی‌اکبر متواضع
۴۲- گواهینامه بین‌المللی کاربری کامپیوتر و اینترنت IC3 (جلددوم)	مترجم: مهندس علی‌اکبر متواضع
۴۳- عیب‌یابی در Microsoft Project 2002	مترجم: حسین یعسوبی
۴۴- خودآموز تصویری Picture IT 7	مترجمان: مهندس مرتضی متواضع، هما تیموری
۴۵- مهندسی مکانیک (حرارت و سیالات)	مؤلفان: علیرضا دهقانی، حسین نعمتی، محسن مزیدی
۴۶- خودآموز برنامه‌نویسی Delphi 8 برای محیط Microsoft.Net (جلد اول)	مترجم: مهندس مهرداد اسماعیلی
۴۷- خودآموز برنامه‌نویسی Delphi 8 برای محیط Microsoft.Net (جلد دوم)	مترجم: مهندس مهرداد اسماعیلی
۴۸- ۱۰۱ راهکار برای ارتقای سایت‌های وب	مترجمان: مهندس امیرحسین رضوی، مهندس ملیحه دهقان
۴۹- راهنمای آموزشی Borland C#Builder	مترجم: مهندس رامین مولاناپور
۵۰- از مبانی برق تا مدارهای الکتریکی	مترجم: دکتر مسعود دوستی
۵۱- مجموعه سؤالات کاربردی Windows XP	مؤلفان: مهندس زهرا رضایی، مهندس غلامرضا محمدی‌فر
۵۲- حسابداری صنعتی مقدماتی	مؤلفان: مستانه رضایی - شهناز نوبخت
۵۳- ماشین آلات ساختمانی و راه‌سازی	مؤلف: مهندس حسین اکبرزادگان
۵۴- راهنمای آموزشی Visio 2003	مترجم: مهندس رامین مولاناپور
۵۵- راهنمای کاربردی Adobe After Effect 6.5	مترجم: مهندس رامین مولاناپور
۵۶- Macromedia Dreamweaver MX 2004 و بانکهای اطلاعاتی	مترجمان: مهندس رامین مولاناپور، لیلا ملکان
۵۷- راهنمای پرسپکتیو برای هنرجویان هنرهای تجسمی	مترجم: فیروزه شیبانی رضوانی
۵۸- مسئول حقوق و دستمزد	مؤلفان: مستانه رضایی، شهناز نوبخت، مجتبی ا... وردی
۵۹- ابزارهای مقابله با مهاجمین و تأمین امنیت شبکه‌ها	مترجمان: مهندس علی ناصح، محمد ناصح
۶۰- آموزش رایانه‌کار درجه ۲ (جلد اول)	مؤلفان: حسین رحیمی، شهرام شکوفیان، قائم حیدری مقدم

نام کتاب	مؤلف / مترجم
۶۱- ریاضی عمومی ۱ (تکمیلی)	مؤلفان: مهندس کامبیز یوسفی، نساء یوسفی
۶۲- آموزش کاربردی AutoCAD 2004 ویژه کاربران صنعتی(پیشرفته)	مؤلف: فرهاد ضرابی
۶۳- نرم‌افزار NU	مؤلف: مهندس مجتبی الله وردی
۶۴- آزمونهای کارشناسی ناپیوسته مهندسی تولیدات دامی	مؤلف: مهندس زینب جعفری
۶۵- خودآموز آسان Paintshop pro 8	مترجم: مهندس رامین مولاناپور
۶۶- مبانی مهندسی نرم‌افزار	مؤلف: امیرمهدی هدایت‌فر
۶۷- امکانات جدید Macromedia Flash MX 2004	مؤلف: بهنام درفشی
۶۸- آموزش گام به گام طراحی وب	مترجم: مهندس رامین مولاناپور
۶۹- ICDL در طراحی به کمک کامپیوتر (CAD)	مؤلف: مهندس مرتضی متواضع
۷۰- ICDL پیشرفته (پایگاه داده) مهارت ۵	مترجم: مهندس علی‌اکبر متواضع
۷۱- مرجع کاربردی Primavera 3.1	مترجم: مهندس علی رضایی
۷۲- زبان تخصصی علوم دامی	مؤلف: مهندس شهرام نصابیان
۷۳- آزمونهای کاردانی پیوسته کامپیوتر	مؤلف: مهندس محمد عادلی‌نیا
۷۴- راهنمای کاربردی Director MX 2004	مترجم: مهندس رامین مولاناپور
۷۵- ساخت سایتهای پویا با استفاده از Macromedia Studio MX 2004	مترجم: مهندس رامین مولاناپور
۷۶- برنامه‌نویسی شبکه در محیط .NET	مترجم: مهندس رامین مولاناپور

RWTUV
دارای گواهینامه ایزو ۹۰۰۱/۲۰۰۰ در زمینه نشر کتاب و طراحی جلد

برنامه‌نویسی VBA در Excel

در این کتاب می‌آموزید:

- نوشتن و رفع اشکال کدهای VBA
- ایجاد کادرهای محاوره و نوار ابزارهای اختصاصی
- استفاده از مدل شیء Excel
- استفاده از فراخوان‌های API
- ایجاد و به کارگیری ماژول‌های کلاس
- ایجاد منوهای شخصی در ساختار منوی اصلی Excel
- گسترش و تقویت توابع محاسبه و جستجو
- نوشتن ماکروهایی که کاربرگ‌ها را به صورت کاملاً اختصاصی درمی‌آورند

Translated by: Javad Ghanbar

موسسه فرهنگی هنری دیباگران تهران
سعادت آباد - میدان کاج - سرو غربی
روبروی خیابان علامه - پلاک ۹۷
تلفن ۷-۲۲۰۹۱۳۲۲ دورنگار ۲۲۰۹۱۳۲۱
فروش اینترنتی www.mftshop.com
E-mail: publishing@mftmail.com
URL: www.mftsite.com

ISO 9001: 2000
In Publication &

ISBN 964-354-7