برنامه نویسی VBA در اکسل

نویسنده : فرشید میدانی

WWW.FARSARAN.COM

بخش اول، اصول برنامه نویسی VBA

با مطالعه این کتاب سلولهای اکسل دیگر برای شما مستطیل نخواهد بود.

بخش اول، ویرایش یکم

قیمت بخش اول: ۱۴ هزار تومان

کلیه حقوق مادی و معنوی این کتاب متعلق به موسسه فرساران تفکر است.

پیشاپیش از رفتار شرافتمندانه رعایت حقوق مادی و معنوی این کتاب و حمایتی که از آن می‌نمایید،

سپاسگذاریم.

کلیه حقوق مادی و معنوی این کتاب متعلق به موسسه فرساران است

اگر کتاب به هر شکلی بجز خرید مستقیم از وب سایت فرساران به دست شما رسیده است، لطفا با مراجعه به وب سایت فرساران به آدرس زیر ، وجه کتاب را پرداخت نمایید:

www.farsaran.com

مشخصات کتاب
عنوان : برنامه نویسی VBA در اکسل (بخش اول)
نویسنده : فرشید میدانی
ویرایش : یکم (اسفند ۱۳۹۵)
قیمت بخش اول : ۱۴ هزار تومان
ناشر نسخه الکترونیک: موسسه فرساران تفکر

اجازه استفاده از این کتاب فقط متعلق به خریدار می‌باشد و خریدار می‌تواند کتاب را برای استفاده شخصی خود در یک نسخه پیرینت نماید . اگر دوستان، همکاران و یا هر فرد دیگری خواستار مطالعه این کتاب است، لطفا از آنها بخواهید که کتاب را از طریق وب سایت فرساران (بخش محصولات) خریداری نمایید.

توجه داشته باشید که فروش، دانلود، چاپ، پخش، به اشتراک گذاری و همچنین هر گونه استفاده دیگری از این کتاب و یا بخشی از آن به هر شکل و یا هر روشی غیر مجاز می‌باشد.

همچنین این کتاب به مدت یکسال دارای گارانتی عودت وجه پرداختی می‌باشد.

درباره بخش اول کتاب:

در بخش اول این کتاب که در حال حاضر پیش روی شما است، اصول برنامه نویسی VBA در اکسل را خواهید آموخت. این اصول کلی هستند و چه با اکسل و یا حتی اکسس برنامه نویسی کنید، باید بر این اصول مسلط باشید.

قصد دارم **در بخش دوم کتاب** به صورت ویژه برنامه نویسی در اکسل را آموزش دهم. اما هنوز بخش دوم آماده نیست.

برنامه من آن است که در نیمه اول سال ۹۶ نگارش بخش دوم را آماده سازم و به شما ارائه کنم. اما واقعا نگارش کتاب کاری سخت است و اصلا مطمئن نیستم که اتمام آن بخش چه موقعی خواهد بود.

به همین دلیل از ابتدا تصمیم داشتم که این کتاب را مرحله به مرحله کامل نمایم و آنچه انجام شده است و را در هر مرحله از طریق وب سایت فرساران ارائه نماییم.

گمان می‌کنم که بخش اول این کتاب و بدون بخش دوم می‌تواند مفید واقع شود و شروع خوبی برای یاگیری برنامه نویسی اکسل باشد.

بنابراین منتظر بخش دوم کتاب باشید و همچنین مرا از نقدها و نظرات خودتان آگاه نمایید.

فرشـــید میـــدانی

f.meidani@farsaran.com

فهرست مطالب

مقدمه ۹

تقدیم این کتاب ۹
قبل از شروع ۹
تصمیم گرفتی که برنامه نویس بشی ۱۰
نگرانی من ۱۱
پیش فرض های من درباره شما ۱۲
قوائد نگارشی کتاب ۱۲
مثال های کتاب ۱۳
حمایت از این کتاب ۱۳

فصل صفرم - شروع داستان VBA ۱۴

معرفی زبان VBA ۱۴
زبان VBA به چه دردی می خورد؟ ۱۵
چند مزیت و ایراد VBA ۱۵

فصل دوم - آشنایی با دنیای VBA ۱۷

معرفی واژه ماکرو ۱۷
تنظیمات امنیتی ماکروها ۱۷
فعال کردن یک ماکرو ۱۹
آشنایی با VISUAL BASIC EDITOR ۲۱
وارد شدن به محیط VBE ۲۱
اجزای محیط VBE ۲۴
تنظیمات محیط VBE ۲۶
معرفی VBAPROJECT ۲۷
تنظیمات VBAPROJECT ۲۸
بررسی قفل گذاشتن روی VBPROJECT ۲۹
معرفی قسمت IMMEDIATE ۲۹
ساختن MODULE یک قدم کوچک و مهم ۳۰
پروسیجر چیست ؟ ۳۲
ریشه شناسی واژه PROCEDURE ۳۳
قوانین نامگذاری پروسیجرها ۳۵
نوشتن اولین برنامه – HELLO WORLD ۳۶
اجرای یک پروسیجر ۳۷
روش اول - استفاده از کلید F5 ..... ۳۷
روش دوم – دکمه RUN ۳۸
روش سوم – منویRUN ۳۸
روش چهارم – از داخل EXCEL ۳۸
روش چهار و نیم – اجرا با کیبورد ۳۹
روش پنجم – با کلیک بر روی یک چیز ۴۰

روش ششم - اجرا از داخل یک پروسیجر دیگر ۴۰
ذخیره فایل ۴۱
باز کردن یک فایل دارای ماکرو ۴۳

فصل سوم - دستورات پایه زبان VBA ۴۴

قوائد و قوانین کلی دستورات ۴۴
منظور ما از دستور چیست ۴۴
حروف بزرگ و حرف کوچک در دستورات ۴۵
رنگ دستورات ۴۶
گذاشتن SPACE بین دستورات ۴۶
استفاده از TAB برای تو رفتگی دستورات ۴۷
استفاده از CTRL+SPACE ۴۸
توضیحات یا COMMENT ۴۸
یک سطری کردن ۵۰
چند سطری کردن ۵۰
معرفی دستور DEBUG.PRINT ۵۱
عملگرها و تقدم عملیات ها ۵۳
متغیر (VARIABLE) چیست ۵۴
نامگذاری متغیر ها ۵۶
آشنایی با DATA TYPE ۵۷
تعریف یک متغیر ۵۹
مزایای DECLARE (تعریف) کردن یک متغیر ۶۰
اجباری کردن تعریف متغیرها ۶۱
تخصیص یک مقدار به یک متغیر ۶۲
میدان دید متغیرها ۶۲
میدان دید محدود به یک پروسیجر ۶۳
میدان دید محدود به ماژول ۶۴
میدان دید در همه ماژول ها ۶۵
تعریف مقادیر ثابت (CONSTANTS) ۶۵
استفاده از مقادیر ثابت پیش فرض ۶۶
تاریخ میلادی ۶۷
استفاده از توابع VBA ۶۷
معرفی تابع MSGBOX ۶۸
معرفی تابع INPUTBOX ۷۱

فصل چهارم – دستورات شرطی و کنترل برنامه ۷۲

معرفی دستور GOTO ۷۲
با کلید F8 آشنا شوید ۷۳
آشنایی با IF ۷۴
مثال ۱ دستور IF – قبول شدید ۷۵
مثال ۲ دستور IF – ازدواج کردی ۷۵
مثال ۳ دستور IF – عدد زوج یا فرد ۷۶
مثال ۴ دستور IF – بخش پذیری ۷۷
آشنایی با عملگرهای منطقی ۷۷

آشنایی با AND ۷۷
مثال ۵ دستور IF – مثلث متساوی الاضلاع ۷۸
آشنایی با OR ۷۹
مثال ۵ دستور IF – مثلث متساوی الساقین ۷۹
مثال ۶ دستور IF – هوای مناسب ۸۰
آشنایی با NOT ۸۰
مثال ۷ دستور IF – خاموش / روشن کردن ۸۰
سایر عملگرها ۸۱
حروف کوچک و بزرگ در IF ۸۲
دستور IF بلوکی ۸۳
مثال ۸ دستور IF – رسم مستطیل یا مثلث ۸۴
استفاده از ELSEIF ۸۴
مثال ۹ دستور IF – مقایسه دو عدد ۸۵
مثال ۱۰ دستور IF - جایزه سیب ۸۶
IF های تو در تو ۸۷
آشنایی با BOOLEAN ۸۷
تابع IIF ۸۸
دستور SELECT CASE ۸۹
مثال ۲ دستور SELECT CASE – فصل سال ۸۹
مثال ۲ دستور SELECT CASE – جایزه سیب ۹۲
انواع شرط‌های SELECT CASE ۹۳

فصل پنجم – حلقه‌ها ۹۵

ایجاد تکرار با GOTO- روشی اشتباه ۹۵
تفسیر دستور X = X +1 ..... ۹۷
مشاهده مقدار متغیر با TOOLTIP ۱۰۰
مشاهده مقدار متغیر با WATCH ۱۰۰
چرا استفاده از GOTO اشتباه است ۱۰۱
دستور FOR – NEXT ۱۰۳
نقش متغیر در دستور FOR – NEXT ۱۰۴
مثال ۱ دستور FOR NEXT - جذر ۱۰۵
مثال ۲ دستور FOR NEXT – اعداد زوج ۱۰۵
استفاده از STEP در FOR – NEXT ۱۰۶
مثال ۳ دستور FOR NEXT – خروج از حلقه ۱۰۷
مثال ۴ دستور FOR NEXT – جمع اعداد ۱۰۸
دستور DO – LOOP ۱۱۰
مثال ۱ دستور DO – LOOP – پیغام سلام ۱۱۰
معرفی عبارت WHILE در دستور DO - LOOP ۱۱۱
مثال ۲ دستور DO – LOOP – برنده شدید ۱۱۱
مثال ۳ دستور DO – LOOP – اعداد دو رقمی ۱۱۲
شرط در ابتدا یا انتهای بلوک DO - LOOP ۱۱۲
مثال ۴ دستور DO – LOOP – خواندن یک فایل ۱۱۳
مثال ۵ دستور DO – LOOP – فاکتوریل ۱۱۵
دستور EXIT DO ۱۱۷
مثال ۶ دستور DO – LOOP – ک م م ۱۱۷
آشنایی با عبارت UNTIL در دستور DO – LOOP ۱۲۰

جمع بندی دستور Do - Loop ۱۲۱
حلقه‌های تو در تو ۱۲۲
مثال ۱ حلقه‌های تو در تو – مستطیل ۱۲۲
مثال ۲ حلقه‌های تو در تو – جدول ضرب ۱۲۳
مثال ۳ حلقه‌های تو در تو – اعداد اول ۱۲۵
مثال ۴ حلقه‌های تو در تو – چاپ آرام زیگزاگ ۱۲۶

فصل ششم - خطاها و مدیریت آنها ۱۲۹

خطاهای نوشتاری - Syntax Error ۱۳۰
خطاهای درهنگام کار برنامه– Run Time Error ۱۳۲
مثال ۱ Error Handling – تقسیم بر صفر ۱۳۲
چند توصیه برای پیشگیری از خطاها ۱۳۴
مدیریت خطاها با دستور On Error GoTo ۱۳۴
مثال ۲ Error Handling – دستور On Error GoTo ۱۳۵
دستور On Error GoTo 0 ..... ۱۳۷
بی‌خیال خطاها – On Error Resume Next ۱۳۷
مثال ۳ Error Handling – حذف شیت اکسل ۱۳۸
مروری بر ابزارهای Debugging در VBA ۱۳۹
دستور Stop ۱۳۹
استفاده از Break Point ۱۴۰

فصل هفتم - پروسیجرها و توابع ۱۴۱

پروسیجرها و برنامه نویسی ساخت یافته ۱۴۱
یادآوری مطالبی قبلی ۱۴۲
میدان دید پروسیجرها ۱۴۲
تعریف پروسیجر از نوع Public ۱۴۲
تعریف پروسیجر از نوع Private ۱۴۳
دادن ورودی به پروسیجر ۱۴۳
کاربرد پروسیجرها با ورودی ۱۴۴
با توابع آشنا شوید ۱۴۵
مقایسه توابع و پروسیجر ها ۱۴۶
کلیات ساخت و استفاده از یک تابع ۱۴۶
Call کردن توابع ۱۴۷
مثال ۱ توابع - تابعی بدون ورودی و بدون خروجی ۱۴۷
میدان دید توابع ۱۴۹
تعریف خروجی یک تابع ۱۴۹
مثال ۲ توابع - اسم من ۱۵۰
مثال ۳ توابع - فقط عدد طبیعی ۱۵۱
ورودی توابع ۱۵۱
مثال ۴ توابع - مکعب ۱۵۱
مثال ۵ توابع - طول وتر مثلث ۱۵۲
ورودی اختیاری توابع ۱۵۳
مثال ۶ توابع - فصل های سال ۱۵۴

تابع با تعداد ورودی زیاد ۱۵۵
مثال ۶ توابع - ادغام متن سلول‌ها ۱۵۵
مثال‌های کاربردی از توابع ۱۵۷
مثال ۷ - تابعی برای تجزیه یک متن به حروف ۱۵۷
مثال ۸ - تابعی برای استخراج یک عدد از متن ۱۵۸
مثال 9- حذف حروف صدا دار ۱۵۹
معرفی عملگر LIKE ۱۵۹
ادامه مثال ۹ - حذف حروف صدا دار ۱۶۱

فصل هشتم - ARRAY ۱۶۳

تعریف متغیری از نوع آرایه ۱۶۴
کار با اعضای یک آرایه، تکنیک FOR - NEXT ۱۶۵
کار با اعضای یک آرایه، تکنیک FOR - EACH ۱۶۷
مثال ۱ آرایه‌ها - بزرگترین عدد ۱۶۷
آرایه‌های داینامیک- متغیری از نوع VARIANT ۱۶۸
مثال ۲ آرایه‌ها - تجزیه یک متن ۱۶۹
مثال ۳ آرایه‌ها - استخراج کد ملی از شرح ۱۷۰
آرایه‌های داینامیک ۱۷۱
آرایه‌های دو بعدی (چند بعدی) ۱۷۲
آرایه‌ها و سلول‌های اکسل ۱۷۴
مثال ۴ آرایه‌ها- خواندن و نوشتن در سلول‌ها ۱۷۴
آرایه‌ها به عنوان ورودی یک تابع ۱۷۶
ورودی تابع از نوع VARIANT ۱۷۷
مثال ۴ آرایهها - مرتب سازی ۱۷۸

مقدمه

قبل از هر چیز از شما برای خریداری این کتاب تشکر می کنم و امیدوارم که آموزش های آن بتواند سریع و دقیق شما را به مقصودتان برساند. نگران نباشید اگر شما با دنیای برنامه نویسی آشنا نیستید. دراین کتاب سعی می‌کنیم که این دنیای جالب را به شما با حوصله نشان دهم و اگر از واژه‌ای فنی استفاده می کنم تلاش خواهم کرد که آنرا با مثال های روزمره برای شما شفاف کنم.

تقدیم این کتاب

سال‌ها پیش وقتی هنگامی که من در دبیرستان درس می‌خواندم، دوست بسیار عزیزی و گرامی داشتم به نام کریم رعنا حسینی. فردی دوست داشتنی و کتاب خوان.

روزی با کریم به شوخی و جدی قرار گذاشتیم که هر کسی کتابی نوشت، به دیگری تقدیم کند. اگر چه کریم سال‌هاست که دیگر در این دنیا نیست، اما همواره من به یادش هستم و به قول شاعر "من از یادت نمی‌کاهم".

حال این کتاب را که اولین کتاب مهم و جدی است که نوشته‌ام را به او تقدیم می‌کنم.

قبل از شروع

کتاب های فارسی برنامه نویسی VBA و برنامه نویسی اکسل انگشت شمار هستند و با هدف نگارش کتابی قابل فهم برای کاربران اکسل اقدام به نوشتن این کتاب کردم. برخی از ویژگی های این کتاب به شرح زیر است:

- سالها تجربه کار و تدریس نویسنده (یعنی من) با اکسل در آن منعکس شده است
- لازم نیست که شما هیچ پیش زمینه ی برنامه نویسی داشته باشید
- کتاب به سبکی آموزشی ما ایرانی ها نگاشته شده است
- از چندین منبع انگلیسی برای نوشتن آن تا حد ممکن تقلب کرده‌ام
- غلط گیری و به روزرسانی سریع و ساده آن
- مطالب با حوصله و جامع گفته شده است
- مطالب را با علاقه و انگیزه زیادی تدریس کرده‌ام

من تجربه سالها برنامه نویسی و انجام پروژه های متفاوتی را در اکسل دارم و بارها دوره‌های برنامه نویسی حضوری را در شرکت ها تدریس کرده ام و تجربه تدریس نزدیک به ۲۰ سال تدریس از زمان DOS تا کنون را سعی کرده ام در این کتاب پیاده کنم .

کتاب های انگلیسی بی نظیر هستند و من قطعا از بهترین آنها استفاده کرده ام تامطالبم را انسجام بیشتری ببخشم و کتابی کامل تری را برای شما مهیا سازم.

در ضمن اینکه این کتاب به صورت PDF است و قاعدتا اگر استقبال و حمایت شود، می‌توانم آنرا به روزرسانی و تکمیل تر کنم پس لطفا از آن با هر شکلی که ممکن است حمایت کنید.

اما در خصوص اینکه مطالب کتاب جامع، کامل و باحوصله است باید بگویم که خود من از روی اینترنت کتابهای فارسی زیادی را دانلود کرده‌ام و البته برخی از آنها مانند این کتاب رایگان هم

نبودند و مشکلی که همه آنها داشتند یک چیز بود! در اوایل کتاب نویسنده با حوصله مطالب را درس می‌دهد اما هر چه کتاب جلوتر می‌رود و مطالب سخت‌تر و پیچیده‌تر می‌شوند، نویسنده بی حوصله می‌شود و خیلی اجمالی مطالب را می‌گوید. من در این کتاب دقیقا سعی کردم که همواره توضیحاتم را کامل بدهم حتی اگر موفق نشوم که کتاب را هیچوقت تکمیل کنم.

شاید افرادی بگویند که این روش نوشتن کتاب که همه چیز را توضیح بدهند خوب نیست و باعث می‌شود که حجم مطالب زیاد شود و شاید خواننده خسته، ناامید و ... بشود.

من با این رویکرد خلاصه گفتن مطالب ،در حوزه برنامه نویسی کاملا مخالفم، در واقع شاید در برخی از علوم بتوانیم کلیاتی را مطرح کنیم اما در برنامه نویسی اینکار باعث می‌شود که در ابتدا خواننده از پیشرفت سریع خودش خوشنود شود اما در اولین قدم‌ها، با مسائل پیچیده‌ای مواجه می‌شود که نمی‌تواند آنها را تفسیر کند و کاملا ممکن است برای همیشه برنامه نویسی را رها کند.

در ضمن اینکه اشاره می‌کنم در کتابهای برنامه نویسی پرفروش انگلیسی، نویسنده هیچ وقت از گفتن همه جانبه مطالب و ریزه‌کاری‌ها طفره نمی‌رود و یا بی حوصله نمی‌شود و تا حد ممکن به آنها می‌پردازد.

تصمیم گرفتی که برنامه نویس بشی ...

برنامه نویسی از اصولی ساده و پایه ای ساخته می شود که یادگیری آن سخت نیست و هر کسی می تواند به سادگی در طی چند ماه یا حداکثر یک سال به سطحی برسد که از عهده کارهای خودش برآید . اما روی دیگر سکه آن است که برنامه نویسی اکسل به شما کمک می کند تا کارهای و وظایف سازمانی خود را سریع تر انجام دهید، بدون اشتباه و این احتمالا یعنی شما بتوانید به موقع محل کارتان را ترک کنید و به منزل بروید و با خانواده باشید و بنابراین برنامه نویسی می‌تواند به زندگی شخصی شما هم کمک کند.

واقعا اعتقاد دارم که برنامه‌هایی که شما در سازمان های بزرگی مثل ایران مثل تامین اجتماعی، دارایی، بانک‌ها، شهرداری و ... می نویسید می تواند کارها را سریع و دقیق کند و در نهایت کشوری منظم تر داشته باشیم و مردمی خوشحال تر داشته باشیم.

خیلی از مواقع ممکن است یک فرد در چند ماه یک برنامه کوچک تولید کند و آن برنامه سالها در آن سازمان بکار گرفته شود و کارهای تکراری حذف شود و دقت بالا برود.

شاید بدانید که من برنامه تاریخ هجری در اکسل را از سال ۸۵ بر روی وبلاگم قرار دادم و شاید این برنامه ساده چقدر توانست مفید واقع شود تقریبا همه جا از آن استفاده می کنند.

پس به خاطر خودتان یا خانواده تان و یا کشورتان لطفا کمی برنامه نویسی اکسل را یاد بگیرید .

البته برنامه نویسی اکسل مزایای زیر را نیز دارد:

- باعث می شود که در سازمان فردی کارآمدتر جلوه کنید
- می توانید باعث صرفه جویی در منابع مالی سازمان شود یا حتی در منابع انسانی
- حس شادی ناشی از خلاقیت و نوشتن یک برنامه دست کم نگیرید.

نگرانی من

واقعیت آن است که شما قرار است یک تکنیک و یک قدرت بالایی را در دنیای کامپیوتر کسب کنید. تا الان کاربر عادی / User بوده اید و حالا قرار است که یک کاربر حرفه ای شود و این یعنی باید چیزهای زیادی را بیاموزید.

این چیزها واقعا سخت نیستند اما حجمشان کمی زیاد است و بجز حوصله چاره دیگری ندارید. من مجبورم که مفاهیم زیادی را به شما بیاموزم و از آنجایی که احتمال دارد شما خسته و یا ناامید شوید، مرا نگران می‌کند.

اگر در فصل هایی خسته و یا نامید شدید، خیلی خوب مطالب را نفهمیدید، اصلا نگران نباشید این یک پدیده کاملا عادی است و می توانیم پیشنهاد بدهم که با سرعت کمتری مطالب را بخوانید و فصل را مرور کنید و یادداشت بنویسید و یا حتی چند روزی مطالعه را متوقف کنید.

اگر در فهم مطلبی مشکلی داشتید و آنرا دو یا سه بار خواندید و باز هم متوجه نشدید، حق ندارید به هوش و یا استعداد خودتان شک کنید بلکه بهترین شک آن است که من به عنوان نویسنده و معلم شما نتوانسته ام آن مطلب را درست آموزش دهم. پس اولا به من ایمیل بزنید و شکایت کنید و حتما پیگیری کنید تا آنرا درست کنم و دوما تا آن زمان بهترین کار آن است که چند ویدئوی آموزش از سایت youtube در خصوص آن مطلب را ببینید و یا اینکه یک کتاب انگلیسی مرجع را بخوانید.

نگرانی دیگری که دارم است آن است که این کتاب را فقط مطالعه کنید و مثال‌ها را انجام ندهید. واقعیت آن است که باید مثال‌ها را هر چند که بسیار در نظر شما ساده باشد، مجددا خودتان تایپ و اجرا کنید . مثال‌ها را تغییر بدهید و نتیجه‌ها را آزمایش کنید تا تصویر مبهم برنامه نویسی در ذهن شما به تدریج شفاف و کامل شود.

تاکید می کنم که قرار است شما مهارت سطح بالایی را یاد بگیرید و یادگیری بجز با حوصله، زحمت و پشتکار هیچ راه میانبری ندارد.

پیش فرض های من درباره شما

خوب حداقل باید موارد زیر را داشته باشید که بتوانید از کتاب استفاده کنید:

- قطعا اکسل را در حد متوسط باید بدانید و در غیر اینصورت بهتر است که ابتدا اکسل را یاد بگیرید. (از دوره های اینترنتی فرساران، دوره‌های حضوری و یا کتاب‌های ما برای اینکار استفاده کنید)
- کاربر دائمی اکسل هستید و در محیط کار به صورت مرتب با اکسل سر و کار دارید.
- قول می‌دهید که حداقل هفته ای ۱ ساعت مطالعه داشته باشید
- همت لازم برای تمرین و به اتمام رساندن این کتاب را داشته باشید.
- حوصله چند بار خواندن یک مطلب را دارید.
- مثال‌ها و تمرین‌های هر قسمت را حتی اگر خیلی ساده هم باشند، حتما اجرا می‌کنید.
- لازم نیست که قبلا برنامه نویسی کار کرده باشید.
- نسخه اکسل شما مهم نیست.

قوائد نگارشی کتاب

من از فونت Consolas که فونت خوانایی است برای نوشتن برنامه ها استفاده میکنم و به تفاوت های عدد یک و صفر با حروف L و O انگلیسی دقت کنید.

```
WowooOoO 10+20
Salam Leila jan
```

ما در برنامه نویسی می توانیم که یک سطر را در چندین سطر بشکانیم و از آنجایی که ممکن است نوشتن یک سطر بلندتر از عرض صفحه کاغذ شود، من از این تکنیک یعنی شکاندن سطر استفاده می کنم. برای آنکه یک سطر را به قطعات کوچکتری تقسیم کنیم باید از یک علامت _ (آندر اسکور یا آندر لاین) که قبلش یک فاصله (اسپیس) گذاشته ایم استفاده نماییم:

```
Selection.PasteSpecial Paste:=xlValues, _
Operation:=xlNone, SkipBlanks:=False, _
Transpose:=False
```

قسمت‌های مهم یک برنامه را که باید به آن توجه کنید، برای شما به صورت **Bold** و *Italic* نمایش داده‌ام: (کلمه Like در مثال زیر و همچنین علامت‌های ستاره)

```
Debug.Print "Shahr Shiraz ziba" Like "*Shiraz*"
```

در هر قسمت از درس برای تاکید بر موضوعی و یا طبقه بندی ذهن شما نکته‌ها و توجه‌هایی درج شده است.

مثال های کتاب

مثال های حل شده هر فصل به صورت جداگانه ای برای شما آماده شده است که با همین کتاب آن فایل ها را دریافت کرده اید.

هر مثال یک نام دارد که در ابتدای نام مثال، نام آن فصل از کتاب که مثال در آن زده شده است، آمده است . مثلا تمامی مثال‌های فصل پنجم کتاب با Chapter5 شروع می‌شوند و نام اولین مثال فصل پنجم Chapter5_Ex1 است و دومین مثال فصل پنجم Chapter5_Ex2 خواهد بود. در برخی از موارد نام مثال را یک واژه قابل فهم گذاشته‌ایم.

توجه: فایل‌های مثال از چندین ماژول تشکیل شده است.

حمایت از این کتاب

اگر از این کتاب خوشتان آمد و آنرا مفید یافتید لطفا آنرا به هر شکلی به دیگران معرفی کنید مثلا لینک معرفی آنرا در وبلاگتان قرار دهید و یا یک ایمیل به همکاران سازمانتان بزنید.

توجه داشته باشید که نسخه اصل آن فقط متعلق به شماست و آنرا با دیگران به اشتراک نگذارید و اگر فردی از شما کتاب را خواست از او بخواهید که با خرید از سایت فرساران این کتاب را تهیه کند. در ضمن اگر مایل هستید می‌توانید فقط برای استفاده شخصی خودتان این کتاب را پیرینت نمایید.

معرفی این کتاب در **LinkedIn** نیز مزید امتنان خواهد بود. ☺

فصل صفرم - شروع داستان VBA

در این فصل می خواهیم اولین قدم ها را برای آشنایی با VBA انجام دهیم و قبل از هر چیز مختصری بگوییم که VBA چیست؟

معرفی زبان VBA

شرکت مایکروسافت سال ها یک زبان برنامه نویسی استاندارد به نام Visual Basic را تولید می‌کرده و به فروش می رسانده است . از روی زبان برنامه نویسی Visual Basic یک نسخه خاص به نام VBA نیز ساخت . بنابراین VBA فرزند زبان برنامه نویسی Visual Basic است.

VBA یک زبان برنامه نویسی است.

حالا که متوجه شدید VBA فرزند Visual Basic است طبیعی است که بگوییم تمام اصول و قوائد زبان مادری را به ارث برده است بنابراین هر کسی که اصول VBA را یاد بگیرد قطعا اصول زبان مادری را یاد گرفته است و همچنین بلعکس.

VBA مخفف واژه Visual Basic For Application است و هدف از این نامگذاری آن است که برنامه-های VBA در داخل یک نرم افزار (Application) «میزبان» مانند اکسل، اجرا می‌شوند و بدون این «میزبان» ما نمی‌توانیم برنامه‌های نوشته شده به زبان VBA را اجرا کنیم.

نکته ۱: لازم نیست کسی VBA را نصب یا دانلود کند. VBA در «داخل» سایر نرم افزار به رایگان گذاشته است. مثلا هنگامی که نرم افزار Excel را نصب می کنید ،در داخل آن VBA نیز نصب شده است.)

نکته ۲: من از واژه «در داخل» استفاده کردم و باید به شما بگویم که VBA به صورت مستقلی اجرا نمی شود و اجرای آن مستلزم آن است که شما اکسل را باز کرده باشید و سپس می توانید از داخل اکسل وارد محیط VBA شوید و برنامه نویسی کنید پس به دنبال گزینه VBA در داخل منوی Start ویندوز نگردید.

نکته ۳: شما در PowerPoint اسلاید دارید، در Word صفحه و در Excel شیت . خوب اگر شما با VBA برنامه‌ای بنویسید که در داخل Wordبا صفحات کار کند، کاملا بدیهی است که آن برنامه در داخل Excel اجرا نشود چون اکسل اساسا صفحه ندارد.

بنابراین اگرچه اصول کلی زبان VBA در تمامی برنامه ها یکی است اما برنامه ای که برای شیت های Excel نوشته شده است را نمی توانید در داخل PowerPoint کپی کنید و انتظار داشته باشید که با اسلایدهای پاورپوینت کار کند.

تذکر: بررسی یک زبان برنامه نویسی، بحثی پیچیده و فنی است و من در اینجا با رویکرد ساده‌گرایانه ای، ویژگی‌های VBA را برای شما توصیف کرده‌ام و برخی از موارد گفته شده در این چند سطر از نظر علمی اعتبار چندانی ندارد.

زبان VBA به چه دردی می خورد؟

بدیهی است که در اکسل شما هزاران قابلیت دارید و می توانید بسیاری از کارها را با اکسل و بدون VBA انجام دهید و حالا باید به شما بگوییم که چرا قرار است VBA هم یاد بگیرید.

الف) سپردن انجام کارهای که شما معمولا انجام می دهید

فرض کنید که یک گزارش هر ماه باید تهیه و به چند نفر ارسال شود، می توانیم خیلی از این فرآیند را برنامه نویسی کنیم و شما درگیر آن کارها نشوید و برنامه برای شما آنها را انجام دهد.

ب) تکرار یک عملیات تکراری

مثلا قرار است که بر روی ۱۲ شیت اکسل یک کاری را انجام دهید، خوب واقعا سخت نیست که برنامه ای بنویسید که اینکار را انجام دهد. در ضمن اگر موفق شدید که برای ۱۲ شیت اینکار را انجام دهید قطعا اکسل برای انجام این کار بر روی ۱۲۰ شیت و یا ۱۲۰۰ شیت اصلا خسته و ناراحت نخواهد شد.

ج) خلق ابزاری جدید

بالاخره ابزارهای اکسل محدود هستند و شما ممکن است یک تابع جدید و یا یک ابزار برای یک کار خاصی را لازم داشته باشید که اکسل آنرا ندارد.

د) کنترل سایر برنامه ها

با کد نویسی در داخل اکسل می توانید سایر برنامه ها را نیز کنترل کنید مثلا از اکسل به Outlook فرمان دهید که یک ایمیل برای کاوه ارسال کند و در داخل آن ایمیل مقدار آخرین موجودی انبارها را بنویس.

چند مزیت و ایراد VBA

الف) واقعا برای یادگیری VBA ساده است .

البته شما حرف من را شاید خیلی قبول نکنید که ساده است اما هنگامی که با مفاهیمی عجیبی در سایر زبان های برنامه نویسی مواجه می شود آنوقت است که می فهمیم VBA چقدر ساده است.

ب) استفاده از قابلیت های اکسل

خیلی از وقت ها ما کارها را برنامه نویسی نمی‌کنیم مثلا اگر بخواهیم یک لیست را Sort کنیم لازم نیست که برویم و الگوریتم های مرتب سازی استفاده کنیم . بهتر است سری به اینترنت بزنید تا متوجه شوید که چندین روش مختلف برای مرتب کردن یک لیست داریم .

فقط کافی است که از ابزار Sort اکسل استفاده کنیم و خود اکسل کارها را انجام می‌دهد.

ج) نیاز به فایل اکسل برای اجرا (عیب)

همواره یک برنامه VBA نیاز دارد تا اکسل بر روی سیستم نصب باشد تا اجرا شود و نمی توان فایل های اجرایی مانند exe ساخت.

د) ساده است و گاهی محدود

همانطور که گفتم VBA واقعا ساده است و خوب این ساده بودن به دلیل آن است که قرار نبوده کارهای عجیب و غریب با VBA انجام دهیم و در نتیجه باید بدانید که انجام کارهای «خاص» در VBA یا خیلی سخت و یا غیر ممکن است. پس بهتر است از VBA انتظارات درستی داشته باشیم.

مثلا انتظار اینکه شما بتوانید اطلاعات یک دستگاه بارکد خوان را با VBA بگیریم می تواند یک مثال خوب از کارهای سخت است . (نگفتم که نمی شود !)

هـ) غیر قابل بودن امکانات فروش و یا توزیع برنامه

خیلی این سوال می شود که چطور می توانم بر روی برنامه اکسلی که نوشته ام پسورد بگذارم و آنرا قفل کنم و یا محدودیت زمانی به آن بدهم!

اساسا اکسل و برنامه نویسی آن برای تولید نرم افزارهای اجرایی و سپس فروش و ... ساخته نشده است و طراح این سوالات در حوزه اکسل اشتباه است.

اگر در نظر دارید که برنامه ای با اکسل بنویسید و سپس روی آن قفل بگذارید و بعدش به فروش برسانید، همین الان بروید به سراغ سایر زبان های برنامه نویسی حرفه ای که برای اینکار ساخته شده اند و این کتاب را ببندید!

فصل دوم - آشنایی با دنیای VBA

حالا کم کم باید وارد کارهای عملی تری شویم و با دنیای VBA آشنا شویم.

معرفی واژه ماکرو

هدف اصلی ما در این کتاب نوشتن یک «برنامه» است و البته واژه های دیگری هم برای توصیف اینکار یعنی «برنامه» به کار می رود که معادل همان واژه برنامه یا برنامه نویسی است. یکی از آن واژه ها ماکرو / Macro است. پس اگر جایی من گفتم یک ماکرو داریم یعنی یک برنامه داریم.

البته معادل واژه «برنامه»، واژه‌های دیگری را در کتاب‌ها، مقالات و ... مشاهده می‌کنید که برخی از مشهورترین آنها را برای شما در زیر لیست کرده‌ام:

- برنامه = ماکرو = پروسیجر = کد = سابروتین
- برنامه نویسی = ماکرو نویسی = نوشتن یک پروسیجر = کد نویسی = ایجاد یک سابروتین

تنظیمات امنیتی ماکروها

مجبورم این تنظیم را در همین ابتدای کتاب بگویم.

فرض کنید که یک نفری یک فایل اکسل رو به شما داده است و شما هم از همه جا بی خبرید که آن فایل ماکرو (برنامه) دارد و فایل رو باز می کنید و به یکباره می بینید که کامپیوتر شما خاموش شد!!!

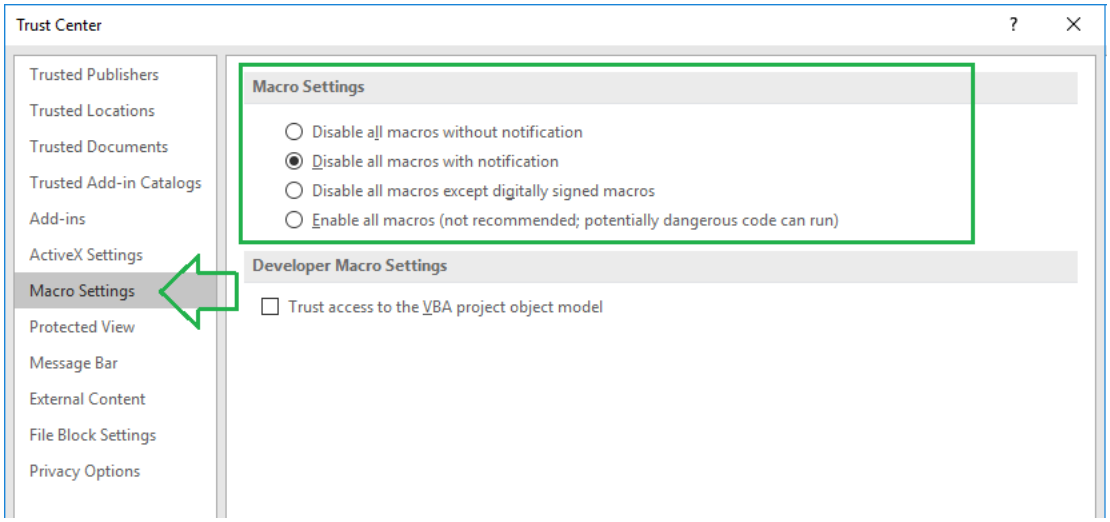
خیلی اتفاق عجیبی نیست وقتی که یک فایل برنامه/ماکرو دارد، قطعا می‌تواند کارهای مخربی را روی دستگاه شما انجام دهد.

به همین دلیل اکسل و سایر برنامه های آفیس به صورت پیش فرض هر فایلی را که ماکرو/برنامه داشته باشه رو قرنطینه یا Block می‌کند و برنامه‌های آن نتوانند اجرا شوند. به همین دلیل شما این فرصت را خواهید داشت که متن برنامه را مطالعه کنید و سپس تصمیم بگرید که آن برنامه بتواند اجرا شود و یا خیر.

این تنظیمات یعنی آیا قرنطینه شدن انجام بشود یا نه، در تنظیمات اکسل (Options)، در جایی به نام Macro Settings وجود دارد که از مسیر زیر قابل دستیابی است:

File → Options → Trust Center → Trust Center Settings → Macro Settings

تصویر تنظیمات امنیتی ماکروها



نکته: این تنظیمات هنگامی که یک فایل را باز می کنید تاثیر دارند و تعیین می کنند که آیا ماکروهایی در فایلی موجود است، هنگام باز شدن، در حالت قرنطینه / غیر فعال / یا Block قرار گیرد یا خیر.

در ادامه شرح هر یک از گزینه‌ها آورده شده است:

1) Disable all macros without notification:

یعنی همه ماکروها رو غیر فعال کند و به کاربر هیچ هشدار یا پیغامی مبنی بر اینکه ماکروها غیر فعال هستند، را نمایش نخواهد داد.

2) Disable all macros with notification:

این تنظیم پیش فرض اکسل است و می‌گوید همه ماکروها غیر فعال باشند اما به کاربر پیغامی نمایش بده که بگوید ماکروهای فایل، غیر فعال شده‌اند و قابل اجرا نمی‌باشند.

3) Disable all macros except digitally signed macros:

همه ماکروها غیر فعال باشند بجز آنهایی که امضا دیجیتالی شده‌اند. من در ایران ندیدم که سازمانی امضای دیجیتال مایکروسافت را پیاده سازی کرده باشد. در واقع این امضای دیجیتال توسط واحد IT قابل پیاده سازی است و افراد سازمان می‌توانند فایل‌های خود را امضا دیجیتالی کنند و البته آن امضا فقط در داخل آن سازمان اعتبار دارد. دقت داشته باشید که ما امضای دیجیتال بین المللی و .. هم داریم اما بهتر است این بحث رو ما ادامه ندهیم زیرا در دنیای اکسل تا آنجایی که من دیده‌ام، اصلا متداول نیست.

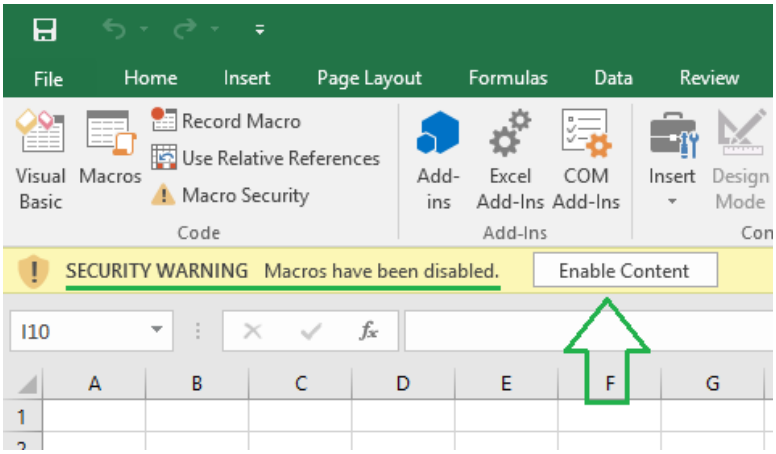
4) Enable all macros (not recommended; potentially dangerous code can run):

این گزینه یعنی ماکروهای همه فایل‌ها همواره فعال باشند و غیر فعال نباشد. به شما هشدار داده است از این گزینه استفاده نکنید زیرا که ممکن است کدهای خطرناکی اجرا بشوند.

فعال کردن یک ماکرو

در دلیل تنظیمات امنیتی ماکروها، اگر گزینه Disable all macros with notification را انتخاب کرده باشد، بلافاصله بعد از اینکه فایلی که ماکرو دارد را باز کنید، پیغام زرد رنگی مانند تصویر زیر به شما نمایش داده می‌شود:

هشدار قرنطینه/غیرفعال بودن ماکروی یک فایل

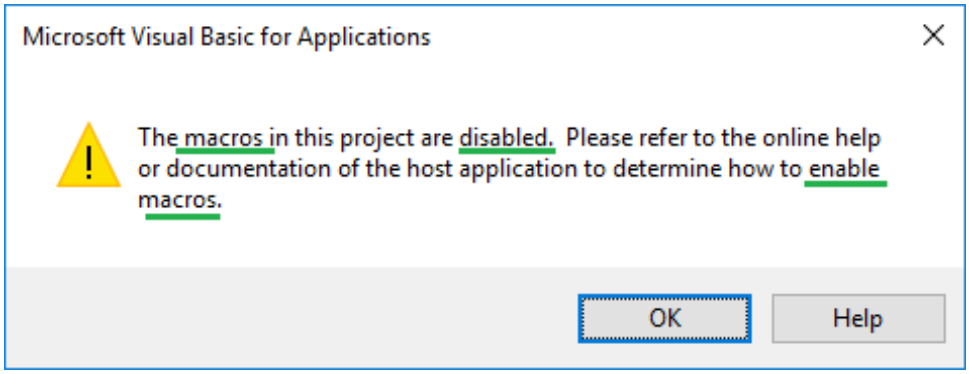


پیغام زرد رنگ رو اگر دقیقا بخوانید به شما گفته است که .Macros have been disables و معنای آن این است که :

۱) این فایل ماکرو دارد
۲) این ماکرو ممکن است که خطرناک باشد (Security Warning و علامت زرد سپر)
۳) ماکروی این فایل غیر فعال / قرنطینه شده است.
۴) چون ماکرو غیر فعال است بنابراین نمی توانید آن را اجرا کنید.
۵) اگر می خواهید با زدن کلید ALT+F11 وارد محیط VBE شوید و آنرا مطالعه کنید (آن جمله را خودم اضافه کردم و در پیغام این نوشته نشده است ☺)
۶) اگر هم که مطمئن هستید که این فایل معتبر است می توانید با زدن کلید Enable Macro، ماکرو را از حالت قرنطینه/غیرفعال خارج کنید و سپس آنرا اجرا نمایید.

اگر که شما به این پیغام زرد رنگ بی‌توجه باشید و گزینه Enable را نزنید، سپس بخواهید که یکی از ماکروهای این فایل را اجرا کنید، پیام زیر را به شما نمایش داده می‌شود.

تصویر پیغامی که به دلیل غیر فعال بودن ماکرو، قابل اجرا شدن نیست



ترجمه این پیغام:

"ماکروهای این پروژه (فایل) غیر فعال هستند. لطفا به راهنمای نرم افزار جهت فعال کردن ماکروها مراجعه کنید."

نکته ۱: از Excel 2010 به بعد، اگر یکبار برای فایلی Enable Macro را بزنید ، دیگر از شما این پیام را در مورد آن فایل نمی‌پرسد و در خاطرش می ماند که این فایل معتبر است.

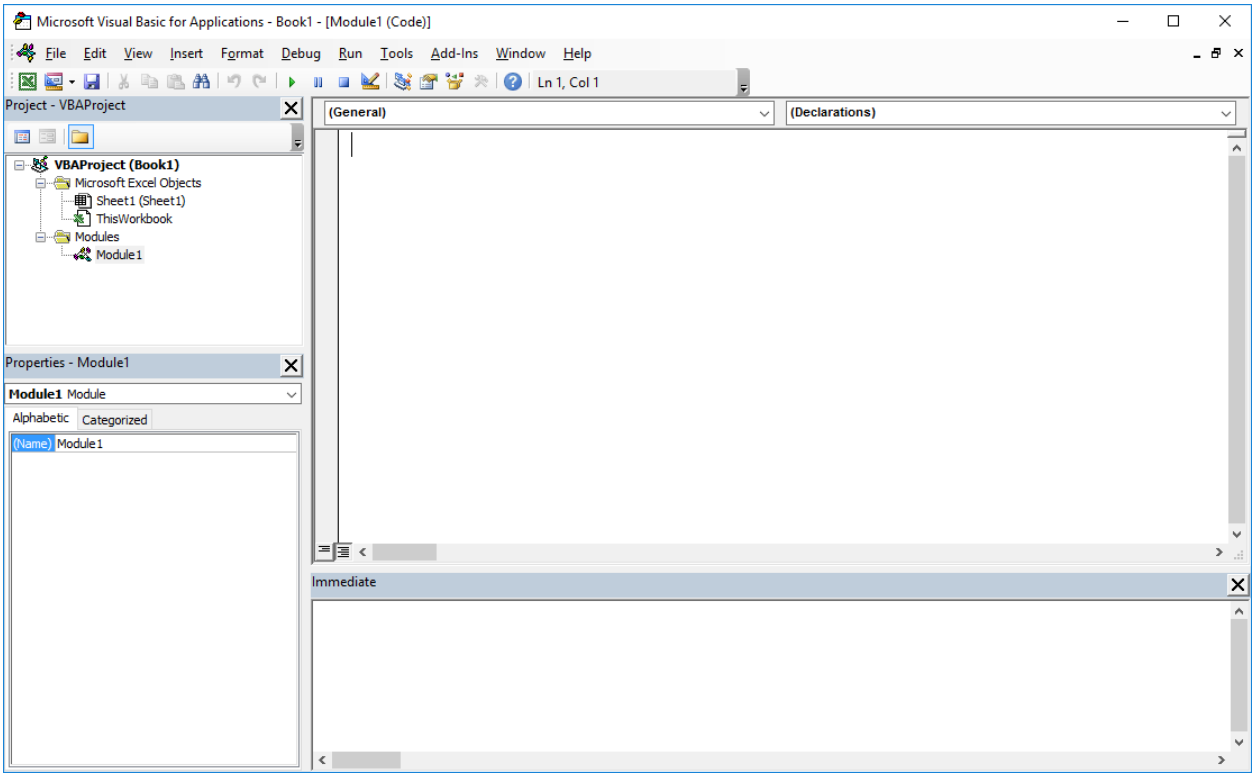
نکته ۲: اگر بلافاصله بعد از باز کردن یک فایل ماکروها را Enable نکنید، دیگر نمی‌توانید آنرا فعال کنید و باید فایل را ببندید و مجدد باز کنید تا پیغام برای شما نمایش داده شود.

توصیه : اگر هر روز با فقط با چند فایل که کاملا معتبر هستند کار دارید، توصیه میکنم که تنظیمات ماکرو رو در حالت همیشه فعال قرار دهید در ضمن آنکه توجه داشته باشید که ویندوز ، آنتی ویروس/ فایروال ها و تنظیماتی که واحد IT سازمان شما در نظر گرفته اند جلوی بسیاری از آسیب ها را خواهند گرفت.

آشنایی با Visual Basic Editor

باید بگویم که VBE همان جایی که شما باید برنامه‌های خود را تایپ خواهید کرد و VBE در واقع Visual Basic Editor است و Editor در دنیای کامپیوتر یعنی جایی که می‌توانید تایپ کنید.

عکس محیط VBE که در آن برنامه ها را تایپ خواهیم کرد



وارد شدن به محیط VBE

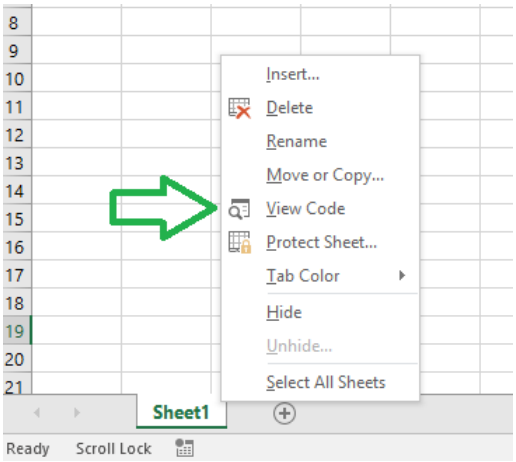
برای وارد شدن به محیط VBE ابتدا اکسل یا اکسس را اجرا می‌کنیم و سپس برای رفتن به محیط VBE چندین روش داریم :

۱) با زدن کلید ALT+F11 وارد محیط VBE خواهید شد.

توجه: اگر شما لپ تاپ دارید ممکن است که سازنده لپ تاپ شما استاندارد کلید F11 را عوض کرده باشد مثلا زدن F11 باعث بلند شدن صدا می شود و برای زدن کلید F11 واقعی باید کلید دیگری را نگه دارید . معمولا این کلید Fn است.

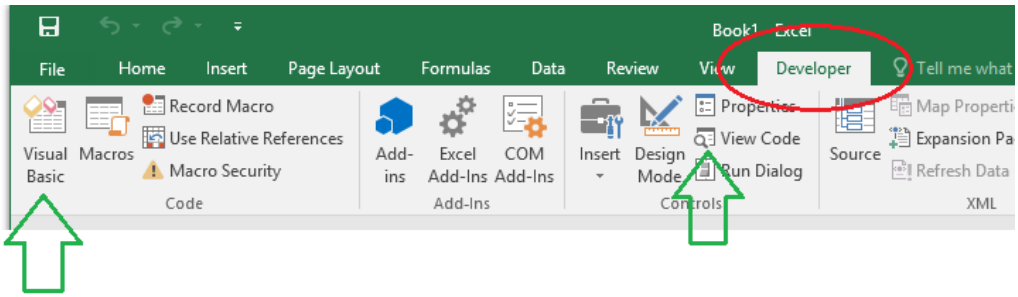
۲) اگر در اکسل روی یک نام شیت R-Click کنید و گزینه View Code را بزنید، وارد VBE خواهید شد.

عکس ورود به محیط VBE با R-Click روی یک نام یک شیت



۳) در اکسل از Developer Tab گزینه Visual Basic را انتخاب می‌کنیم.

عکس ورود به محیط VBE از Developer Tab

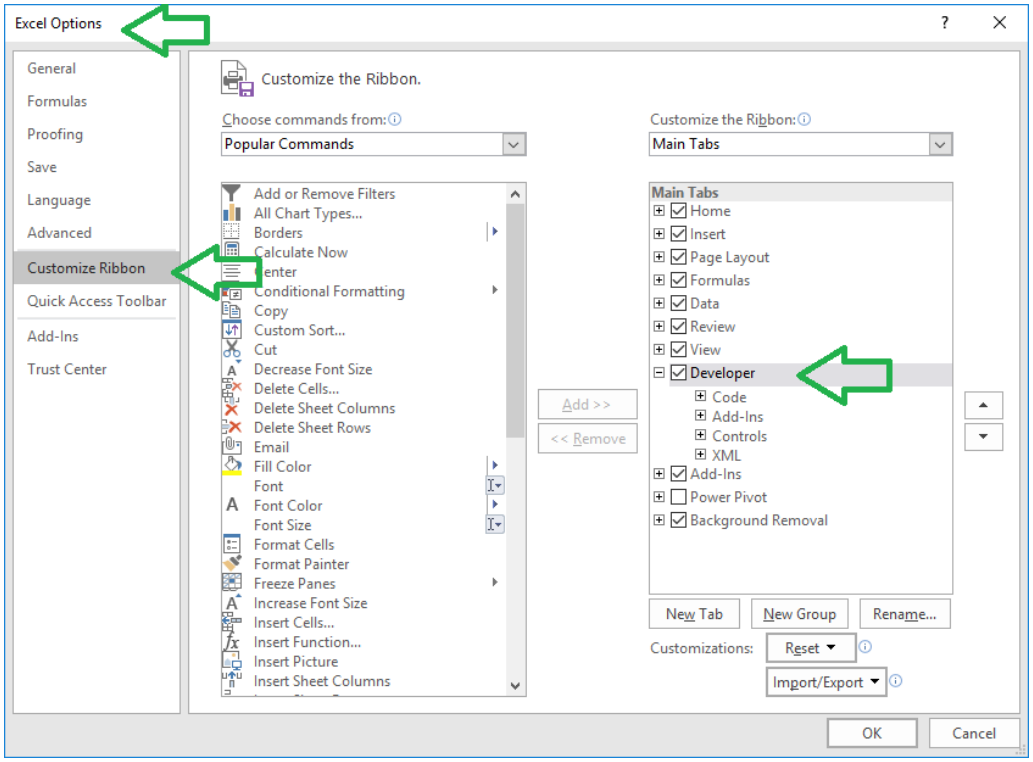


برای نمایش Developer Tab از مسیر زیر آنرا انتخاب کنید:

File → Options → Customize Ribbon

در نسخه Excel 2007 برای فعال کردن Developer Tab باید مسیری کمی متفاوت را طی کنید. برای یافتن آن یک گوگل کنید. (یعنی در گوگل جستجو کنید).

تصویر تنظیم فعال کردن نمایش Developer Tab در Ribbon



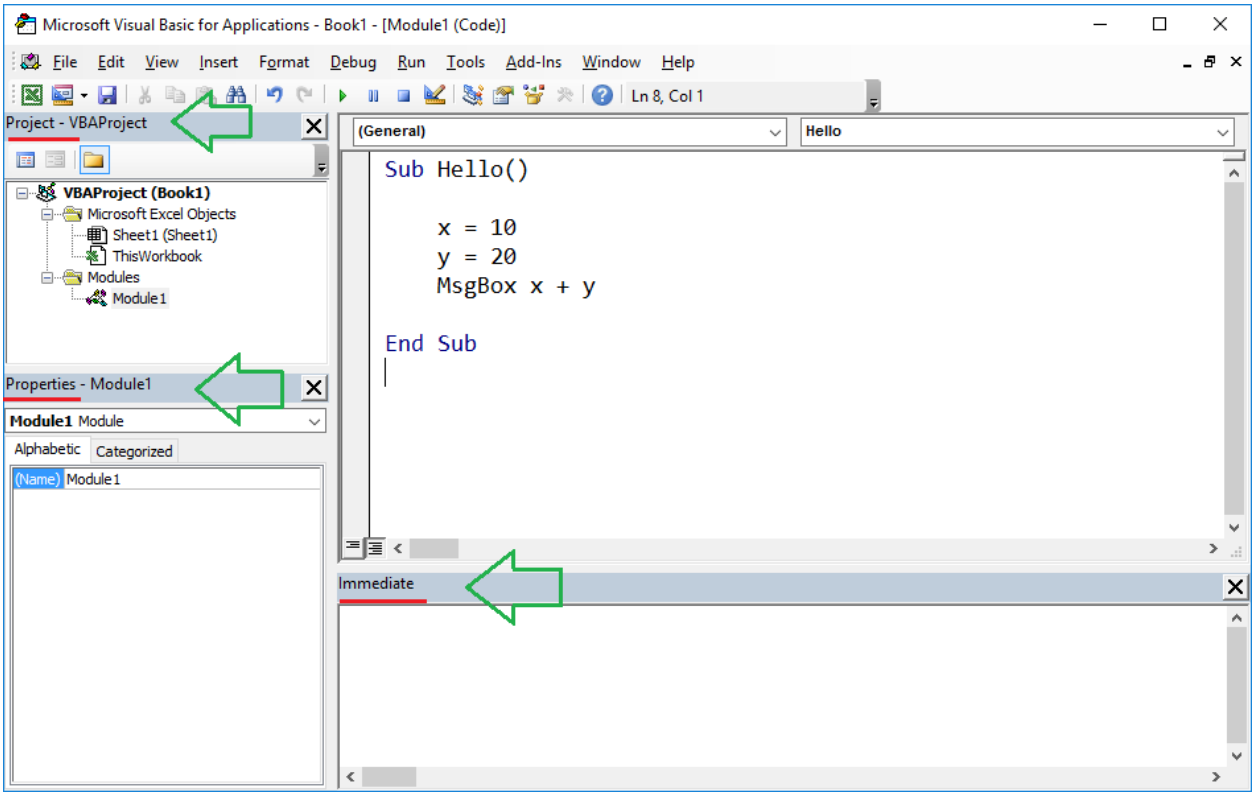
اجزای محیط VBE

در تصویر زیر من سه فلش گذاشته‌ام که هر کدام عنوان یک قسمت از محیط VBE هستند:

۱- قسمت Project
۲- قسمت Properties
۳- قسمت Immediate

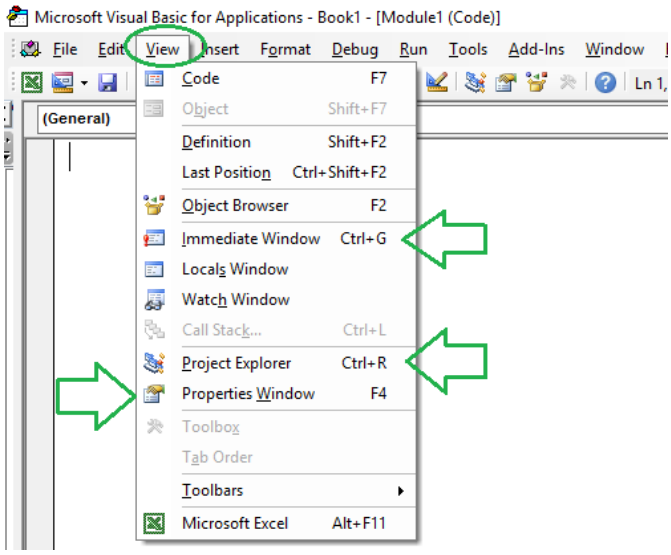
اگر خوب نگاه کنید هر یک از پنجره ها یک دکمه «ضربدر» که می توانیم آن قسمت را ببنیم و اگر آن قسمت با بستیم و خواستیم که مجدد آن را بیاوریم باید از منوی View این کار را انجام دهیم.

تصویری از محیط VBE



نکته: اگر احتمالا یکی از این قسمت‌ها را بر روی دستگاه خود مشاهده نمی‌کنید، می‌توانید از منوی View که در تصویر بعدی مشاهده می‌کنید، بر روی آنها کلیک کنید تا فعال شوند.

تصویری از منوی View



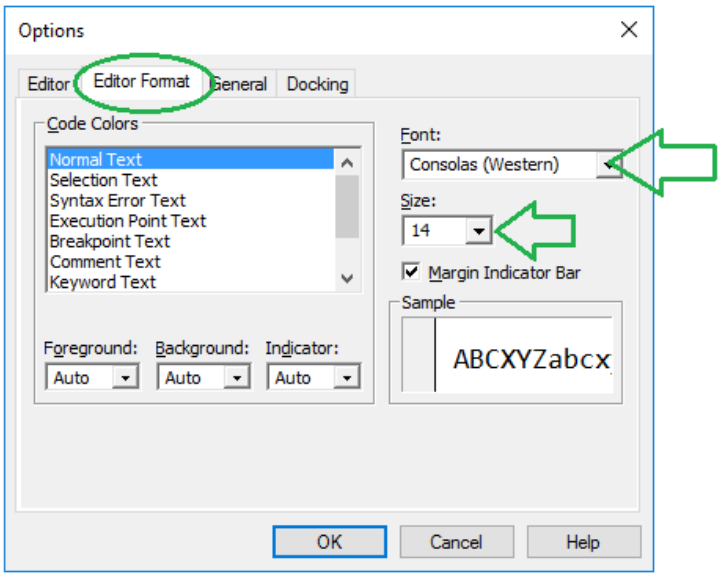
تمرین ۱ : لطفا یک نگاهی به همه منوها محیط VBE بیندازید. نگران نباشید این محیط بسیار ساده است.

تمرین۲ : پنجره Immediate را ببنید و سپس با کلید میانبر (shortcut key) مجددا آنرا باز کنید. (یعنی ممکنه کسی باشه که بپرسه که کلید میانبرش چیه؟؟؟)

تنظیمات محیط VBE

از مسیر Tools → Options می توانید تنظیمات محیط VBE را انجام دهید. بهتر است که شما فونت را کمی بزرگتر کنید و اگر از فونت Consolas خوشتان می آید آنرا به عنوان فونت این محیط انتخاب کنید.

تصویر تنظیمات فونت در Options



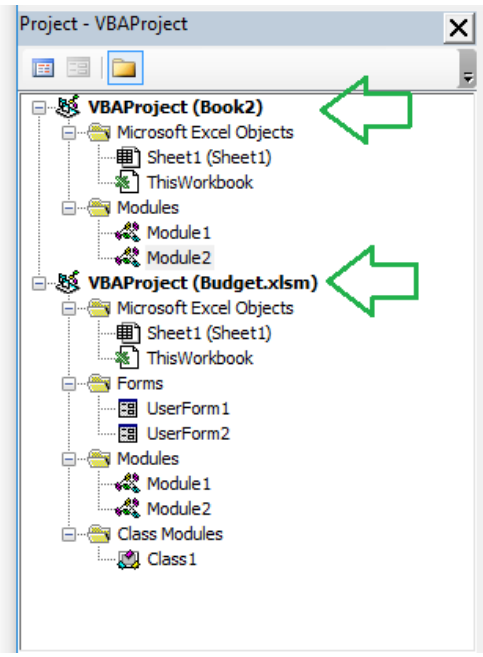
تمرین: یک نگاهی به همه گزینه های محیط Options بیاندازید . ما در این کتاب آنها را نخواهیم گفت.. چون تقریبا ساده هستند و قابل فهم و اگر گزینه ای برایتان جالب بود و متوجه اش نشدید، چرا آنرا گوگل نمی کنید.

معرفی VBAProject

تمام برنامه‌های شما در زیر مجموعه چیزی به عنوان VBAProject قرار می گیرند. می‌توانید VBAProject یک ظرف در نظر بگیرید که تمامی برنامه های شما در آن ریخته می شود.

لطفا به تصویر زیر دقت کنید. (دقت کنید یعنی تک به تک نوشته های آنرا بخوانید و ساختار آنرا تشخیص دهید)

تصویر VBAProject



تصویر نشان می دهدکه دو فایل به نام Budget , Book2 باز است و هر یک از این فایل‌ها دارای یک VBAProject است. در هر یک از VBAProjectها اجرایی مانند Module و یا UserForm ممکن است وجود داشته باشند.

نکته: VBProject اصلا واژه مهم و یا یک مفهوم کلیدی نیست.

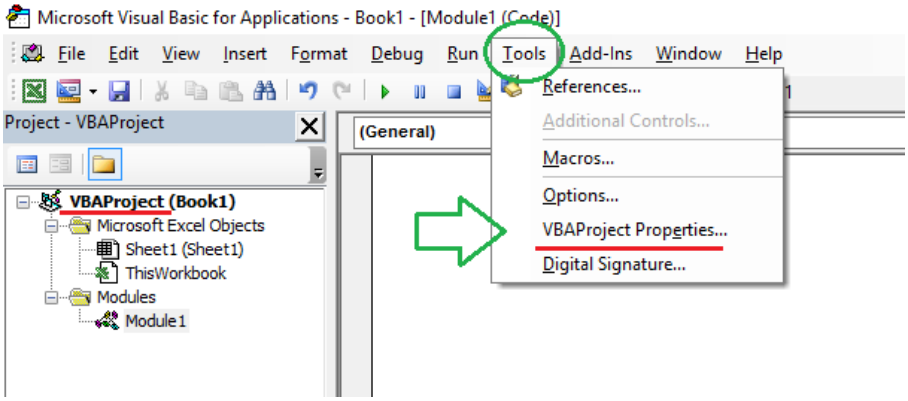
تنظیمات VBAProject

متوجه شدیم که هر فایل VBAProject خودش را دارد که در آن برنامه‌ها و سایر اجزای مربوط به برنامه نویسی در آن قرار می‌گیرید.

یکی از کاربردهای VBAProject آن است که برای مشاهده آن یک پسورد بگذاریم تا دیگران نتوانند متوجه شوند که در آن چیست و در نتیجه برنامه‌های ما را نخواهند دید.

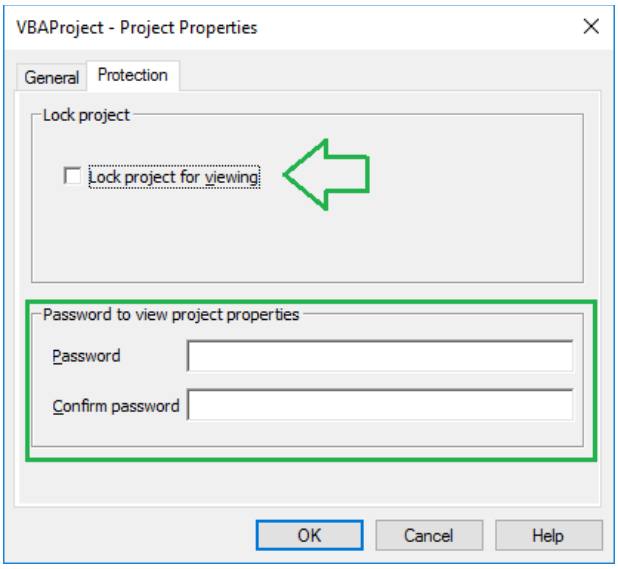
قبل از رفتن تنظیمات VBAProject هر فایل ابتدا مطمئن شوید که روی VBAProject آن فایل کلیک کرده اید و سپس از منوی Tools → VBAProject Properties را انتخاب کنید.

تصویر Tools → VBAProject Properties



قسمت General اگر مایل هستید یک اسم دلخواه برای VBAProject بگذارید و یا از قسمت Protection برای VBAProject یک پسورد قرار دهید و گزینه Lock Project for viewing را فعال کنیم تا دیگر کسی قادر به دیدن برنامه‌های ما نباشد.

تصویر تنظیم پسورد برای VBAProject یک فایل



بررسی قفل گذاشتن روی VBProject

اینجا من یک بخش مستقل باز کردم زیرا این سوال بارها و بارها پرسیده می شود که چطور می توانم از فایل اکسلی یا اکسسی که ماکرو دارد محافظت کنیم و برای کدهای آن قفل بگذاریم.

معمولا این سوال را کاربرانی از اکسل می پرسند که یک برنامه ای ساخته اند و گمان می کنند که با قفل گذاشتن و محدود ساختن آن می توانند آنرا به فروش برسانند.

متوجه شدیم که می‌توان بر روی VBAProject قفل گذاشت و در نتیجه کسی نمی تواند برنامه‌های ما را مشاهده کند

اما واقعیت در عمل چیز دیگری است و می‌توان پسورد هر VBAProject را به راحتی (مثل آب خوردن) را برداشت و کدها را دید و آنها را دستکاری کرد. اساسا قرار نبوده که اکسل و یا اکسس این قابلیت (یعنی تجاری سازی) را داشته باشد و به همین منظور ابزارهای برای تجاری سازی برنامه در آن تعبیه نشده است.

پس پسورد VBAProject جنبه بیشتر پیشگیرانه دارد که مبادا کاربری به اشتباه برنامه را دستکاری کند و هدف آن ایجاد ابزاری برای فروش نرم افزار نبوده است.

اگر به دنبال تجاری سازی برنامه‌های خود هستید پیشنهاد می‌کنم که به فکر استفاده از زبان‌ها و تکنولوژی‌های سطح بالاتری مانند Net. و یا PHP و .. باشید نه VBA .

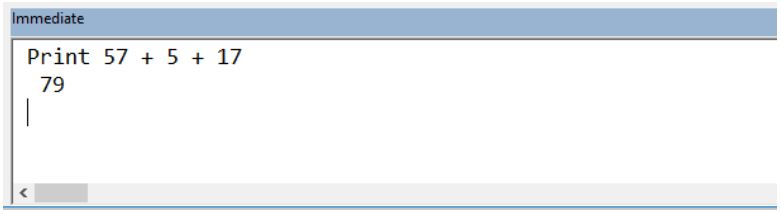
معرفی قسمت Immediate

همانطور که تصاویر قبلی دیدید در VBE یک قسمتی به نام Immediate وجود دارد. ما از Immediate برای آزمایش برنامه‌های خود استفاده می‌کنیم .

مثلا فرض کنید که در یک برنامه قرار است یک محاسبه ای انجام شود و بر اساس نتیجه آن محاسبه یک عملیات دیگری آغاز گردد. ما به عنوان برنامه نویس دوست داریم که در هنگام اجرای برنامه بدانیم که نتیجه آن محاسبه چیست و آنرا با چشم ببینیم. در این حالت می توانیم دستوری در برنامه اضافه کنیم تا نتیجه محاسبه را در قسمت Immediate نمایش دهد. (با این دستور بعدا آشنا خواهید شد.)

قسمت Immediate کار دیگری نیز انجام می دهد. هر دستوری در آن تایپ کنید و Enter را بزنید، آن دستور را اجرا می کند مثلا می خواهیم نتیجه محاسبه ۵۷ + ۵ + ۱۷ را ببینیم، کافی است دستور زیر را در Immediate تایپ کنیم و Enter را بزنیم:

تصویر قسمت Immediate و اجرای یک محاسبه ساده در آن



ساختن Module یک قدم کوچک و مهم

خلاصه کنم که تا اینجا ما فهمیدیم که برای نوشتن در یک فایل اکسل با زدن کلید ALT+F11 وارد محیط برنامه نویسی اکسل به نام VBE شویم.

حالا می خواهیم که یک برنامه بنویسیم. اما هنوز همه مقدمات را فراهم نکرده‌ایم.

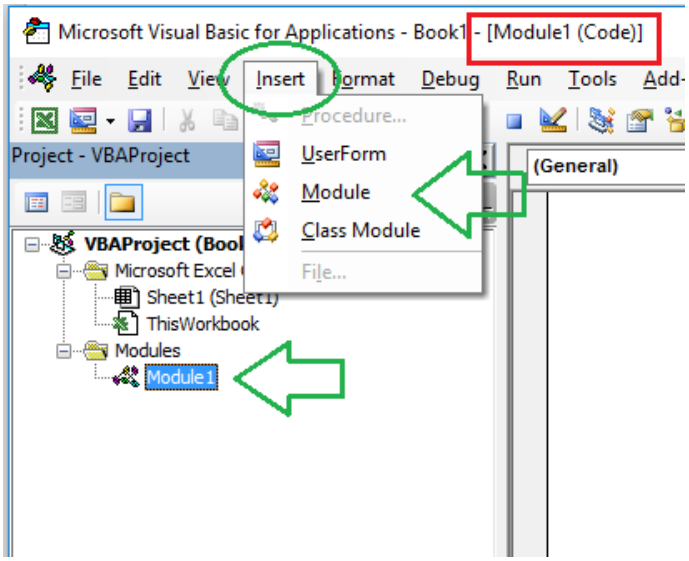
بگذارید یک مثال بزنم اگر بخواهیم یک نامه بنویسیم اول باید یک برگه کاغذ داشته باشیم و اگر بخواهیم برنامه ای بنویسیم اول باید یک برگه کاغذ داشته باشیم و به این برگه کاغذ در محیط برنامه نویسی می گویند Module.

ماژول یعنی چیزی شبیه اسلاید پاورپوینت / صفحه ورود / شیت اکسل که می توانیم در آن برنامه نویسی کنیم.

برای ساختن یک ماژول در VBE از منوی Insert گزینه Module را بزنید و سپس یک ماژول دارید.

می‌توانید هر چند تعداد ماژول که مایل هستید از منوی Insert ایجاد کنید و هیچ تفاوتی نمی کند که شما برنامه را در کدام ماژول بنویسید. هدف از ایجاد ماژول‌های متعدد، نظم بخشیدن است.

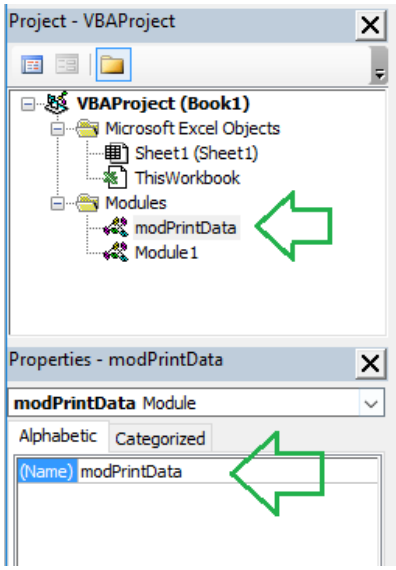
تصویر درج Module



در تصویر بالا یک ماژول به نام Module1 داریم و اگر دقت کنید در مستطیل قرمز رنگ نام آن ماژول نوشته شده است یعنی الان آن ماژول فعال است و کدهای ما در آن ماژول نوشته خواهد شد.

نام ماژول ها را هم می‌توان از پنجره Properties عوض کرد . به شکل بعدی دقت کنید.

تصویر تغییر نام یک ماژول

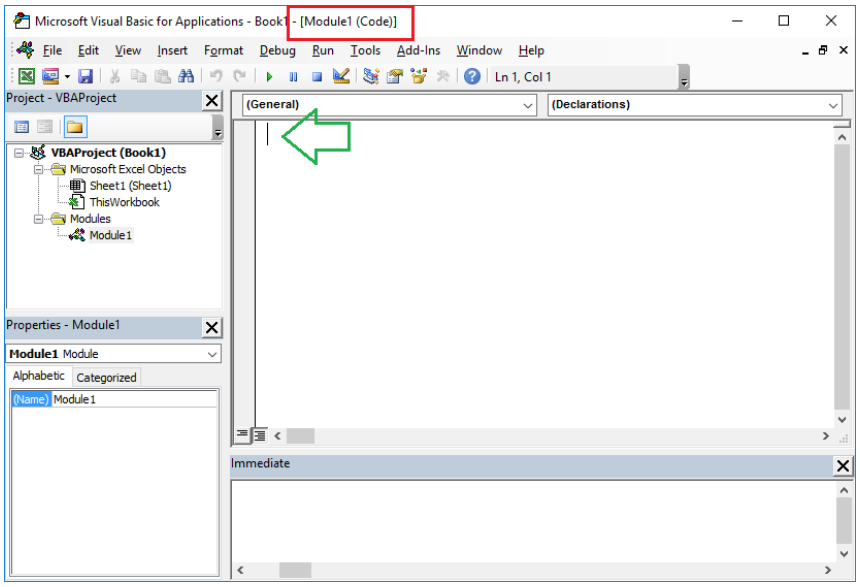


یادآوری: اگر قسمت Properties را ندارید از منوی View آنرا می توانید فعال کنید.

نکته: برای حذف یک ماژول کافی است روی آن R-Click کنید و گزینه ..... Remove را انتخاب کنید.

حال که ماژولی دارید،می‌توانید اولین برنامه خود را تایپ کنید.

تصویر VBE و علامت چشمک زن برای شروع تایپ



نکته: دقت کنید که حتما باید داخل ماژولی که ساخته‌اید باشد و نام ماژول را در قسمت Title Bar می‌توانید ببینید. (بر روی نام یک ماژول در قسمت Project ، D-Click کنید تا وارد آن شوید)

پروسیجر چیست ؟

ابتدا کارهایی را که تا اینجا انجام داده‌ایم را خلاصه کنم.

۱) یک فایل اکسل دارید
۲) با زدن ALT+F11 وارد محیط VBE شده‌اید
۳) از منوی Insert یک ماژول ساخته‌اید.

حالا باید یک پروسیجر بسازیم.

من از شما می پرسم قصیده چیست؟ و هر جوابی که شما بدهید من به شما خواهم گفت که فرض کنید که پروسیجر همان قصیده است. یعنی اگر سعدی برنامه نویس می‌شد به جای قصیده برای ما پروسیجر می نوشت.

خیلی وارد بحث های ادبیاتی قصیده نشوم اما در یک تعریف ساده ، قصیده نوعی از شعر است که معمولا ۲۰ تا ۷۰ بیت دارد و یا یک مطلع آغاز می شود و شاعر از سرودن آن قصیده ، مقصود خاصی داشته است مثلا می خواسته پند واندرزی بدهد.

وقتی که ما می خواهیم برنامه بنویسیم خوب بدیهی است که آن برنامه باید کاری (مقصود خاصی) را انجام دهد و از چندین دستور تشکیل می شود و آن برنامه باید از جایی شروع (مطلع) و از جای دیگری به پایان برسد تا قابل تفکیک از یک برنامه دیگر باشد و به این در دنیای برنامه نویسی «پروسیجر» می گویند.

بدیهی است شروع و پایان یک پروسیجر باید مشخص شود و اینکار در محیط VBA با دو دستور Sub و End Sub انجام می شود در ضمن آنکه باید برای هر پروسیجر یک نام هم بگذاریم.

به صورت کلی یک پروسیجر به نام Hello اینگونه خواهد شد:

```
Sub Hello()

End Sub
```

در داخل این پروسیجر ما دستورات VBA را خواهیم نوشت و سپس پروسیجر را اگر اجرا کنیم، آن دستورات یکی پس از دیگری و به ترتیبی که نوشته شده اند اجرا خواهند شد.

توجه ۱: اگر چه می توان صدها خط دستور در داخل یک پروسیجر نوشت، اما توصیه می شود که طول یک پروسیجر بیش از ارتفاعی که در مانتیور می‌توانید مشاهده کنید، نشود. یعنی بهتر است که کل پروسیجر را با یک نگاه روی مانیتورتان ببینید . یعنی حدود ۳۰ الی ۵۰ خط.

توجه ۲: ما یک برنامه بزرگ را به چندین پروسیجر تقسیم می‌کنیم تا مدیریت و کارهایمان نظم بیشتری داشته باشد. مثلا یک پروسیجر درست می‌کنیم که مرحله الف را انجام دهد و در پروسیجر دیگری کارهای پیرینت گرفتن و در یک پروسیجر هم کارهای مرحله آخر را.

توجه ۳: علامت ( ) اجباری است و باید گذاشته شود و دلیل آنرا بعدا خواهید فهمید.

ریشه شناسی واژه Procedure

قبل از ادامه کارهای عملی باید چند نکته در مورد واژه پروسیجر و سایر واژه‌هایی که معادل آن است را با شما در میان بگذارم:

نکته ۱:

که کلمه Sub مخفف واژه سابروتین نیست و در واقع Sub مخفف واژه Sub Procedure (ساب پروسیجر) است.

معمولا واژه Sub به معنی «زیر مجموعه» و یا «جزیی از یک کل» بکار می‌رود به همین دلیل اگر بگوییم Sub Procedure داریم، یعنی منطقا باید Procedure هم داشته باشیم. اما از نظر فنی ما در VBA چیزی به نام Procedure نداریم تا بخواهیم که آنرا به چندین Sub Procedure تقسیم بندی کنیم.

نکته دیگر آن است که در منابع علمی واژه Sub Procedure به چشم نمی‌خورد و اگر شما ساعت‌ها به گفت و گوی برنامه‌نویسان گوش دهید، احتمالا حتی یکبار واژه «ساب پروسیجر» را نخواهید و خواهید دید که برنامه نویسان از واژه هایی مانند «پروسیجر» و یا «سابروتین» استفاده می‌کنند.

بنابراین ما در این کتاب از واژه «ساب پروسیجر» استفاده نمی‌کنیم و به جای آن از واژه «پروسیجر» که پر کاربرد تر است استفاده خواهیم کرد.

نکته ۲:

جالب است بدانید که واژه «پروسیجر» هم چندان در دنیای برنامه نویسی استفاده نمی‌شود و معمولا به جای آن از واژه «**سابروتین**» استفاده می‌شود.

اگر در ویکی پدیا به دنبال واژه پروسیجر بگردید، متوجه می‌شوید که این واژه تعریفی ندارد! و در زیر مجموعه واژه سابروتین تعریف شده است:

تصویر تعریف واژه سابروتین در ویکی پدیا

In computer programming, a **subroutine** is a sequence of program instructions that perform a specific task, packaged as a unit. This unit can then be used in programs wherever that particular task should be performed. Subprograms may be defined within programs, or separately in libraries that can be used by multiple programs. In different programming languages, a subroutine may be called a **procedure**, a **function**, a **routine**, a method, or a **subprogram**. The generic term **callable unit** is sometimes used.[1]

اگر به جای واژه «پروسیجر» از واژه «سابروتین» استفاده کنیم، کاملا کار درست و علمی را انجام داده‌ایم و تعریف سابروتین عبارت است:

من در اولین ویرایش این کتاب از واژه «سابروتین» استفاده کردم اما تصمیم گرفتم که ویرایش دوم به دنیای VBA وفادار بمانم و از همان واژه «پروسیجر» استفاده کردم اگر چه اعتقاد دارم که واژه «سابروتین» علمی‌تر و معمول‌تر می‌باشد.

نکته ۳:

برای خوانایی این کتاب واژه انگلیسی Procedure را به صورت فارسی «پروسیجر» نوشته شده است و ممکن است گاهی شما آنرا «پراسیجر» هم بشنوید. پیشنهاد می‌کنم که تلفظ درست آنرا در سایت‌های مرجع انگلیسی بررسی کنید:

تصویر تلفظ واژه Procedure در دیکشنری cambridge

نکته ۴:

با توجه به تعریفی که در ویکی پدیا آمده است و کاملا درست هم است، «به تعدادی از دستورات متوالی که کار مشخصی را انجام می دهند و یک واحد را تشکیل می‌دهند»، در دنیای برنامه نویسی **سابروتین** یا **پروسیجر** یا **تابع** یا **متد** گوییم.

بنابراین این واژه‌ها به یک معنا در دنیای برنامه نویسی استفاده می‌شوند و فقط بستگی دارد که شما برنامه نویسی چه زبانی هستید. در C# برنامه نویسان از واژه «متد» استفاده می‌کنند و ما در VBA از واژه «پروسیجر» و یا «ماکرو» و یا «سابروتین» استفاده می‌کنیم.

نکته ۵:

باید بدانید که در نسخه‌های اولیه اکسل ما VBA نداشتیم و قابلیت برنامه نویسی به زبان VBA در نسخه 5 اکسل در سال ۱۹۹۳ اضافه شده است. تا قبل از اضافه شدن VBA، اکسل دارای قابلیتی بود به نام **Macro**. یعنی به کاربران این امکان را می‌داد که با نوشتن دستوراتی کارهایی را به شکل پیاپی انجام دهند. اگر چه واژه «ماکرو» مربوط به به آن زمان است، با این حال این واژه از بین نرفته است و در این کتاب هر جایی که ما از واژه ماکرو استفاده کردیم، دقیقا منظورمان همان پروسیجر است.

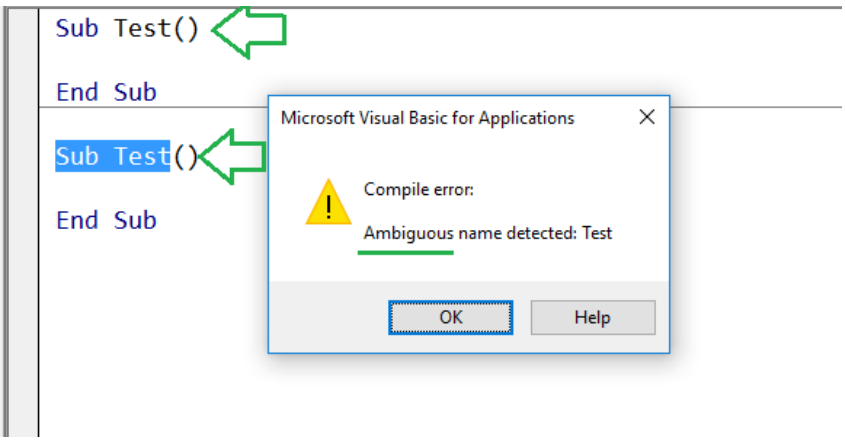
قوانین نامگذاری پروسیجرها

هر پروسیجر باید یک نام داشته باشد و توصیه می‌کنیم که از نامهای بامعنا استفاده کنید که توصیف کننده کار پروسیجر باشد در ضمن آنکه قوانین نامگذاری پروسیجر ها به شرح زیر است:

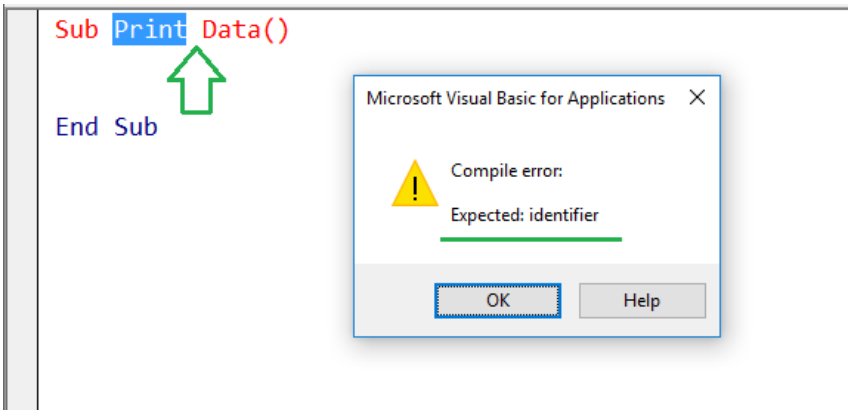
۱- نام پروسیجر نمی‌توانید علامت Space یا Dot داشته باشد مثلا Print Data غلط است و به جای Space می توانید _ بگذارید مانند Print_Data
۲- نمی تواند نام را با علامت‌هایی مانند «_» و یا «؟» و یا یک عدد شروع شود.
۳- نمی توانید از کلمات که برای VBA دارای مفهومی است استفاده کنید مثلا اسم پروسیجر را نمی‌توانید Null بگذارید.
۴- نباید در یک ماژول اسم تکراری داشته باشید.
۵- توصیه می شود که از چهار حرف بیشتر باشد مثلا اسم یک پروسیجر را A20 نگذارید زیرا این نام با سلول A20 می‌تواند تداخل ایجاد می‌کند.

توجه: شاید بدانید که برخی از نرم افزارهای برنامه نویسی به حروف بزرگ و کوچک انگلیسی حساس هستند یعنی واژه Farshid و یا FARSHID و یا fArSHID برای آنها سه چیز جداگانه محسوب می شود، اما VBA اینگونه نیست و به حروف کوچک و بزرگ در نامگذاری های حساس نیست و همه این سه نام یکسان هستند.

تصویر خطای نام پروسیجر تکراری در یک ماژول



تصویر خطای استفاده از Space در نام پروسیجر



نوشتن اولین برنامه – Hello World

در کتاب ها و آموزش های برنامه نویسی اولین برنامه ای که آموزش می دهند آن است که برنامه‌ای نوشته شود تا پیغام Hello World را نمایش دهند و ما هم همین کار را می‌کنیم بنابراین صورت مساله ما می شود:

«برنامه ای بنویسید که پیغام Hello World را به ما نمایش دهد».

برنامه ما این خواهد بود:

```
Sub MyCode()
      MsgBox "Hello World"
End Sub
```

شرح این برنامه:

یک پروسیجر به نام MyCode درست کرده ایم و در داخل آن توسط دستور MsgBox پیغام را به کاربر نمایش می دهیم. از آنجایی که پیغام یک متن است (نه محاسبه و یا عدد) باید آنرا در داخل علامت کوتیشن (دقیق تر بگوییم علامت Double Quote") قرار دهیم.

قطعا در آینده در مورد MsgBox بیشتر صحبت می‌کنیم.

این شد اولین برنامه VBAشما در اکسل. به همین سادگی.

بگذارید که همه کارهای را که تا به حال انجام داده‌ایم را خلاصه کنیم:

۱- وارد اکسل شدید
۲- کلید ALT+F11 را زدید و وارد VBE شدید
۳- از Insert → Module یک ماژول ساختید
۴- در داخل آن ماژول دستوراتی را تایپ کردید (یک پروسیجر ساخته اید که در داخل آن پروسیجر یک دستور است که پیامی را به کاربر نمایش می دهد.)

توجه ۱: به علامت Space ها بین کلمات دقت کنید مثلا دستور MsgBox سرهم است و فاصله ای ندارد و دستور End Sub یک Space دارد.

توجه ۲: لازم نیست که شما حروف را به صورت بزرگ بنویسید مثلا اگر به جای MsgBox شما بنویسید msgbox کاملا صحیح است و نکته آنجاست که اگر شما دستور را درست نوشته باشد، بلافاصله بعد زدن Enter خود VBE این دستور را به صورت تعریف شده اش که همان MsgBox است می نویسد. و از اینجا شما متوجه می شوید که دستور را درست نوشته اید و اگر دستور به حروف بزرگ تبدیل نشد، یعنی آنرا اشتباه تایپ کرده‌اید.

تصویر دستور MsgBox که اشتباه تایپ شده است
و به حروف بزرگ تبدیل نشده است

```
Sub WrongCode()
    mgsbox "Hello World"
End Sub
```

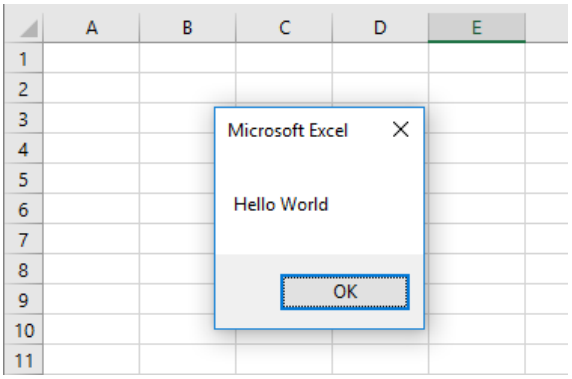
اجرای یک پروسیجر

حالا که ماژول دارید و در آن ماژول یک پروسیجر وجود دارد و در داخل پروسیجر هم یک دستور برای نمایش پیغام تایپ کرده‌ایم، باید پروسیجر رو اجرا کنید و برای اجرای پروسیجر بیش از پنج روش گوناگون در اکسل وجود دارد که در ادامه آنها را خواهیم گفت:

روش اول - استفاده از کلید F5

کافی است که یک جایی در بین سطرهای پروسیجر، Cursor (چشمک زن) را قرار دهید و سپس کلید F5 رو بزنید. بلافاصله پروسیجر شما اجرا می شه و پیام رو خواهید دید.

تصویر نتیجه اجرای پروسیجر MyCode

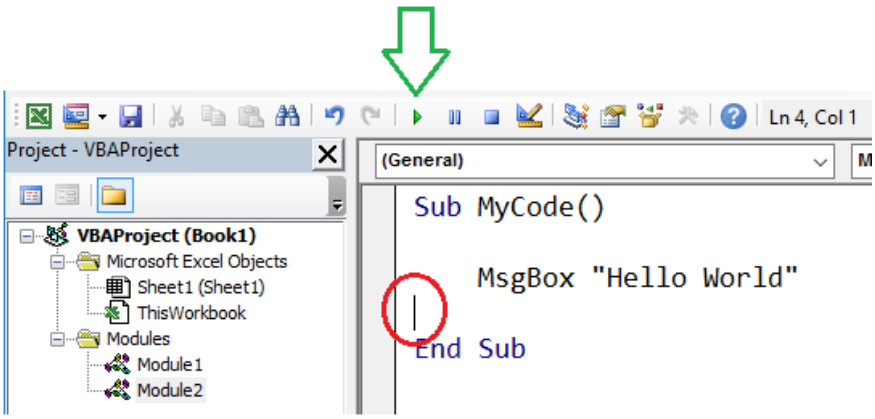


روش دوم – دکمه Run

برای اجرای پروسیجر می‌توان دکمه Run را در Toolbar بالای محیط VBE بزنید.

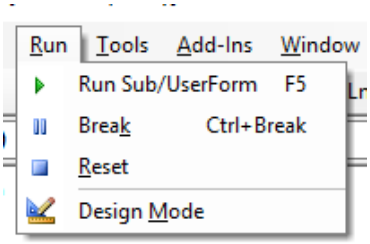
توجه : به محل Cursor (دایره قرمز رنگ در تصویر زیر) دقت کنید در داخل پروسیجر است . در واقع اگر چند پروسیجر در یک ماژول داشته باشید ، آن پروسیجری اجرا می شود که Cursor در آن قرار دارد.

تصویر دکمه Run برای اجرای پروسیجر



روش سوم – منوی Run

می‌توانید برای اجرا از منوی Run گزینه Run Sub/UserForm را انتخاب کنید.

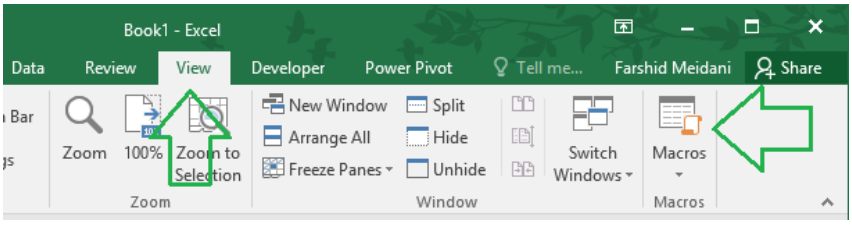


روش چهارم – از داخل Excel

در سه روش قبلی ما از داخل محیط VBA یک پروسیچر را اجرا کردیم و اگر از داخل اکسل بخواهید پروسیجری را اجرا کنید می توانید از مسیر زیر پنجره Macros را بیاورید.

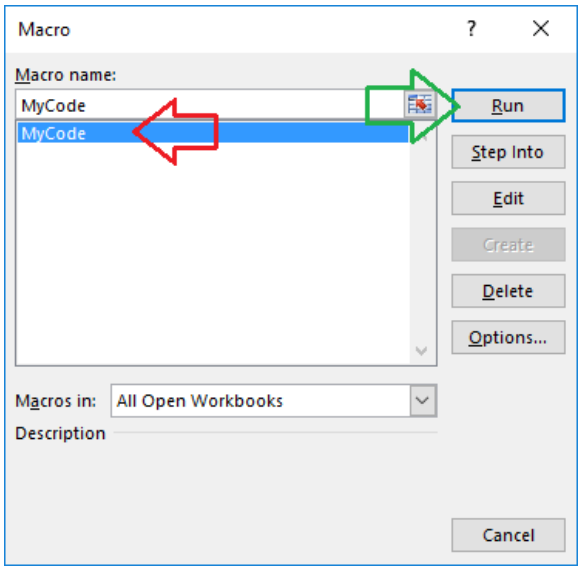
View → Macros

تصویر محل گزینه Macro در View



سپس در پنجره Macro، ماکرو (یا همان پروسیجر) را انتخاب و آنرا Run کنید.

تصویر پنجره Macros

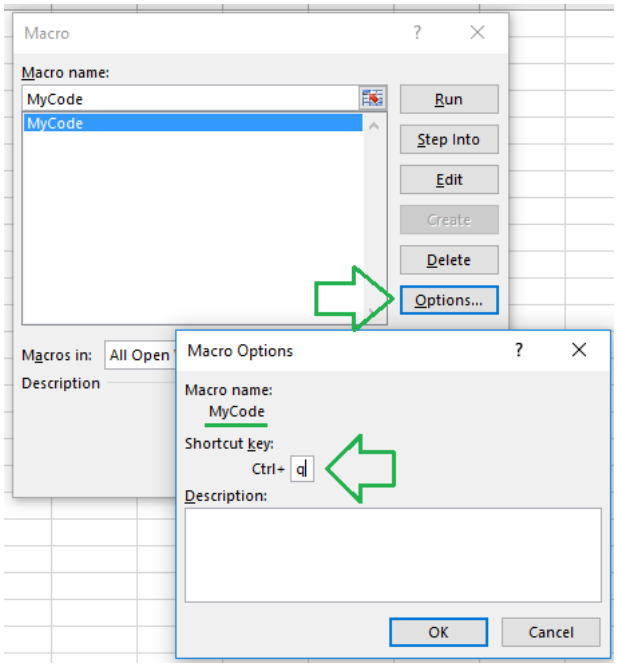


یادآوری: همانطور که گفتیم واژه «ماکرو» و «پروسیجر» در اینجا یک معنا دارند.

نکته : برای نمایش پنجره Macros می‌توانید از کلید ALT+F8 استفاده کنید.

روش چهار و نیم – اجرا با کیبورد

در پنجره که با زدن ALT+F8 آوردید (مرحله قبل) اگر دقت کنید یک دکمه به نام Options را می بنید، وارد آن شوید و یک کلید میانبر دلخواه برای ماکروی خود بگذارید.



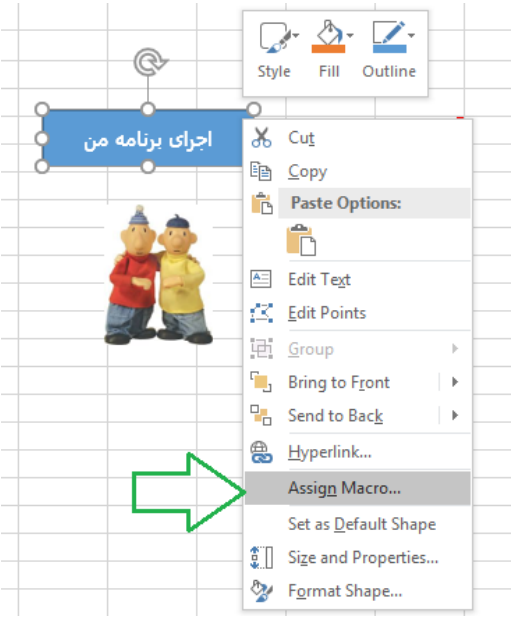
سوال : می دانیم که کلید CTRL +z برای Undo کردن است ، اگر این کلید را به عنوان میانبر به ماکرویی بدهیم، با زدن CTRL + z ماکرو اجرا می شود یا فرمان Undo؟

روش پنجم – با کلیک بر روی یک چیز

ممکن است که یک فایل رو برای یک نفر دیگر بفرستیم و بهتره که کمی روش اجرای ماکرو/پروسیجر ما استانداردتر شود. یعنی یک دکمه ای بگذاریم و اگر کاربر روی آن دکمه کلیک کرد، پروسیجر/ ماکرو ما اجرا بشه.

خوب اینکار بسیار ساده است و کافی است که یک شکلی / یک عکسی / یک آیکونی در شیت بگذاری و روی آن R-Click کنید و از گزینه Assign Macro استفاده کنید. از این پس هر وقت روی آن شکل/ آیکون / عکس کلیک کنید ، ماکرو Run می شود.

تصویر R-Click روی یک شکل و Assign ماکرو



نکته: اگر خواستید که شکل و یا عکسی را که به یک ماکرو Assign کرده‌اید را انتخاب کنید تا مثلا اندازه آنرا تغییر دهید، باید روی آن R-Click کنید.

روش ششم – اجرا از داخل یک پروسیجر دیگر

با دستور Call می توانیم یک پروسیجر را از داخل پروسیجر دیگری اجرا کنیم. به عنوان مثال اگر یک پروسیجر مانند زیر داشته باشیم ، با اجرای آن باعث می شویم که پروسیجر MyCode اجرا شود.

```
Sub SecondCode()
      Call MyCode
End Sub
```

توجه ۱: علامت پرانتز در هنگام Call کردن اجباری نیست.

توجه ۲: نوشتن واژه Call هم اجباری نیست و اگر فقط نام پروسیجر را بنویسید ، اجرا خواهد شد.

```
Sub SecondCode()
      MyCode
End Sub
```

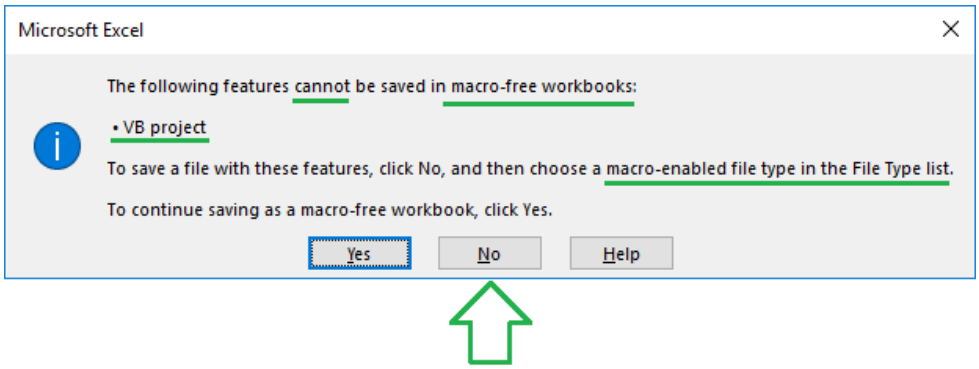
ذخیره فایل

حال وقت آن است که برنامه خود را ذخیره کنید. همانطور که قبلا گفتم شما یک VBAProject دارید که در آن تمامی برنامه‌هایی که نوشته‌اید (ماژول‌ها و پروسیجرها) قرار دارد.

باید بدانید که VBProject یکی از اجرای فایل اکسل است دقیقا مانند شیت‌ها و بدیهی است که اگر بخواهید که VBProject را برای استفاده‌های بعدی ذخیره کنید، باید فایل اکسل خود را ذخیره کنید.

باید بدانید که ذخیره کردن فایلی که پروسیجر/ماکرویی دارد دارای یک نکته است و اگر مطابق معمول برای ذخیره آن فایل دکمه Save را در Excel بزنیم، خواهیم دید که ذخیره انجام نمی‌شود و یک پیغام هشدار نمایش داده می‌شود:

تصویر هشدار حذف VB Project (ماکروها) در فایل های عادی اکسل هنگام ذخیره کردن

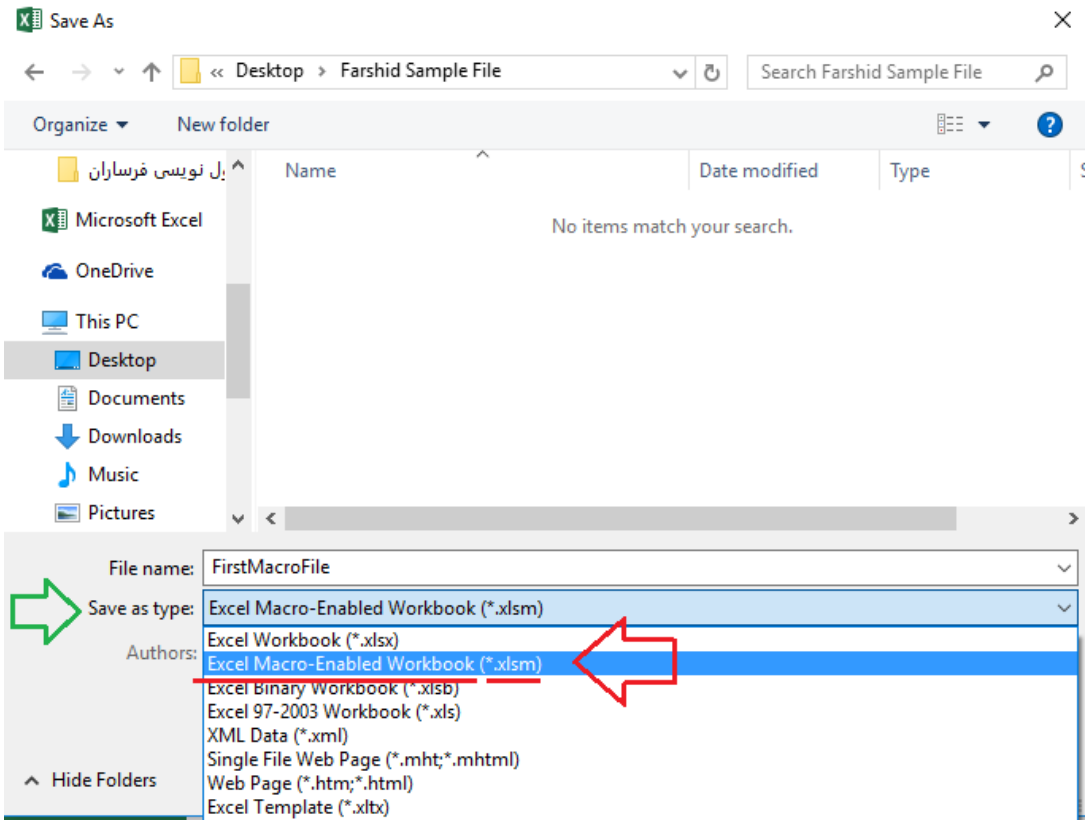


باید این پیغام را جدی بگیریم و معنای این پیغام به شرح زیر است:

۱- نمایش این پیغام یعنی اینکه یک VBProject در فایل شما وجود دارد.
۲- VBProjectی که در فایل شما وجود دارد، قابل ذخیره در فایل‌های عادی اکسل نیست. فایل های عادی اکسل با پسوند xlsx ذخیره می شوند و فایل های xlsx قرار نیست که دارای هیچ VBProjectی باشند. در واقع سازندگان اکسل تصمیم گرفته‌اند که فایل های ماکرو دار و بدون ماکرو رو از هم جدا کنند (در جایی اشاره شده است که اینکار به همان دلایل امنیتی است که قبلا اشاره کردیم) و به همین دلیل یک فرمت ویژه برای فایل‌های ماکرودار با پسوند «xlsm» در نظر گرفته‌اند که به این فرمت Macro Enabled Workbook می گویند و باید حتما فایل خود را با این فرمت ذخیره کنید.
۳- در پیغام گفته است که آیا مایل هستید که فایل را در حالت بدون ماکرو ذخیره کنید و هشدار داده است که اگر گزینه Yes را بزنید، VBProject شما حذف خواهند شد.
۴- همچنین در پیغام گفته است که اگر مایلید که VBProject خود را ذخیره کنید، گزینه No را بزنید و سپس در هنگام ذخیره گزینه Macro Enabled Workbook رو از Save as Type انتخاب کنید.

بدیهی است که ما می‌خواهیم پروسیجر/ماکروهای خود را ذخیره کنیم و به همین دلیل روی گزینه No کلیک می‌کنیم تا فایل را در حالت «ماکرو دار» ذخیره کنیم.

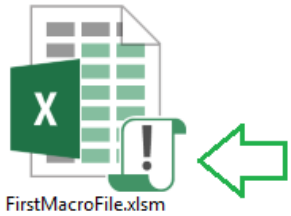
تصویر انتخاب نوع فایل xlsm برای ذخیره ماکرو



نکته ۱: به دلیل تنظیمات ویندوز شما ممکن است در تصویر قبلی واژه «xlsm» را نبینید.

نکته۲: بعد از ذخیره شدن یک فایل در حالت Macro Enabled، اگر به آیکون آن فایل توجه کنید، خواهید دید که دارای یک علامت «!» است به عنوان هشدار.

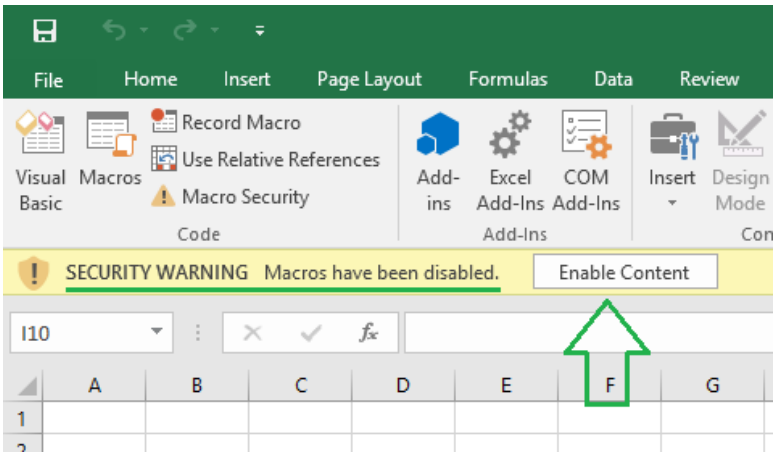
تصویر آیکون فایلی که در حالت MacroEabled ذخیره شده است (Excel 2016)



باز کردن یک فایل دارای ماکرو

همانطور که قبلا گفته شد، بلافاصله بعد از باز کردن یک فایل xlsm ،باید ماکروهای آن فعال شود:

تصویر دکمه Enable Content برای فعال کردن ماکرو



فصل سوم – دستورات پایه زبان VBA

ما در فصل گذشته توانستیم یک برنامه خیلی ساده بنویسیم و آنرا اجرا کنیم. حال در این فصل باید قوائد، قوانین و دستورات پایه زبان VBA را یاد بگیریم . باید بدانید که این قوائد در همه جا یکسان است و فرقی نمی کند که شما در Excel برنامه نویسی می کنید یا در Access. اصلا اگر دوست دارید برای تمرین این قوائد PowerPoint را اجرا کنید و با زدن ALT+F11 وارد محیط ادیتور شوید و برنامه های خود را بنویسید.

من یک نگرانی دارم و آن این است که شما بخواهید از روی این فصل سریع رد شوید و آنرا سرسری بخوانید، قطعا این کار را نکنید و مطمئن شوید که همه چیزها را خوب یاد گرفته اید و تمامی کدهای این فصل را یکبار خودتان تایپ و اجرا کنید.

قوائد و قوانین کلی دستورات

قبل از آنکه بخواهیم دستورات را شروع کنیم بهتر است که چند نکته را با شما در میان بگذاریم تا کار و درک دستورات برای شما ساده تر گردد.

منظور ما از دستور چیست

عنوان این فصل با واژه «دستورات» شروع کردم و بارها در این کتاب از واژه «دستور» استفاده می‌کنم و مایلم که قبل از هر کاری تکلیف این واژه را مشخص کنم.

در واقع من واژه انگلیسی Statement را به «دستور» ترجمه کنم و این ترجمه کمی می‌تواند اشتباه باشد زیرا ترجمه دقیق‌تر واژه Statement می‌تواند «گزاره» و یا «عبارت» باشد. اما این ترجمه ها برای ما نامانوس است و شاید استفاده از ترجمه کمی غلط اما مانوس بهتر باشد.

حال که این بحث باز شد بگذارید تعریف Statement که ما آنرا «دستور» ترجمه کرده ایم را به شما بگویم:

A *Statement* is a command that perform an action.

ترجمه این تعریف اینگونه است :

یک Statement فرمانی است که عملی را انجام می دهد مثل ذخیره کردن یک فایل یا نمایش یک پیغام به کاربر.

حال اگر در کتاب ها و یا سایت مایکروسافت چیزی شبیه تصویر بعدی را مشاهده کردید، شاید بهتر بتوانید آنرا درک کنید.

تصویری از راهنمای دستور IF در سایت مایکروسافت

```
' Multiple-line syntax:
If condition [ Then ]
    [ statements ]
[ ElseIf elseifcondition [ Then ]
    [ elseifstatements ] ]
[ Else
    [ elsestatements ] ]
End If
```

هر کجا که واژه Statement بکار رفته است به این معنا است که شما می توانید هر دستوری مجاز VBA را بکار ببرید. بدون هیچ محدودیتی.

حروف بزرگ و حرف کوچک در دستورات

زبان VBA برایش تفاوتی نمی‌کند که یک دستور را با حروف بزرگ بنویسید و یا کوچک و در هر دو صورت دستور را متوجه می شود و اجرا می کند.

کار خوبی که ادیتور (منظور از ادیتور همان VBE است) انجام می دهد آن است که به صورت خودکار، بعد از زدن Enter و رفتن به سطر بعدی، «کلمات کلیدی» را به حالت استاندارش تبدیل می کنید.

منظور از کلمات کلیدی یعنی واژه‌هایی که VBA آنها را می‌تواند متوجه شود و تفسیر کنید مانند Sub و یا IF .

به عنوان مثال اگر ما دستور "Hi" msGbOX را بنویسیم، بلافاصله بعد از زدن Enter به حالت "Hi" MsgBox در می آید.

این کار (یعنی استاندارد کردن حروف بزرگ و کوچک) به ما کمک می کند تا همان لحظه در هنگام تایپ، غلط های تایپی که انجام داده ایم مطلع شویم و آنها را تصحیح کنیم.

نکته ۱: در سایر زبان های برنامه نویسی مانند #C دستورات از ابتدا باید صحیح نوشته شوند و تصحیح خودکاری نداریم . Visual Basic یک زبان Basic و غیر سخت گیرانه است.

نکته ۲: وقتی که دو عبارت را باهم مقایسه می‌کنیم، (مثلا در دستور IF) حروف کوچک و بزرگ مهم می شود و مقدار Farshid و fArShld باهم یکی نیستند. این نکته را در فصل‌های بعدی خواهیم دید.

رنگ دستورات

هنگامی که دستورات کلیدی VBA را تایپ می‌کنیم (و با زدن Enter به سطر بعدی می‌رویم)، رنگ آنها تغییر می کند. مثلا در شکل زیر رنگ دستورات Sub , If , Then , End Sub همگی آبی شده اند و تغییر رنگ یعنی آنها، یعنی از کلمات کلیدی VBA هستند .

همچنین رنگ به برنامه نویس کمک می کند که متوجه شود که چه چیزهایی را اشتباه نوشته است، در ضمن آنکه به خوانایی بالای برنامه نیز کمک می کند.

تصویر تغییر رنگ دستورات VBA به آبی

```
(General)                    Chapter2_Ex1
Sub Chapter2_Ex1()

    MsgBox "Hi"
    ' This is a comment
    If x > 10 Then MsgBox "Bye"

End Sub
```

نکته: یک ظرافتی در رنگ تصویر قبلی شاید متوجه شده باشد، رنگ MsgBox تغییر نکرده است و این یعنی MsgBox یک دستوری کلیدی VBA نیست بلکه یک تابع (Function) است . ما با Functionها به صورت کامل در آینده آشنا خواهیم شد.

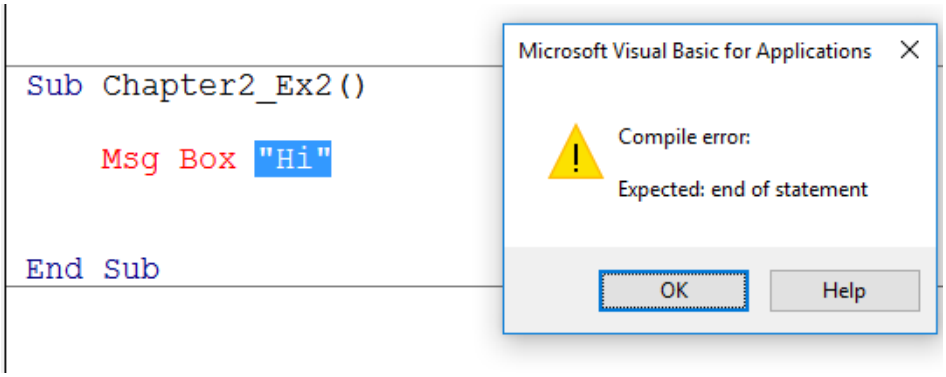
گذاشتن Space بین دستورات

ما در نگارش فارسی هم اینکار را می‌کنیم و بین واژه ها Space می گذاریم و این قاعده در مورد دستورات VBA هم صادق است . مثلا دستور End Sub باید بین کلمه Sub و End یک فاصله باشد و یا دستور MsgBox هیچ فاصله ای بین دو واژه Msg و Box نیست.

حالا فرض کنید که من Msg Box بنویسیم یعنی یک space که نباید بگذارم را، بگذارم. در واقع اصلا لازم نیست که برنامه را Run کنید تا نتیجه را ببنید بلکه بلافاصله خود ادیتور سعی می کند که بعد از زدن Enter یک بررسی های اولیه را انجام دهد و خطاها را گوشتزد کند و در نتیجه به شما خطای زیر را خواهد داد که معنی آن این است که «دستوری را که نوشته‌اید را یک بررسی اولیه کردم و نتوانستم بفهمم که آن دستور چیست و کجا به پایان می رسد.»

نکته دیگر آن است که خط «قرمز» رنگ می شود و به صورت کلی یعنی خطای تایپی دارید.

تصویر خطای تایپی بعد از زدن Enter به دلیل Space اضافه



نکته: زدن Space بیش از یک عدد بین دستوراتی که باید جدا از هم نوشته شوند اشتباه محسوب نمی‌شود و توسط Editor بعد از زدن Enter آن Space های اضافه حذف می شوند.

استفاده از Tab برای تو رفتگی دستورات

این موضوع که الان می خواهیم بررسی کنیم یک اجبار و قانون نیست، بلکه یک عادت بسیار خوب و حرفه ای برنامه نویسی است که ما در نوشتن برنامه سعی کنیم که خوانایی آنرا بالاتر ببریم و اینکار با زدن کلید Tab در ابتدای دستورات مانند تصویر زیر انجام می شود.

تصویر استفاده از Tab برای خواناتر کردن برنامه

```
Sub Chapter2_Ex3()
    MsgBox "Hi"
    If x >= 10 Then
        MsgBox "bye"
        MsgBox "******"
    End If

End Sub
```

با یک نگاه به دستور بالا متوجه می شود که دستورات MsgBox “Bye” و MsgBox “******” زیرمجموعه‌ی دستور IF هستند .

نکته : زدن کلید Shift + Tab یا Backspace باعث حذف Tab می شود.

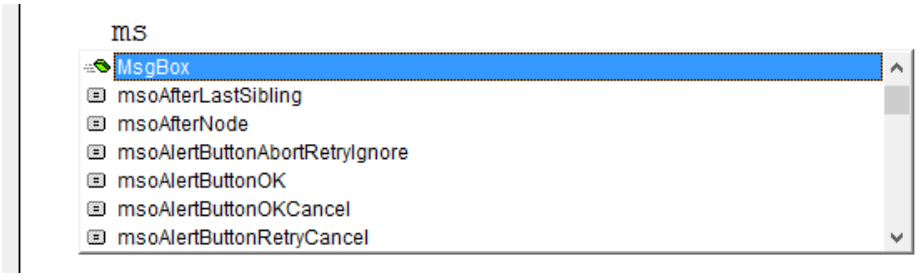
استفاده از CTRL+Space

محیط VBE همانطور که گفتیم یک ادیتور است و به ما کمک می کند که تایپ دستورات را راحت‌تر انجام دهیم.

یکی از امکانات این محیط آن است که می توانیم از حالت تکمیل/تایپ خودکار استفاده کنیم. به عنوان مثال شما بنویسید ms و در ادامه کلید CTRL+Space را بزنید . خواهید دید که یک لیست از همه چیزهایی که با ms شروع می شود برای شما نمایش داده می شود.

نکته: به این لیست اصطلاحا Member List می‌گویند

تصویر تکمیل خودکار با زدن CTRL+Space



حال کافی است که با کلیدهای بالا/پایین دستور مورد نظر را Highlight کنید و کلید Tab را بزنید تا آن دستور نوشته شود.

سوال : بنویسید appl و سپس CTRL+Space را بزنید. چه واژه ای نوشته می شود و چرا آن لیست برای شما نمایش داده نشد؟

توضیحات یا Comment

گفته بودیم که تمامی دستورات یک پروسیجر بعد از Run کردن ، سطر به سطر اجرا می شوند. حال اگر بخواهید که یک سطر را اجرا نکنید کافی است که آنرا به Comment تبدیل کنید.

سطرهایی که Comment هستند توسط برنامه تفسیر و اجرا نمی شوند و برای Comment کردن کافی است که در ابتدای آن سطر واژه rem را بنویسید. (احتمالا مخفف واژه remark باشد).

نکته ۱: سطرهای comment سبز رنگ می شوند.

نکته ۲: به جای واژه Rem می توانید علامت از علامت آپستروف مانند تصویر استفاده کنید.

نکته ۳: هیچ اشکالی ندارد که Comment ها در انتهای یک سطر واقع شوند.

تصویر Comment ها

```
Sub Chapter2_Ex4()

    MsgBox "Hi"
    Rem  MsgBox "bye"
    ' MsgBox "bye"
    ' Author : Farshid
    ' Version 1.001
    MsgBox "Hi VBA"     ' just for fun :)

End Sub
```

حال که متوجه شدید comment چیست، باید سعی کنم که به شما بگویم که چه مواقعی از آن برنامه نویسان حرفه ای استفاده می کنند.

الف) برای توضیحات برنامه

در نظر داشته باشید که یک برنامه حرفه ای ممکن است صدها خط کد داشته باشد و اگر خود برنامه نویس و یا افراد دیگری بخواهند واقعا از آن سر در بیاورند لازم است که یک توضیح خیلی مختصری در مورد دستورات داده شود و اینکار در متن برنامه با comment ها انجام می شود.

ب) نوشتن نام برنامه نویس و توضیحات برنامه

قاعدتا خیلی از مواقع نسخه برنامه/ کلیدهای میانبر / نام برنامه نویس را در Comment ها می‌نویسم.

ج) تکنیک مهرتاش

مهرتاش از آن به عنوان یک استراتژی و تکنیک استفاده می کرد. (مهرتاش دوست من است و برنامه نویس حرفه ای است.)

وقتی که می خواهد یک خط برنامه را تغییری دهد، آن خط را ویرایش نمی کند ، از آن خط یک کپی می گیرد و سپس کپی را ویرایش می کند و نسخه اصلی خط را Comment می‌کند.

سطر Comment شده را تا تکمیل برنامه نگهداری می کند و می‌گوید Code Cleanup آخر سر.(یعنی پاک کردن Commentها در آخرین مرحله تولید برنامه قرار می‌گیرد)

زیرا در بسیاری از مواقع می‌خواهیم به حالت قبلی برگردیم و اگر سطر قبلی را داشته باشیم اینکار خیلی سریع اتفاق می افتد.

نکته ۴: در VBE ما راهی نداریم که به یکباره چندین سطر را به Comment تبدیل کنیم و این را گفتم تا بدانید که VBE یک ادیتور خیلی پیشرفته و مدرن نیست و خیلی محدود و ساده است و شاید همین برای کاربران تازه کار مزیت باشد و از طرف دیگری یک ادیتور بسیار مدرن و جدید و پیچیده مانند Visual Studio که برنامه های Net. را در آن می نویسند، هر آنچه را که فکر کنید دارد. حتی برایش کتاب هم نوشته اند!!

یک سطری کردن

بدیهی است که باید هر دستور را در یک سطر مجزا بنویسیم و اگر اینکار را نکنید ، برنامه خطا می‌دهد و اجرا نمی شود.

حال ممکن است برای زیبایی یا برای خلوت شدن بخواهید که چندین دستور را که اساسا باید در چند سطر مختلف نوشته شوند را در یک سطر بنویسید این کار با علامت «:» مقدور است.

```
Sub Chapter2_Ex5()
    MsgBox "Hi": MsgBox "good morning": MsgBox "bye"
End Sub
```

چند سطری کردن

فرض کنید که دستوری دارید که آن دستور بسیار طولانی شده است و در عرض مانتیور روی صفحه نمایش داده نمی شود و خواندن آن برای شما کمی دشوار است در این حالت می توانید آن دستور را که اساسا باید در یک سطر نوشته شود را در چند سطر متفاوت بنویسید .

برای اینکار ازSpace و سپس یک علامت «_» (آندرلاین) استفاده می‌کنیم در تصویر زیر یک کد طولانی را مشاهده می کنید :

```
Application.ActiveSheet.Range("A1").Offset(2, 2).Resize(2, 2).Interior.Color = vbRed
```

که بعد از شکستن می توانیم آنرا به شکل زیر چند سطری کنیم.

```
     Application.ActiveSheet _
        .Range("A1").Offset(2, 2) _
            .Resize(2, 2).Interior.Color = vbRed
```

یا هر جور دیگری که مایلیم آنرا تقسیم نماییم.

```
     Application _
        .ActiveSheet _
            .Range("A1").Offset(2, 2) _
                 .Resize(2, 2) _
                    .Interior.Color = _
                vbRed
End Sub
```

نکته ۱: دقت کنید که «بعد از» هر دستور _ +Sapce را گذاشته ایم و یک دستور مانند ActiveSheet را نمی‌توانید بشکنید.

نکته ۲: دقت داشته باشد که باید برای شکاندن یک سطر ابتدا یک Space بگذارید و سپس یک آندر لاین.

تصویر نمایش محل Space ها قبل از علامت آندرلاین

```
Application.ActiveSheet _
    .Range("A1").Offset(2, 2) _
        .Resize(2, 2).Interior.Color = vbRed
```

یادآوری: برای خوانا شدن از Tab در ابتدای هر سطر استفاده کرده ایم.

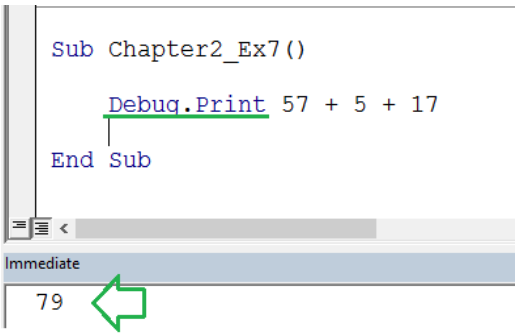
معرفی دستور Debug.Print

تا به حال ما از دستور MsgBox در چند مثال استفاده کرده‌ایم و تست را انجام می دادیم. اما برای کارهای آموزش این کتاب و در آینده برای تست هایی که شما انجام می دهید این دستور یک مشکل کوچک دارد، آنهم این است که شما را وارد اکسل می کند و باید کلید Ok را برای پایان آن بزنید و اگر مثلا بخواهید ۱۰ تا چیز را نمایش دهید، باید ۱۰ بار پیاپی Ok را بزنید که واقعا کار کلافه کننده‌ای می شود.

از این به بعد ما خروجی هایمان را در قسمت Immediate با دستور Debug.Print نمایش می‌دهیم. مثلا می خواهیم که حاصل فرمول ۵۷ + ۵ + ۱۷ را ببینیم ، کافی است که در یک پروسیجر دستور زیر را اجرا کنیم.

```
Debug.Print 57 + 5 + 17
```

تصویر اجرای Debug.Print



نکته ۱: بین دو واژه Debug و Print یک نقطه باید باشد.

نکته ۲: اگر قسمت Immediate را نمی بینید برای نمایش آن کافی است که CTRL+G را بزنید و یا از منوی view آنرا فعال کنید.

نکته ۳: می خواهم بگویم که واژه Debug به معنی خطایابی یا خطازدایی است و می‌بتوانم دستور Debug.Print را اینطور تفسیر/معنی کنم که «به قصد خطایابی یا فهمیدن برنامه برای من مقداری را در قسمت Immediate چاپ کن».

تمرین: پروسیجر زیر را تایپ و اجرا کنید و خروجی آنرا ببینید. (این اولین برنامه‌ای بود که من با کمودور ۲۰ اجرا کرده ام.)

```
Sub Chapter2_Ex8()

    For x = 1 To 255
        Debug.Print Chr(x)
    Next x

End Sub
```

نکته ۳: دستور Debug.Print هر خروجی را در یک سطر جداگانه چاپ می‌کند و اگر می‌خواهید که خروجی این دستور پیاپی و پشت سر هم چاپ شوند کافیست که در انتهای آن دستور علامت «;» (سمیکالن یا نقطه ویرگول) را مانند مثال زیر تایپ کنید:

```
Debug.Print Chr(x);
```

نکته ۴: به جای علامت سمیکالن می‌توانید از علامت کاما نیز استفاده کنید که این علامت باعث گذاشتن ۱۳ فاصله (Space)بین چاپ های پیاپی در یک سطر خواهد شد.

```
Debug.Print Chr(x),
```

عملگرها و تقدم عملیات ها

من این بحث را مایل نیستم که باز کنم زیرا باید شما در اکسل با آن آشنا باشید و به صورت اجمالی عملگرهای اصلی عبارتند از :

عملگر	مثال	نتیجه مثال	شرح
*	x = 2 * 10	20	علامت ضرب
/	x =10 / 3	3.333333	علامت تقسیم
+	x = 2 + 10	12	علامت جمع برای دو عدد
+	x = "a" + "b"	Ab	اگر دو مقدار متن باشند، آنها به یکدیگر چسبانده می شوند.
&	x = "a" & "b"	ab	چسباندن دو مقدار به یکدیگر
-	x = 10 – 2	8	علامت تفریق
^	x = 2 ^ 10	1024	علامت توان و مقدار
Mod	x = 10 mod 3	1	محاسبه باقی مانده یک تقسیم (با خارج قسمت صحیح)
\	x = 10.2 \ 2.5	5	ابتدا اعشار دو عدد را حذف می کند و سپس تقسیم را انجام می دهد و نتیجه تقسیم را بدون اعشار نمایش می دهد.
=	x = 10		مقدار 10 را به متغیر x اختصاص می‌دهد.

نکته ۱: عملگرهای <> , <= , < , >= , > برای مقایسه ها بکار می روند و جواب آنها یکی از دو مقدار True یا False می شود.

نکته ۲: عملگرهای And, OR , Not نیز دقیقا مشابه کاری که در اکسل انجام می دهند را دارند و برای خلاصه شدن این فصل از توضیح و مثالهای آنها خودداری می‌کنیم.

نکته ۳: در جدول بالا به علامت «=» دقت نمایید. به این علامت Assignment Operator می‌گویند.

متغیر (Variable) چیست

هر وقت که می خواهیم این بحث را در کلاس درس بدهم احساس می کنم که همه را خسته خواهم کرد و اگر می توانستم واقعا آنرا حذف می کردم. اما نمی شود چون یک بحث حیاتی در دنیای برنامه نویسی است.

بگذارید با یک مثال شروع کنم مثلا دستورات زیر را در نظر بگیرید:

```
x = 10
Debug.Print x
x = 15
Debug.Print x
```

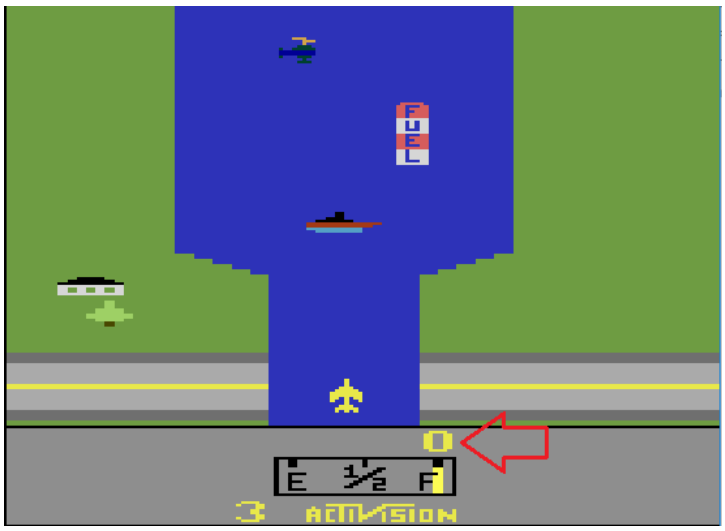
مقدار x یک متغیر است ، یعنی اول مقدار ۱۰ را داشته است و بعد مقدار ۱۵ را و هر بار بگوییم که x را چاپ کن، مقداری که در آن لحظه را دارد، چاپ خواهد شد.

این موضوع را کمی علمی تر می‌کنم. همه اتفاق ها در دنیای کامپیوتر در حافظه RAM می افتد و وقتی که شما می نویسید x = 10 ، در RAM دستگاه یک خانه ای (یک جایی /یک فضایی / یک ظرفی / یک جعبه‌ای) به نام x ساخته می شود و «سپس» (به واژه سپس دقت کنید) مقدار ۱۰ در آن خانه قرار داده می شود.

بدیهی است که آن خانه می توانیم یک چیز دیگر را بگذاریم به همین دلیل است که به x می گوییم متغیر یعنی در هر زمان می توانیم در x یک چیز جدید قرار دهیم و مقدار x را تغییر دهیم و البته در هر لحظه می توانیم از آنچه در خانه x است باخبر شویم (اصطلاحا بخوانیم) و از آن استفاده کنیم.

یک مثال خوب از متغیرها و کاربرد آنها «امتیازی» است که در یک بازی کامپیوتری می‌گیرید. در ابتدا امتیاز شما صفر است و هنگامی که یک هلیکوپتر دشمن را می‌زنید به امتیاز شما ۶۰ واحد اضافه می‌شود و اگر یک تانک را بزنید، به امتیازهای قبلی شما ۳۰ واحد دیگر اضافه خواهد شد. در ضمن همواره مقدار آخرین امتیاز شما در صفحه نمایش داده می‌شود:

تصویر، مقدار متغیری که امتیاز شما را نگه می‌دارد در ابتدای بازی 0 است



مقدار متغیر امتیاز بعد از اینکه اولین هلیکوپتر دشمت را زدید

تصویر مقدار متغیر بعد از اینکه ۲ مرحله را طی کردم

نکته ۱ : نام هایی متغیرها می تواند چیزهایی مانند x و MyCity و Your_City و ... باشد که البته در بخش های آینده در مورد نامگذاری صحبت می‌کنیم.

نکته ۲: متغیرها می توانند هر چیزی را در داخل خودشان نگه دارند مثلا یک عدد ، عدد اعشاری، متن و حتی یک چارت

نکته ۳: برای مقدار دهی یک متغیر از علامت = استفاده می‌کنیم مثلا x = 10

نامگذاری متغیر ها

قوانین نامگذاری متغیرها مانند پروسیجر ها است و مثلا نمی تواند Space داشته باشد ویا با یک عدد شروع شود.

واضح است که بهتر است نامی با معنا برای متغیرهایمان بگذاریم تا اگر تعداد متغیرهای ما در یک پروسیجر زیاد شد بدانیم که هر متغیری برای چیست.

اما جالب است بدانید که نامگذاری متغیرها برای خودش اصول و قواعدی دارد و در شرکت های برنامه نویسی شما باید از آن قواعد پیروی کنید مثلا با حروف کوچک شروع شوند و ... .

به عنوان مثال فرض کنید که دستمزد فردی ساعت ۱۰۰۰۰ تومان است و او ۸ ساعت کار کرده است و می خواهیم که محاسبه کنیم کلا چقدر باید به او اجرت بدهیم. می توانیم اینگونه متغیرها را نامگذاری کنیم:

```
payRate = 10000
hoursWorked = 8
totalPay = hoursWorked * payRate
```

اگر با واژه های انگلیسی مشکل دارید، پیشنهاد می کنم که اینطوری بنویسید:

```
dastmozdSati = 10000
satKari = 8
dastmozdKoli = dastmozdSati * satKari
```

ما در نامگذاری متغیرها از روشی به نام camelCase استفاده کرده ایم (کوهان شتری). یعنی اولین حرف کلمات وسط، به صورت بزرگ نوشته می شود. یعنی متغیر dastmozdSati از دو واژه تشکیل شده است و واژه dastmozd با حروف کوچک و سپس واژه Sati با حروف بزرگ نوشته می‌شود.

البته شما می توانید از _ نیز استفاده کنید مثلا به جای آنکه بنویسید myVariable بنویسید my_Variable و در واقع اینها همگی قوائدی هستند که خود شما باید آنها را بکار بگیرید و اجباری نیستند.

یک روش دیگری هم است که معمولا ابتدای نام یک متغیر را با مخفف دیتاتایپ آن شروع می کنند مثلا اگر یک متغیری از نوع integer است و نام آنرا می خواهد Counter بگذارند، سه حرف اول int را از integer به Counter می چسبانند و می شود intCounter و قطعا dblNumber یعنی از نوع double .

قطعا شما هر چقدر برنامه های بزرگتر و پیچیده تری بنویسید آنوقت خواهید دید که این استاندارد سازی چقدر کار شما را ساده می کند.

آشنایی با Data Type

گفتیم که متغیرها خانه‌هایی هستند در حافظه کامپیوتر شما که نامی دارند و حالا باید بگویم که اندازه این خانه که یک متغیر اشغال می‌کند بسیار مهم است. زیرا حافظه کامپیوتر محدود است و باید ما به اندازه ای که فضا نیاز داریم از حافظه را بگیریم و نه بیشتر. اگر که از حافظه بهینه استفاده نکنیم برنامه ما ممکن است کند شود!

در واقع شعار ما این است ، اشغال حافظه کمتر ، برنامه سریعتر .

پس ما در نظر داریم که یک متغیر تعریف کنیم و البته بگوییم که چقدر از حافظه را باید اشغال کند و برای خودش بردارد

شاید بپرسید که تا الان و در مثال های قبلی که اصلا از این حرف ها نداشتیم و ما یک متغیر تعریف کردیم و همه چی اجرا شد. من در پاسخ شما می گویم که VBA یک زبان مهربان است و اگر نوع متغیر را تعریف نکنید، خودش کارها را مدیریت می کند اما (به این اما دقت کنید) همه بحث ما آن است که حافظه را بهینه استفاده کنیم و در نتیجه برنامه ما چندین برابر سریعتر اجرا خواهد شد.

در دنیای کامپیوتر همه چیز با واژه بیت اندازه گیری می‌شود و هر ۸ بیت مانند آجرهای یک ساختمان که کنار هم قرار بگیرند، کوچکترین خانه واحد حافظه را که ما می‌توانیم استفاده کنیم را خواهند ساخت که به آن «بایت» می‌گویند. حال باید مشخص کنیم که هر متغیر قرار است چند «بایت» از حافظه را اشغال کند.

جدول Data Type ها و مقدار حافظه مورد نیاز آنها

Data Type or Subtype	مقدار حافظه مورد نیاز بر حسب بایت	محدوده
Integer	2 bytes	–32,768 to 32,767
Long Integer	4 bytes	–2,147,483,648 to 2,147,486,647
Single	4 bytes	–3402823E38 to –1.401298E–45 OR 1.401298E–45 to 3.402823E38
Double	8 bytes	–1.79769313486232E308 to –4.94065645841247E–324 OR 1.79769313486232E308 to 4.94065645841247E–324
Currency	8 bytes	–922,337,203,477.5808 to 922,337,203,685,477.5807
Date	8 bytes	January 1, 100 to December 31, 9999
Fixed String	String's length	1 to 65,400 characters
Variable String	10 bytes plus the number of characters	0 to 2 billion characters
Object	4 bytes	Any Access object, ActiveX component or Class object
Boolean	2 bytes	–1 or 0

Variant	16 bytes	Same as Double
Decimal	14 bytes	−79,228,162,514,264,337,593,543,950,335 to 79,228,162,514,264,337,593,543,950,335 −7.2998162514264337593543950335 to 7.9228162514264337593543950335
Byte	1 byte	0 to 255

بله حق با شماست این جدول رو کی سر در می آره ؟ البته لازم نیست که اینها رو به خاطر بسپاریم و من یک توضیحی در مورد آن می دهم.

مثلا Integer را ببینید، یک دیتا تایپ است برای اعداد بین حدودا ۳۲ هزار تا ۳۲- هزار و اگر لازم داشته باشیم که یک متغیر تعریف کنیم که یک عدد باشد و آن عدد هم در محدوده ۳۲ هزار منفی تا ۳۲ هزار مثبت، از از این دیتاتایپ استفاده خواهیم کرد . در ضمن اینکه جدول نشان می دهد که فضایی که Integer در RAM اشغال می شود ۲ بایت است.

از آنجایی که ما قرار نیست که برنامه‌ای‌های تجاری و مهمی را در حد ملی و یا جهانی بنویسیم، با کمی ساده‌سازی، جدول دیتاتایپ‌ها را برای شما به شرح زیر خلاصه می‌کنم:

- اگر عدد داشتید و آن عدد اعشار نداشت از Long استفاده کنید.
(لازم نیست که از Integer یا Byte استفاده کنید و نگران سرعت هم نباشید.)
- اگر عدد اعشاری داشتید از Double استفاده کنید.
(لازم نیست که از Single استفاده کنید وDecimal هم یک چیز خاص است که بازهم لازم ندارید که با جزئیات آن آشنا شوید.)
- اگر متن بود از String.

نکته مهم: اگر شما دیتاتایپ متغیری را مشخص نکنید خود نرم افزار آنرا از نوع Variant می گیرد و یعنی هر مقداری را می توانید در آن متغیر قرار دهید اما همانطور که از جدول می بینید اینکار اصلا خوب نیست چون مقدار حافظه ای که اشغال می شود ۱۶ بایت است و برنامه شما بسیار کند خواهد شد.

تعریف یک متغیر

تا به حال دیدیم که یک متغیر داریم و متغیرها هم می توانند دیتاتایپ خاصی داشته باشند اما نگفتیم که چگونه می توانیم اینکار را انجام دهیم یعنی یک متغیر را تعریف کنیم و دیتا تایپ آنرا مشخص کنیم. به این عمل اصطلاحا Declare (اعلام / تعریف) کردن متغیر گفته می‌شود.

دستور Declare (تعریف متغیر) در VBA کلمه Dim می باشد که مخفف Dimension است.

با دستور زیر یک متغیر از نوع Long تعریف می‌کنیم:

```
Dim x As Long
```

و یا دستور زیر دو متغیر از نوع double

```
Dim myVar1 As Double , myVar2 As Double
```

و اگر دستور زیر را بنویسید یک متغیر از نوع Variant تعریف کرده اید و می توانید در داخل آن هر چیزی را قرار دهید:

```
Dim w As Variant
```

نکته ۱: اگر در دستور قبلی Variant را ننویسم، به صورت خودکار متغیر از نوع Variant خواهد شد.

```
Dim w
```

مزایای Declare (تعریف) کردن یک متغیر

چندین بار تاکید شد که هدف اصلی ما سرعت بخشیدن به اجرای برنامه است و اگر دیتاتایپ ها را تعریف نکنیم به صورت پیش فرض از نوع Variant می شود که بسیار کند است.

اما نکته دیگری را که می خواهم به آن اشاره و تاکید کنم آن است که تعریف متغیر برای جلوگیری از خطاهای تایپی در هنگام نوشتن برنامه است.

زیرا با نوشتن اولین حروف از نام متغیر و سپس زدن کلید CTRL+Space می توانیم از تکمیل خودکار استفاده کنیم:

تصویر تکمیل نام یک متغیر با زدن CTRL+Space

```
Sub Chapter2_Ex9()

Dim lngLastYearBudget As Long

Debug.Print l

End Sub
```

LenB
lngLastYearBudget
Load
LoadPicture
LoadPictureConstants

در ضمن آنکه هر جایی که از متغیر استفاده شود (سپس به سطر بعدی برویم)، شکل نوشتاری آن از نظر حروف کوچک و بزرگ مشابه شکل اصلی (که در Declare کردن برای آن تعیین می‌کنیم) می شود و اگر حالت حروف و کوچک تغییر نکرد یعنی اشتباه تایپ کرده ایم.

تصویر زیر نشان می دهد که در تایپ نام متغیر در جلوی دستور Debug.Print حرف d به اشتباه تایپ نشده است و به همین دلیل حروف بزرگ و کوچک شبیه شکل اصلی متغیر نشده است.

تصویر غلط تایپی در نام متغیر که از روی حروف کوچک و بزرگ
بلافاصله قابل تشخیص است

```
Dim lngLastYearBudget As Long

Debug.Print lnglastyearbuget
```

تعریف متغیر و نوع دیتا تایپ آن مزیت دیگرش در آن است که پیشاپیش جلوی بسیاری از اشتباهات منطقی ما را خواهد گرفت.

به عنوان یک مثال اگر متغیری از نوع عددی و بخواهیم یک متن را در آن ذخیره کنیم، بلافاصله برنامه ما خطا خواهد داد و به راحتی متوجه آن خواهیم شد. حال اگر متغیر را از نوع عددی تعریف نکرده باشیم، ممکن است برنامه کارش را ادامه دهد و در مراحل بعدی دچار خطایی شویم که در این حالت یافتن منشا این خطا برای ما دشوار خواهد بود.

اجباری کردن تعریف متغیرها

همانطور که دیدیم تعریف (Declare) کردن متغیرها کاری است حرفه‌ای و البته شاید بخواهید خودتان را مجبور کنید که همه متغیرها را تعریف کنید تا نتوانید از متغیری که تعریف نشده است، استفاده کنید.

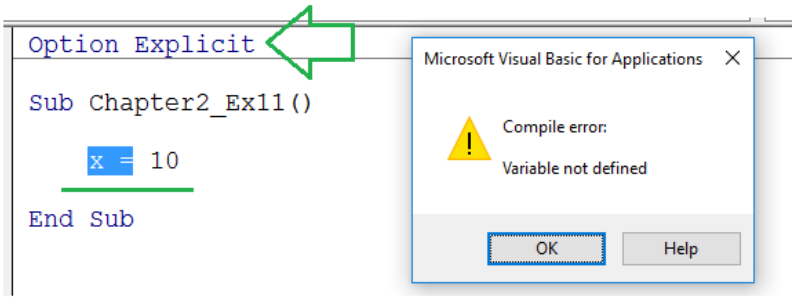
برای اینکار در ابتدای ماژول (قبل از شروع پروسیجر ها) دستور زیر را بکار می برند:

```
Option Explicit
```

از این به بعد در آن ماژول باید همه متغیرها را تعریف کنید و در غیر اینصورت خطا خواهید داشت.

نکته: حتما باید دستور را قبل از اولین پروسیجر، در ابتدای ماژول تایپ نمایید.

تصویر خطای استفاده از متغیر بدون تعریف آن



نکته ۱: در ضمن آنکه می توانید از مسیر Tools → Options → Editor(tab) زیر گزینه Require Variable Declaration را فعال کنیم تا در ابتدای همه ماژول ها دستور Options Explicit به صورت خودکار درج شود.

نکته ۲: توصیه می‌شود که این تنظیم را همواره فعال کنیم تا ملزم باشیم هر متغیری را حتما قبل از استفاده، تعریف کنیم.

نکته ۳: من در فایل‌های مثال این کتاب برای آنکه برخی از مثال‌ها ساده و خلوت‌تر به نظر آیند، از اجباری کردن تعریف متغیرها و در مواردی از تعریف متغیر صرفنظر کرده‌ام. اعتراف می‌کنم که اینکار اشتباه است و به شما توصیه می‌کنم که با دستور Option Explicit ، تعریف متغیرها را اجباری کنید.

تخصیص یک مقدار به یک متغیر

عنوان این موضوع در انگلیسی Assignment Statement است و معنی Assign یعنی اختصاص دادن/ تخصیص دادن. مثلا در فارسی می گوییم که ۱۰ میلیارد تومان به پروژه آبرسانی روستاها تخصیص داده شده است.

عملگر Assignment همان علامت «=» است که در جدول عملگرها به شما گفتیم که به آن Assignment Oprator می‌گویند.

دستور x = 10 را در نظر بگیرید . در این دستور ما مقدار ۱۰ را به x تخصیص می‌دهیم و می توانیم با این روش متغیر x را مقدار دهی کنیم.

نکته مهم آن است که ما در این تخصیص دادن یا مقدار دهی، نمی گوییم x برابر است با ۱۰ . بلکه میگوییم که مقدار ۱۰ را در داخل x قرار می دهیم.

متوجه نکته شدید؟ در واقع ما علامت = را طبق معمول از سمت چپ به راست نمی‌خواهیم بلکه ابتدا سمت راست مساوی را می خوانیم و «سپس» آن مقدار را به x تخصیص می دهیم.

یعنی VBA هر جا علامت «=» را مشاهده کرد، ابتدا سمت راست مساوی را محاسبه می نماید و سپس هر آنچه نتیجه شد را به متغیر سمت چپ علامت مساوی، اختصاص خواهد داد. این موضوع در درس های آینده کاملا مهم است!

نکته : در فصل پنجم از تکنیک assingment بسیار استفاده خواهیم کرد و در آنجا به شما Incrementatl Assignment را آموزش خواهیم داد.

میدان دید متغیرها

عنوان این موضوع در انگلیسی Scoping Variables است و بگذارید با یک مثال ساده بحث را آغاز کنیم.

یک کودک ۲ ساله را در نظر بگیرد، آیا قبول دارید که فقط می‌تواند در داخل خانه بازی کند و می‌توانیم بگوییم که محدوده بازی او فقط در خانه است.

البته وقتی که این کودک ۵ یا ۶ ساله شود قطعا می‌تواند در حیاط خانه هم بازی کند و میدان بازی او بزرگتر خواهد شد و وقتی که این کودک به سن نوجوانی برسد، حتما پدر و مادرش به او اجازه می‌دهند که در کوچه هم بازی کند.

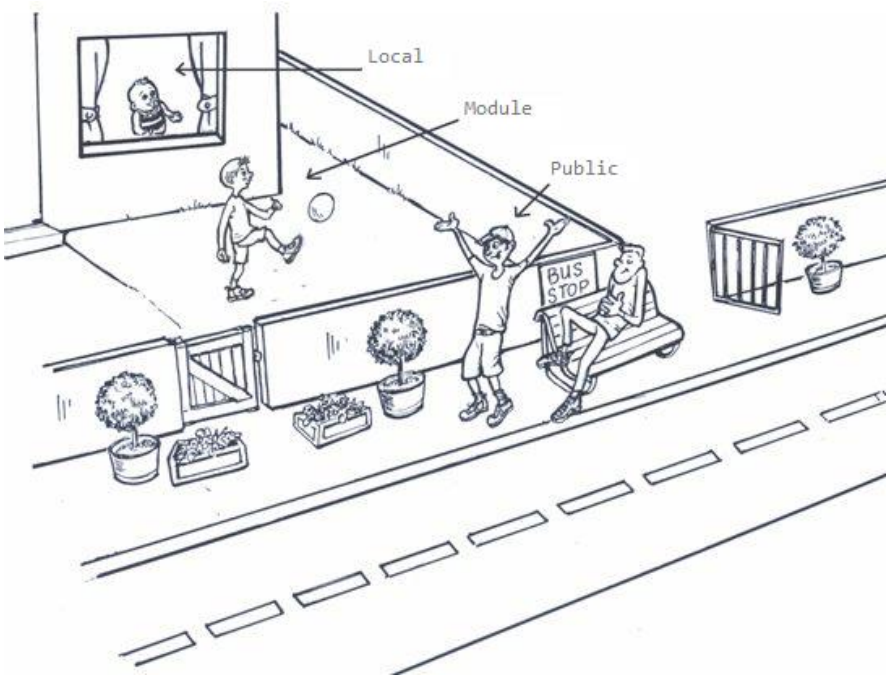
بنابراین میدان بازی سه حالت زیر است:

۱- در داخل منزل
۲- در داخل حیاط
۳- در داخل کوچه

همین قاعده هم در خصوص یک متغیر داریم و منظور از میدان دید یا Scope یک متغیر آن است که از آن متغیر در کدام ماژول ها و یا پروسیجر بتوانیم استفاده کنیم.

۱- میدان دید محدود به یک پروسیجر (Procedure Scope/ Local Scope)
۲- میدان دید محدود به یک ماژول ، (Module scope)
۳- میدان دید در همه ماژول ها (Public scope)

تصویر میدان دید (Scope) انواع متغیرها



حال به بررسی یک به یک این حالت ها می پردازیم.

میدان دید محدود به یک پروسیجر

اگر در یک پروسیجر یک متغیر تعریف شود، فقط در آن پروسیجر آن متغیر قابل استفاده است و در سایر پروسیجر ها نمی توانید از آن متغیر استفاده نمایید.

در واقع دستور End Sub باعث می شود که همه متغیرهایی که در حافظه ایجاد شده‌اند، از بین بروند و حافظه آزاد شود. در غیر اینصورت(یعنی اگر متغیرها برای همیشه در حافظه می‌ماندند)، بعد از چند بار اجرای یک برنامه کل حافظه از متغیرها پر می شود.

مثل زیر را با ۲ پروسیجر در نظر بگیرید:

```
Sub First()
    Dim x As Long
    x = 10
    Call Second
End Sub

Sub Second()

    Debug.Print x * 2

End Sub
```

حال پروسیجر First را اجرا می‌کنیم و همانطور که می بینید یک متغیر به نام x تعریف کرده ایم و عدد ۱۰ را در آن متغیر ذخیره کرده ایم .

سپس با دستور Call Second ، باعث اجرای پروسیجر دوم می شویم . در این پروسیجر خواسته ایم که مقدار x*2 را نمایش دهیم و خواهیم دید که مقدار 0 چاپ می شود یعنی پروسیجر Second اصلا خبر ندارد که ما در پروسیجر First مقدار x را 10کرده ایم.

در واقع متغیر x فقط در پروسیجر First وجود دارد و پروسیجرهای دیگر اصلا از وجود آن خبر ندارند به این متغیر که در داخل پروسیجر قابل استفاده است اصطلاحا متغیرهای Local می گوییم.

میدان دید محدود به ماژول

اگر بخواهیم که یک متغیر در تمامی پروسیجر های یک ماژول قابل استفاده باشد، کافی است که آن متغیر را در «ابتدای» آن ماژول تعریف کنیم.

```
Dim x As Long

Sub First()
    x = 10
    Call Second
End Sub

Sub Second()
    Debug.Print x * 2
End Sub
```

ما مقدار x را در پروسیجر First برابر مقدار ۱۰ کرده ایم و با دستور Call Second باعث اجرای پروسیجر Second شدیم و در پس از اجرای این پروسیجر مقدار 20 برای ما نمایش داده می شود.

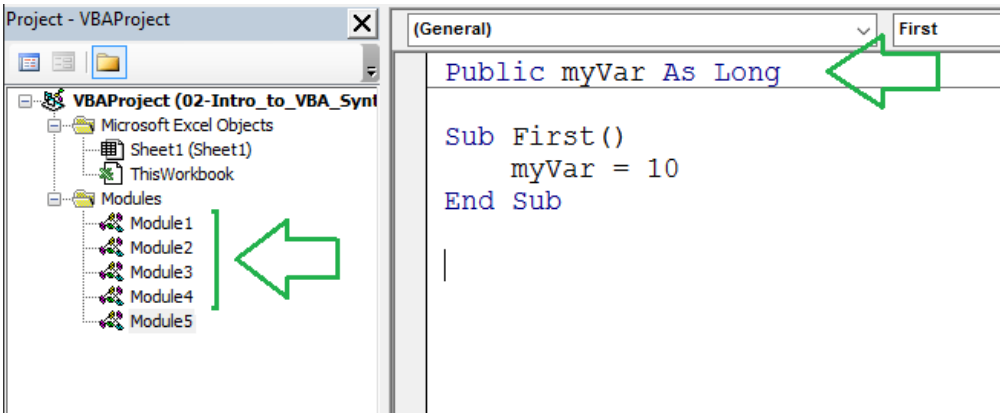
در واقع با تعریف x به صورت یک متغیر در محدوده کل ماژول، تمامی پروسیجر های آن ماژول از مقدار x که سایر پروسیجر ها مقدار دهی شده است ، خبر دارند.

نکته : اگر پروسیجر Second در یک ماژول دیگری باشد، مقدار 0 چاپ خواهد شد زیرا همانطور که گفتیم مقدار x فقط در این ماژول دیده می شود نه سایر ماژول ها.

میدان دید در همه ماژول ها

اما اگر خواسته باشیم که یک متغیر عمومی یا Public باشد یعنی در همه ماژول ها قابل استفاده شدن باشد، آنرا باید از نوع Public و در ابتدای یکی از ماژول ها تعریف می‌کنیم.

تصویر تعریف یک متغیر از نوع Public



نکته ۱: کافی است که متغیر Public در ابتدای یکی از ماژول ها تعریف شود و لازم نیست در تک تک ماژول ها آنرا تعریف کنیم.

نکته۲ : متغیر Public باید در یک ماژول تعریف شود و نمی توانید آنرا در کدهای یک شیت یا UserForm تعریف کنیم. (حالا اینها که گفتم چی هستند بماند برای بعد)

نکته ۳: شاید بپرسید که چرا همه چیز را Public تعریف نکنیم تا محکم کاری کرده باشیم و هر وقت که خواستیم از آنها استفاده کنیم. باید در پاسخ نخست بگویم که با اینکار باعث می شوید که حافظه زیادی اشغال شود و شاید برنامه هنگ کند و یا کند شود .

دلیل دوم آن است که شما یک برنامه نوشته اید و متغیری به نام x در آن وجود دارد و من نیز یک برنامه که مکمل برنامه شما است را برایتان ایمیل می کنم تا از آن در برنامه خودتان استفاده نمایید. اگر من نیز یک x از نوع Public داشته باشم، آنوقت دیگر مشخص نیست که منظور از x ، کدام x است، مقداری که من می گویم یا شما و البته برنامه خطا خواهد داد یا کلا اجرا نمی شود.

تعریف مقادیر ثابت (Constants)

مقادیر ثابت مانند متغیرها نیستند که هر لحظه بتوان آنها را تغییر داد. در واقع یک Constant چیزی است که مقداری برای آن تعریف می شود و این مقدار دیگر در طول آن برنامه قابل تغییر نیست. قوائد میدان دید متغیرها در مورد مقادیر ثابت نیز کاملا صادق است.

```
Const x as Integer = 4
```

شاید بتوان توجیه کرد که برنامه نویس با تعریف یک مقدار ثابت، مطمئن می‌شود که جلوی اشتباهاتی مانند تغییر مقدار را خواهد گرفت.

استفاده از مقادیر ثابت پیش فرض

اصطلاحا انگلیسی آن Predefined Constants است و بگذارید از مثال اندازه صفحه کاغذ در اکسل شروع کنیم. اکسل بیش از ۱۰ اندازه کاغذ دارد و اگر بخواهید در کد نویسی بنویسیم اندازه کاغذ ما A5 است باید عدد ۱۱ را به یاد داشته باشیم زیرا هر اندازه کاغذ با یک عدد مشخص شده است و عدد کاغذ A4 ، ۹ می باشد.

موافق هستید که به خاطر سپردن این اعداد سخت است و به همین دلیل به صورت پیش فرض برای این اعداد، مقادیری ثابتی از قبل تعریف شده است که ما می توانیم از آن مقادیر ثابت به جای این اعداد استفاده کنیم.

یعنی به جای آنکه بنویسیم سایز کاغذ برابر عدد ۱۱ است ،از مقدار ثابت xlPaperA5 استفاده کنیم.

تصویر مقادیر ثابت

```
Sub Chapter2_Ex12()

    Dim ws As Worksheet
    ws.PageSetup.PaperSize = |
End Sub
```

xlPaper10x14
xlPaper11x17
xlPaperA3
xlPaperA4
xlPaperA4Small
xlPaperA5
xlPaperB4

تمرین: در قسمت immediate دستورات زیر را تایپ کنید و سپس Enter را بزنید، چه اعدادی را مشاهده می کنید؟

```
print xlPaperA5
print xlDashDot
print xlCSV
```

نکته: زبان VBA نیز مقادیر ثابتی دارد که با حرف VB شروع می شوند. برای تمرین می توانید دستورات زیر را در Immediate تایپ کنید و مقدار عددی آن مقادیر را مشاهده کنید.

```
print vbred
print vbBlue
print vbYellow
Print vbYes
```

نگران اینکه این مقادیر ثابت کجا به کار شما می آیند نباشید و در فصل های آینده قطعا مثالهای فراوانی خواهید دید که از این مقادیر ثابت پیش فرض استفاده می‌کنیم.

تاریخ میلادی

برای آنکه بتوانیم از تاریخ میلادی در VBA استفاده کنیم باید آنرا در بین دو علامت «#» مانند مثال زیر قرار دهیم:

```
x = #8/8/1978#
```

نکته ۱: باید همواره در VBA تاریخ به صورت month/day/year تایپ شود.

نکته ۲: می توانیم از تابع DateSerial نیز برای تاریخ میلادی استفاده کنیم:

```
x = DateSerial(1978,8,8)
```

استفاده از توابع VBA

زبان برنامه نویسی VBA دارای توابعی است. البته تنوع و تعداد توابع آن بسیار کم است و فقط محدود به چند تابع اصلی است.

یادآوری: ویژگی تابع یا function به زبان خیلی ساده این است که ورودی می گیرد و خروجی به ما می دهد و می‌توانیم از خروجی آنها استفاده کنیم.

تمرین: پروسیجر زیر را تایپ و سپس اجرا کنید. هر تابع چه کاری انجام می دهد.

```
Sub Chapter2_Ex13()

   Dim myStr As String
   Dim myVal As Long
   mystr = "Iran"
   myVal = 25

   Debug.Print Left(mystr, 2)
   Debug.Print Sqr(myVal)
   Debug.Print StrReverse(mystr)
   Debug.Print UCase(mystr)
   Debug.Print IsNumeric(myVale), IsNumeric(mystr)
   Debug.Print Date
   Debug.Print Now()

End Sub
```

نکته: البته می توانیم در محیط VBA از توابع Excel استفاده کنیم که این نکته را در فصول آینده که با Objectها آشنا شده باشیم، خواهید دید.

معرفی تابع MsgBox

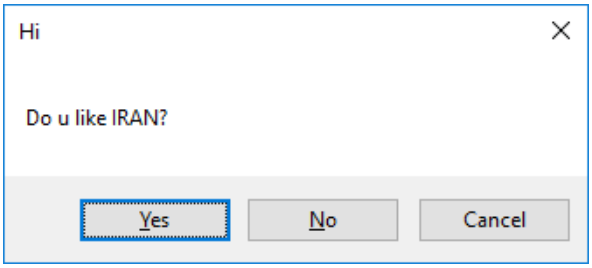
شما قبلا با MsgBox آشنا شده اید و با آن یک پیغام را به کاربر نمایش دادید و اکنون به شما می‌گویم که MsgBox یک تابع است! یعنی خروجی دارد! بله ، خروجی دارد.

در واقع MsgBox یک پیغام را به کاربر نمایش می دهد و سپس کاربر باید برای بستن پیغام، یکی از دکمه‌های Ok , Yes و ... را کلیک کنید . خروجی این تابع در واقع به ما می‌گوید که کاربر چه کلیدی را کلیک کرده است.

بگذارید پروسیجر زیر را اجرا کنیم و با این مثال تابع را بررسی کنیم.

```
Sub Chapter2_Ex14()
      Dim answer As Integer
      answer = MsgBox("Do u like IRAN?", vbYesNoCancel ,"Hi")
      Debug.Print answer
End Sub
```

تصویر MsgBox با ۳ کلید Yes/No/Cancel



شرح برنامه:

ما یک متغیر از نوع Integer به نام answer تعریف کرده ایم بنابراین می تواند یک عدد تا ۳۲ هزار را در خودش ذخیره کند.

سپس از تابع MsgBox استفاده کردیم و خروجی تابع MsgBox را در متغیر answer ذخیره کرده ایم و در آخر هم مقدار answer را با نمایش داده‌ایم و یک عدد برای ما نمایش داده شد.

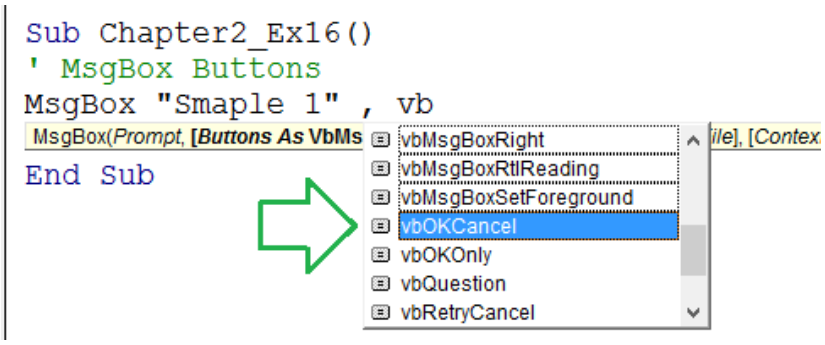
حال می‌خواهیم تا خود تابع MsgBox را بررسی کنیم .این تابع پنج ورودی دارد که ما ۳ تای اول آنرا بررسی میکنیم:

```
MsgBox(prompt[, buttons] [, title] [, helpfile, context])
```

Prompt : هر پیغام دلخواهی می تواند باشد و این پیغام به کاربر نمایش داده می‌شود.

Buttons: این ورودی اختیاری است و می توانید اصلا آنرا نگذارید. در واقع اگر هیچ دکمه ای تعریف نکنید، همان دکمه Ok نمایش داده می شود و شما می توانید هر یک از ترکیب‌های استاندارد را انتخاب کنید. (حالت تکمیل خودکار به شما در انتخاب کمک می کند).

تصویر انتخاب دکمه‌های مختلف



نکته ۱: در قسمت Buttons می توانیم آیکون پیام را نیز تغییر دهیم ، بگذارید من توضیح ندهم و خودتان کدهای زیر را اجرا کنید:

```
Sub Chapter2_Ex18()
      MsgBox "Smaple 1", vbOKOnly + vbCritical
      MsgBox "Smaple 1", vbOKOnly + vbInformation
      MsgBox "Smaple 1", vbOKOnly + vbExclamation
End Sub
```

Tilte : عنوان دلخواه برای پنجره است. تست کنید و نتیجه را ببنید.

نکته ۲: هنگامی که ما از MsgBox به شکل یک تابع استفاده می‌کنیم بدیهی است که باید در سمت راست علامت مساوی نوشته شود و بنابراین باید پرانتز را بگذاریم.

```
answer = MsgBox("Do u like IRAN?", vbYesNoCancel ,"Hi")
```

اما اگر فقط می خواستیم یک پیام را نمایش دهیم و نیازی به ذخیره چیزی نداشتیم، نباید از پرانتز استفاده کنیم.

```
MsgBox "I Like Iran."
```

این قاعده در مورد سایر چیزها هم صادق است یعنی وقتی که چیزی به شکل تابع استفاده می-شود و در سمت راست علامت مساوی قرار می گیرد، باید پارامترهایش داخل پرانتز قرارداده شود.

تذکر : هنگامی که IF را فراگرفتید، می توانید از پاسخی که کاربر در پاسخ MsgBox می‌دهد، استفاده کنید مثلا به کاربر پیام دهید که آیا شیت حذف شود، و اگر او پاسخ Yes را داد، آنوقت حذف انجام شود.

نکته ۳: اگر خواستیم که پیغام در چند سطر نمایش دهیم، برای آنکه بتوانیم Enter را در پیغام اضافه کنیم از یکی از تکینک های زیر استفاده کنید:

```
Sub Chapter2_Ex15()
    MsgBox "Note!" & vbNewLine & "Hi  Second Line"
    MsgBox "Note!" & Chr(13) & "Hi  Second Line"
    MsgBox "Note!" & vbCr & "Hi  Second Line"
    MsgBox "Note!" & vbCrLf & "Hi  Second Line"
End Sub
```

هیچ یک تفاوتی نمی کند از کدام دستور استفاده کنید .من از دستور اولی vbNewLine به دلیل اینکه واضح تر است استفاده می‌کنیم.

نکته ۴: VBA نرم افزاری Non-Unicode است. یعنی بجز انگلیسی، زبان دیگری را متوجه نمی‌شود و به همین دلیل ما نمی‌توانیم در آن از پیغام‌های فارسی استفاده کنیم. البته برای حل این مشکل، راهکارهای وجود دارد که در فصول آینده به آن‌ها خواهیم پرداخت.

معرفی تابع InputBox

اگر بخواهیم از کاربر یک مقداری را بگیریم، از این تابع استفاده می‌کنیم. مثلا می‌خواهید از کاربر شعاع یک دایره را بگیرید و سپس مساحت آن دایره را به او نمایش دهید.

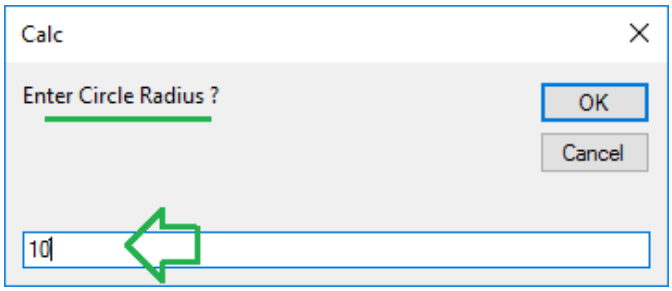
```
Sub Chapter2_Ex19()
    Dim r As Double
    Dim s As Double
    Const pi As Double = 3.141592

r = InputBox("Enter Circle Radius ?", "Calc", 10)
s = pi * (r ^ 2)
MsgBox "your Circle Areas is:" & _
        vbNewLine & _
        Format(s, "#,#.00")

End Sub
```

شرح برنامه:

من کمی برنامه بالا را حرفه ای نوشتم و شاید در عمل اینقدر دقیق لازم نباشد مثلا می توانیم قسمت Const pi As Double = 3.141592 را حذف کنیم و در محاسبه s عدد 3.14 را به جای مقدار ثابت pi بنویسیم.

ما توسط InputBox("Enter Circle Radius ?", "Calc", 10) از کاربر یک مقدار را می گیریم و در متغیر r که از نوع Double است قرار می دهیم . مقدار پیش فرض r عدد ۱۰ است.

تصویر InputBox با مقدار پیش فرض 10



مساحت را محاسبه می‌کنیم و در متغیر s ذخیره می‌کنیم و سرانجام مقدار s را به کاربر در حالتی که سه رقم سه رقم جدا شده است و تا ۲ رقم اعشار، توسط تابع Format به کاربر نمایش می‌دهیم.

نکته: دو تابع MsgBox و InputBox مخصوص VBA هستند و همانطور که گفتیم فرقی نمی کند که شما از این توابع در اکسل، اکسس و یا پاورپوینت استفاده کنید. جالب است بدانید که Excel یک InputBox مخصوص خودش را دارد که در فصل Formها به آن اشاره خواهیم کرد و با دستور Application.InputBox از آن می‌توان استفاده کرد.

فصل چهارم – دستورات شرطی و کنترل برنامه

حال که با مفاهیم پایه‌ای VBA آشنا شدیم وقت آن است که بتوانیم روند اجرای برنامه را کنترل کنیم. قبلا گفتیم که دستورات (Statements) یک پروسیجر سطر به سطر و یکی پس از دیگری اجرا می‌شوند.

اکنون می‌خواهیم با دستوراتی مانند IF برنامه را کنترل کنیم تا چه دستوری در چه شرایطی اجرا شود. در این فصل با دستورات Goto , Select Case آشنا خواهیم شد.

معرفی دستور GoTo

قبل از شروع توضیح دستور GoTo لازم است یک نکته را بدانید. در نسخه‌های بسیار قدیمی زبان Basic (وقتی من دبیرستانی بودم)، نوشتن شماره خط اجباری بود. یعنی هر سطر دارای یک شماره بود:

تصویری از شماره خط‌ها در نسخه‌های قدیمی بیسیک

```
240 REM tHE METROPOLIS-HASTINGS ALGORITHM
250 FOR I = 1 TO NSAMPLES - 1
260  FOR J = 0 TO NPARAMS - 1
270    PARAMS!(J) = SAMPLES!(I - 1, J)
280  NEXT J
285  REM
290  GOSUB 520
300  CURRDENS! = LOGDENS!
310
```

حال در VBA شماره سطر اجباری نیست و اگر شما بخواهید می‌توانید برای سطر ها یک شماره و یا یک نام بگذارید.

حال با استفاده از دستورGoTo می‌توانیم به برنامه بگوییم که کدام سطر باید اجرا شود. مثلا اگر بنویسیم GoTo 100 یعنی باید سطر شماره 100 اجرا شود.

نکته ۱: نامگذاری سطرها با علامت «:» می باشد.

نکته ۲: برای نامگذاری می‌توان از عدد و یا حروف استفاده کرد.

نکته ۳: لازم نیست که حتما در جلوی شماره سطر حتما دستوری را بنویسیم و می‌توانیم جلوی آنرا خالی بگذاریم.

```
Sub Chapter4_Ex1()

Debug.Print "One"
    GoTo 40
20: Debug.Print "Three"
30: Debug.Print "Four"
MyLine:
40: Debug.Print ""

End Sub
```

شرح برنامه: همانطور که گفتیم برنامه سطر به سطر اجرا و تفسیر می‌شود و هنگامی که سطر Goto 40 اجرا شود، برنامه به سطر 40 می‌پرد و سایر دستورات که در سطرهای MyLine,30,20 هستند اجرا نمی‌شوند.

نکته: ما در برنامه‌ها به ندرت از Goto ها معمولا استفاده نمی‌کنیم زیرا باعث می‌شود که برنامه دارای پرش‌های زیادی شود و معمولا یافتن منطق برنامه سخت می‌شود. اصطلاحا می‌گویند برنامه اسپاگتی، یعنی یک کلاف سردر گم شده است.

با کلید F8 آشنا شوید

از آنجایی که یک برنامه ممکن است از دستورات IF یا Goto استفاده کند، درک روند اجرای برنامه و خطایابی آن برای برنامه نویسان دشوار می‌شود.

بنابراین برنامه نویسان برای درک یک پروسیجر آن را به یکباره اجرا نمی‌کنند بلکه سطر به سطر آنرا اجرا می‌کنند و با اینکار متوجه می‌شویم پس از اجرای هر دستوری چه اتفاقی خواهد افتاد.

برای این کار از کلید F8 در VBE استفاده می‌کنیم.

کافی است که در بین دستورات یک پروسیجر کلیک کنید و سپس کلید F8 را بزنید. خواهید دید که اولین سطر، زرد رنگ می‌شود. هر سطری که زرد رنگ است یعنی آن سطر آماده اجرا شده است و همین که مجددا کلید F8 را بزنید، آن سطر اجرا می‌شود و دستور بعدی زرد (آماده اجرا) می‌شود.

تصویر اجرای سطر به سطر برنامه با کلید F8

```
Sub Chapter4_Ex1()

Debug.Print "One"
    GoTo 40
20: Debug.Print "Three"
30: Debug.Print "Four"
35:
40: Debug.Print ""

End Sub
```

نکته: به این کارها، یعنی سعی در خطایابی برنامه و درک نحوه کارکرد آن به صورت کلی Debug کردن می‌گویند. شما می‌توانید به جای زدن کلید F8 از منوی Debug → Step Into استفاده کنید.

آشنایی با IF

دستور IF در VBA از همان اصول کلی IF در اکسل پیروی می‌کند یعنی یک عبارت یا دستور شرطی از سه بخش اصلی تشکیل می‌شود:

```
IF condition Then  statement_1 Else statement_2
```

بگذارید که ابتدا در مورد این شکل از دستور IF توضیحاتی کلی را بدهیم:

Condition: یک عبارت منطقی است، یعنی باید چیزی که در قسمت Condition نوشته می‌شود حاصلش True یا False باشد. مثال‌های زیر نمونه‌هایی از Conditionهای مختلف در دستور IF است:

```
IF x >= 2
IF name = "Iran"
```

Statement_1: اگر خاطرتان باشد گفتیم Statement یعنی یک دستور و یعنی در این قسمت شما می‌توانید هر دستور VBA را بنویسید و اگر حاصل قسمت Condition مقدار True شد آنگاه این دستور اجرا می‌شود.

```
IF x >= 2 Then MsgBox "Ok"
IF name = "Iran" Then y = y * 2
```

Statement_2: اگر حاصل قسمت Condition شما مقدار False شد، آنگاه این دستور اجرا خواهد شد.

```
IF x >= 2 Then MsgBox "Ok" Else MsgBox "Reject"
IF name = "Iran" Then y = y * 2 Else y = y / 2
```

و اگر بخواهیم که این دستورات را به فارسی ترجمه کنیم، می‌توانیم اینگونه آنها را اینگونه بیان کنیم:

- اگر x بزرگتر یا مساوی عدد 2 بود آنگاه پیغام Ok را نمایش بده و در غیر اینصورت پیغام Reject را .
- اگر name برابر Iran بود آنگاه y = y * 2 شود و در غیر اینصورت y = y /2

از آنجایی که من می‌خوام اطمینان پیدا کنم که شما خوب IF را یاد گرفته‌اید، چندین مثال می‌زنم و لطفا شما هم آنها را تایپ و اجرا کنید.

مثال ۱ دستور IF – قبول شدید

برنامه‌ای بنویسید که یک نمره یک کاربر را بگیرد و اگر نمره او بیش از ۱۰ شده بود به او بگوید که آفرین، قبول شدی و اگر کمتر از ۱۰ بود بگوید اشکالی ندارد.

```
Sub Chapter4_Ex2()
Dim x As Double
x = InputBox("Nomereh ra vared konid.")

If x >= 10 Then MsgBox "Afarin, Ghabool shodi" Else MsgBox "Eshkali Nadareh"

End Sub
```

تصویر دستور IF مثال اول

```
If x >= 10 Then MsgBox "Afarin, Ghabool shodi" Else MsgBox "Eshkali Nadareh"
```

Condition — Statement_1 — Statement_2

شرح برنامه: از کاربر یک عدد گرفته می‌شود و در متغیر x ذخیره می‌شود. سپس در قسمت Condition دستور IF، شرط x >=10 را بررسی می‌کنیم که آیا این شرط درست است و یا غلط.

اگر پاسخ x >= 10 مقداری True بود (یعنی اگر شرط ما صحیح بود) آنگاه دستوری که در قسمت Statement_1 نوشتیم اجرا می‌شود و در غیر اینطورت (یعنی اگر پاسخ False بود)، دستوری که در Statement_2 نوشتیم، اجرا خواهد شد.

تمرین ۱: برنامه بالا را با کلید F8 اجرا کنید.

تمرین ۲: هنگامی که از شما برای مقدار x سوال می‌کنید ، به جای یک عدد ، یک کلمه وارد کنید، چه اتفاقی خواهد افتاد؟

مثال ۲ دستور IF – ازدواج کردی

برنامه‌ای بنویسید که از کاربر سوال کند که آیا ازدواج کرده‌است. اگر پاسخ او Yes بود به پیام آفرین را نمایش دهد واگر پاسخ او No بود، به او پیام صد آفرین را نمایش بدهد.

```
Sub Chapter4_Ex3()

Dim answer As Integer
answer = MsgBox("Aya ezdevaj kardid?", vbYesNo)

If answer = vbYes Then MsgBox "Afarin, :)" Else MsgBox "100 Afarin, :D"

End Sub
```

شرح برنامه: گفتیم که MsgBox یک تابع است و می‌توانیم کلیدی را که کاربر انتخاب می‌کند را به عنوان خروجی این تابع داشته باشیم بنابراین خروجی MsgBox را در متغیری به نام Answer ذخیره می‌کنیم و اگر خاطرتان باشد خروجی MsgBox یک عدد است.

در دستور IF می‌خواهیم بدانیم که کاربر آیا Yes را زده است و یا نه. به همین خاطر قسمت Condition ما به صورت answer = vbYes نوشته شده است.

تکلیف MsgBox ها که برای نمایش پیام ها هم بکار می‌رود مشخص است. (این MsgBox ها فقط یک پیام به کاربر نمایش می‌دهند و لازم نداریم انتخاب کاربر را بدانیم ، البته در هر حال کاربر کلید OK را خواهد زد . چون یک کلید بیشتر ندارد)

مثال ۳ دستور IF – عدد زوج یا فرد

برنامه‌ای بنویسید که یک عدد از کاربر بگیرد و مشخص کند که این عدد زوج است یا فرد.

```
Sub Chapter4_Ex4()
      Dim Num As Long
      Num = InputBox("Addadi varid konid")

      If Num Mod 2 = 0 Then MsgBox "Zoj" Else MsgBox "Fard"
End Sub
```

شرح برنامه: عدد زوج ، عددی است که اگر آنرا بر ۲ تقسیم کنیم، باقیمانده‌اش صفر خواهد شد و برای یافتن باقیمانده از عملگر Mod استفاده می‌کنیم Condition ما عبارت Num Mod 2 = 0 است که بررسی می‌کند آیا باقیمانده صفر شده است.

یادآوری: ما متغیر Num را از نوع Long گرفتیم یعنی عددی در بازه ۲ میلیارد مثبت تا ۲ میلیارد منفی.

سوال: فرض کنید که کاربر به جای آنکه یک عدد صحیح تایپ کند، یعنی عدد اعشاری می‌دهد مثلا 19.99 ، آنوقت این عدد زوج است یا فرد! چرا؟

مثال ۴ دستور IF – بخش پذیری

برنامه‌ای بنویسید که ۲ عدد صحیح را از کاربر بگیرید و اگر عدد اول بر عدد دومی بخش پذیر بود، پیام بخش پذیر است را نمایش دهد.

```
Sub Chapter4_Ex5()

      Dim Num1 As Long, Num2 As Long
      Num1 = InputBox("Adad Avali ra vared Konid")
      Num2 = InputBox("Adad Dovom ra vared Konid")
      If Num1 Mod Num2 = 0 Then MsgBox "Bakhsh pazi ast:"

End Sub
```

بررسی برنامه: نکته خاصی این برنامه ندارد. به نحوه تعریف دو متغیر Num1 , Num2 دقت کنید در ضمن اینکه نوشتن قسمت Else در دستور IF الزامی نیست و ما در برنامه فقط قسمت Statement_1 را نوشتیم.

آشنایی با عملگرهای منطقی

همانطور که گفتیم در قسمت Condition دستور IF باید یک عبارت منطقی بنویسیم یعنی چیزی بنویسیم که نتیجه آن True یا False شود مثلا x =>10 و یا Num1 Mod 2 = 0 .

حال اگر به خاطر داشته باشیم ما گفتم که یک چندین عملگر (Operator) داریم مانند * ، / ، + که به آنها عملگرهای ریاضی می‌گوییم یعنی روی مقادیر عددی کار می‌کنند.

اکنون می‌خواهیم که شما را با عملگرهای منطقی در VBA آشنا کنیم ، عملگرهای منطقی ، آنهایی هستند که روی مقادیر منطقی (یعنی True / False) کار می‌کنند.

آشنایی با And

اگر دو عبارت یا مقدار منطقی داشته باشیم (یعنی مقدار آنها True یا False باشد)، آنگاه آن دو عبارت یا مقدار را با هم And کنیم نتیجه اینگونه می‌شود:

- اگر اولی True باشد و دومی هم True نتیجه And آنها True می‌شود.
- اگر اولی False باشد و دومی هم True نتیجه And آنها False می‌شود.
- اگر اولی True باشد و دومی هم False نتیجه And آنها False می‌شود.
- اگر اولی False باشد و دومی هم False نتیجه And آنها True می‌شود.

فکر کنم که این نحوه توضیح دادن خیلی کار را شلوغ می‌کند و به همین دلیل معمولا توضیحات بالا را به صورت خلاصه در جدولی مانند زیر می‌نویسند.

(واژه «عبارت» ترجمه Expression است که به صورت خلاصه در جدول زیر Exp نوشته شده است)

Exp 1	Exp 2	Exp1 **And** Exp2
TRUE	TRUE	TRUE
FALSE	TRUE	FALSE
TRUE	FALSE	FALSE
FALSE	FALSE	FALSE

مثال ۵ دستور IF – مثلث متساوی الاضلاع

برنامه‌ای بنویسید که سه عدد به عنوان طول سه ضلع یک مثلث را بگیرد و مشخص کنید که آیا این مثلث متساوی الاضلاع است یا نه.

یادآوری: مثلث متساوی الاضلاع، مثلثی است که هر سه ضلع آن برابر باشند.

```
Sub Chapter4_Ex6()

Dim Num1 As Double, Num2 As Double, Num3 As Double
Num1 = InputBox("Enter Side 1 ?")
Num2 = InputBox("Enter Side 2 ?")
Num3 = InputBox("Enter Side 3 ?")

If Num1 = Num2 And Num2 = Num3 Then MsgBox "This is Equilateral"
End Sub
```

شرح برنامه: سه متغیر از نوع Double در نظر گرفتیم و با InputBox آنها را مقدار دهی کردیم. سپس با AND بررسی کردیم که آیا اولین ضلع مساوی دومین ضلع است و همچنین آیا دومین ضلع مساوی سومین ضلع است و اگر پاسخ این بررسی مقدار True شود، پیغامی به کاربر نمایش داده می‌شود.

نکته ۱: لازم نیست که ضلع سوم و اولی را مقایسه کنیم زیرا اگر Num1=Num2 باشد و Num2=Num3 حتما Num1=Num3 است.

نکته ۲: بهتر است که هر عبارت را برای خوانایی بیشتر در داخل پرانتز بنویسیم:

```
If (Num1 = Num2) And (Num2 = Num3) Then MsgBox "This is Equilateral"
```

نکته ۳: ما می توانیم معادل فارسی And را واژه «همچنین» در نظر بگیریم مثلا بگوییم اگر ضلع اول برابر دومی بود و «همچنین» اگر دومی با سومی برابر بود، آنگاه آن مثلث متساوی الاضلاع است.

آشنایی با OR

امیدوارم که از جدول زیر بتوانید متوجه شوید که عملگر OR چگونه کار می‌کند و نتیجه OR دو عبارت منطقی چه خواهد شد.

Exp 1	Exp 2	Exp1 **OR** Exp2
TRUE	TRUE	TRUE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	FALSE	FALSE

ما در فارسی OR را، «یا» ترجمه می‌کنیم.

مثال ۵ دستور IF – مثلث متساوی الساقین

برنامه‌ای بنویسید که سه عدد به عنوان طول سه ضلع یک مثلث را بگیرد و مشخص کنید که آیا این مثلث متساوی الساقین است یا نه. برای ساده تر شدن این مثال فرض می‌شود که مثلث ما متساوی الاضلاع نیست و فقط می‌خواهیم بدانیم متساوی الساقین است یا نه.

یادآوری: مثلث متساوی الساقین، مثلثی است که دو ضلع آن برابر باشند.

```
Sub Chapter4_Ex7()

Dim Num1 As Double, Num2 As Double, Num3 As Double
Num1 = InputBox("Enter Side 1 ?")
Num2 = InputBox("Enter Side 2 ?")
Num3 = InputBox("Enter Side 3 ?")

If Num1 = Num2 Or Num2 = Num3 Or Num1 = Num3 Then MsgBox "This is Isosceles"
End Sub
```

شرح برنامه: با عملگر OR هر ضلع را با دو ضلع دیگر مقایسه می‌کنیم تا بدانیم که آیا باهم برابر هستند یا خیر.

نکته ۱: همانطور که محدودیتی برای بکارگیری عملگر جمع ندارید و می‌توانید چندین مقدار را با هم جمع کنیم، برای عملگر OR یا And و ... هیچ محدودیتی نداریم که بنویسیم:

```
Exp1 And Exp2 And Exp3 And Exp4
```

نکته ۲: اگر می‌توانیم عملگر جمع و تفریق را باهم بکار ببریم، می‌توانیم عملگرهای منطقی را نیز با هم ترکیب کنیم . مثلا بنویسیم :

```
Exp1 And Exp2 OR Exp3
```

نکته ۳: همانطور که می‌دانید ضرب بر جمع تقدم دارد و عملگرهای منطقی نیز بر هم تقدم‌هایی دارند و از آنجایی که حفظ کردن این تقدم‌ها ضرورتی ندارد، ما از پرانتز برای آنکه معلوم کنیم چه عبارتی ابتدا باید محاسبه شود، استفاده می‌کنیم :

```
(Exp1 And Exp2) OR Exp3
(Exp1 And Exp2) OR (Exp3 And Exp3)
```

مثال ۶ دستور IF – هوای مناسب

برنامه‌ای بنویسید که یک عدد را به عنوان دمای هوا بگیرد و اگر آن عدد بین مقدار ۱۸ تا ۲۴ بود، پیغام نمایش دهد که هوا مناسب است.

```
Sub Chapter4_Ex8()
      Dim temp As Double
      If (temp >= 18) And (temp <= 24) Then MsgBox "Hava Monaseb ast"
End Sub
```

نکته: همواره برای آنکه بدانیم یک عدد در یک بازه (بین دو عدد است) باید از عملگر And استفاده کنیم . اکسل عملگری مشابه Between در Sql را ندارد.

آشنایی با NOT

این عملگر خیلی ساده است و نتیجه را معکوس می‌کند. یعنی اگر عبارت ما مقدارش True است و ما آنرا با عملگر Not ترکیب کنیم، حالش Flase می‌شود.

Exp 1	**Not** Exp1
TRUE	False
FALSE	TRUE

مثال ۷ دستور IF – خاموش / روشن کردن

تقریبا من هر مثالی من از NOT برای شما بزنیم، می‌توان آنرا با AND یا OR و یا علامت <> هم نوشت. مثلا می‌خواهیم که بررسی کنیم عدد x برابر صفر نباشد. می‌توانم بنویسم x <> 0 و یا اینکه آنرا با NOT اینگونه بنویسیم NOT(x = 0) .

برای آنکه یک مثال کاربردی بزنم اجازه بدهید که یک دستوری بکار ببرم که هنوز برای شما آنرا شرح نداده‌ام، اما کاربرد NOT را در یک مثال عملی خواهید دید.

برنامه‌ای بنویسید که با اجرای آن Gridlineهای اکسل خاموش یا روشن بشود. در واقع باید برنامه شبیه کلید خاموش و روشن کامپیوتر عمل کند یعنی اگر Gridline ها روشن هستند،آنها را خاموش کند و اگر خاموش هستند، آنها را روشن کند.

```
Sub Chapter4_Ex9()
      ActiveWindow.DisplayGridlines = Not (ActiveWindow.DisplayGridlines)
End Sub
```

شرح برنامه: نتیجه عبارت ActiveWindow.DisplayGridlines یک مقدار منطقی است یعنی اگر Gridlineها روشن باشند ، مقدارش True است و اگر خاموش باشند مقدارش False .

قبلا یادگرفتیم که ابتدا عبارت سمت راست مساوی محاسبه می‌شود.

فرض کنید که اکنون Gridlineهای اکسل روشن هستند وبنابراین مقدار عبارت ActiveWindow.DisplayGridlines برابر True خواهدبود و هنگامی که آنرا NOT می‌کنیم، برای ما مقدار Flase می شود و در نتیجه کل سمت راست مساوی False خواهد شد.

حال که سمت راست مقدار Flase شده است پس اکسل باید آن خط را اینگونه تفسیر کند:

```
ActiveWindow.DisplayGridlines = False
```

که نتیجه این دستور خاموش شدن Gridlineها خواهد بود.

تمرین: در داخل شیت اکسل خود یک دکمه / شکل قرار دهید و آن را به این ماکرو Assign کنید. سپس هربار که دکمه را کلیک کنید، خواهید دید که وضعیت Gridlineها برعکس حالتی قبل از کلیک شما خواهند شد.

سایر عملگرها

ما در VBA عملگرهای دیگری را داریم و از آنجایی که معمولا کمتر استفاده می‌شوند من اینجا فقط جهت آشنایی آنها را ذکر می‌کنم. در ضمن آنکه می‌توانیم این عملگرها را با ترکیب سایر عملگرها هم بسازیم و معمولا کاربران چون AND / OR / NOT آشنایی بیشتری دارند، از آنها استفاده می‌کنند:

عملگر Xor

وقتی مقدارش True می‌شود که یکی از دو عبارت True باشد و نه هر دوی آنها.

Exp 1	Exp 2	Exp1 **Xor** Exp2
TRUE	TRUE	False
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	FALSE	FALSE

عملگر Eqv

وقتی True می‌شود که هر دو عبارت یا True باشند و یا هر دو False

Exp 1	Exp 2	Exp1 **Eqv** Exp2
TRUE	TRUE	TRUE
FALSE	TRUE	FALSE
TRUE	FALSE	FALSE
FALSE	FALSE	TRUE

حروف کوچک و بزرگ در IF

همانطور که گفتیم در تایپ دستورات و یا متغیرهای VBA حروف کوچک و یا بزرگ انگلیسی مهم نیستند و با هم برابر می باشند.

اما هنگامی که می‌خواهیم یک Condition بنویسیم و دو متن را مقایسه کنیم، آنوقت حروف کوچک و بزرگ کاملا مهم خواهند شد و دو چیز کاملا متفاوت هستند.

مثلا فرض کنید که کدهای زیر را داریم

```
Dim s As String
s = "Iran"
If s = "iran" Then MsgBox "Welcome"
```

همانطور که می‌بینید متغیر s مقدارش کلمه Iran با حرف بزرگ انگلیسی است و در دستور IF، مقدار S را با کلمه iran که با حروف کوچک انگلیسی شروع می‌شود، مقایسه کرده‌ایم .چون حرف i بزرگ و کوچک در مقایسه‌ها دو چیز کاملا مختلف هستند، بنابراین "s="iran نیست و مقدار این عبارت منطقی False می‌شود و بنابراین پیغامی نمایش داده نخواهد شد.

حال فرض کنید که برای شما کوچک و بزرگی حروف مهم نباشد و فقط می‌خواهید اگر s مقدارش برابر با کلمه ایران (البته انگلیسی) است پیام خوش آمد را نمایش دهید. برای اینکار از روشی بسیار ابتدایی استفاده می‌کنیم . در واقع ابتدا هر دو کلمه را به حالت بزرگ (یا کوچک) در می‌آوریم و سپس آنها را باهم مقایسه می‌کنیم:

```
If UCase(s) = "IRAN" Then MsgBox "Welcome 2"
```

نکته ۱: تابع Ucase برای تبدیل حروف به حالت بزرگ انگلیسی استفاده می‌شود و تابع Lcase برای حالت تبدیل به حروف کوچک .

نکته ۲: بدیهی است که Space در مقایسه‌ها مهم است یعنی اگر در انتهای کلمه Iran یک Spaceهم تایپ شود و سپس بخواهیم آنرا با Iran که هیچ Space ی ندارد مقایسه کنیم، این مقایسه False خواهد شد. برای اینکار یعنی حذف Space از ابتدا و انتهای یک متن در VBA به ترتیب از توابع LTrim و RTrim استفاده می‌کنیم و برای سادگی آنها را اینگونه بکار می‌بریم:

```
Dim s As String
s = " Iran "
If RTrim(LTrim(s)) = "Iran" Then MsgBox "Welcome 3"
```

نکته ۳: اگر چه ما بحث مقایسه حروف کوچک و بزرگ انگلیسی را در بخش IF ها مطرح کرده‌ایم، اما این موضوع در هر دستوری که مقایسه‌ای انجام می‌شود، صادق است و گمان نکنید که فقط در دستور IF حالت کوچک و بزرگی حروف انگلیسی معنا دارد.

دستور IF بلوکی

در VBA ما می‌توانیم دستور IF را به دو شکل بنویسیم، شکل ساده و شکل بلوکی و تا کنون از شکل ساده این دستور استفاده می‌کردیم.

شکل دستور IF ساده:

```
IF condition Then  statement_1 Else statement_2
```

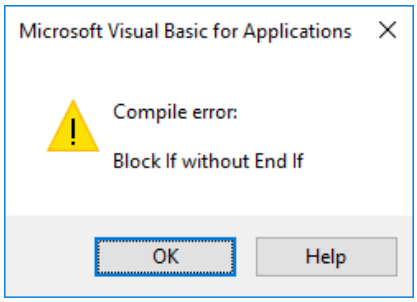
شکل دستور IF بلوکی:

```
IF condition Then
      statement
      statement
      …
Else
      statement
      statement
      …
End If
```

فکر کنم که متوجه مزیت دستور IF به صورت بلوکی (یا چند سطری) شده‌اید، در واقع در این حالت می‌توانیم چندین دستور را در صورت برقراری شرط و یا عدم برقراری آن تایپ کنیم. اما در حالت ساده دستور IF فقط می‌توانیم یک دستور را تایپ کنیم.

نکته ۱: حتما هر دستوری که به صورت بلوک (چند سطری) نوشته می‌شود، باید دارای انتها باشد و در VBA از دستور End IF برای اینکه مشخص کنیم انتهای بلوک IF کجاست، استفاده می‌شود. اگر انتهای IF بلوکی را مشخص نکنیم، خطایی قبل از اجرا داده می‌شود.

تصویر خطای استفاده از IF بلوکی بدون مشخص کردن انتهای بلوک با End IF



مثال ۸ دستور IF – رسم مستطیل یا مثلث

خواهیم برنامه‌ای بنویسیم که از کاربر یک عدد بگیرد و در صورتی که آن عدد مثبت بود، برای او یک مستطیل در قسمت Immediate رسم کند و اگر عدد منفی بود برای او یک مثلث.

```
Sub Chapter4_Ex14()

Dim num As Long
num = InputBox("Enter a number , plz")
If num > 0 Then
    Debug.Print "**********"
    Debug.Print "*        *"
    Debug.Print "**********"
Else
    Debug.Print "  * "
    Debug.Print " * *"
    Debug.Print "*****"
End If

End Sub
```

شرح برنامه: ما برای رسم مستطیل و مثلث از چاپ چندین ستاره کمک گرفتیم و چون باید اینکار را با چند دستور Debug.Print انجام دهیم، بنابراین از IF بلوکی استفاده کرده‌ایم.

نکته : همانطور که می‌بینید با گذاشتن Tab در ابتدای خطوط وضوح و خوانایی برنامه را افزایش داده‌ایم و با یک نگاه مشخص است که کدام دستورات در قسمت Then هستند و کدام یک در قسمت Else.

استفاده از ElseIF

ما در VBA می‌توانیم IF های پیچیده‌تری را بنویسیم که چندین شرط را فقط با یک دستور IF بررسی کنیم.

توجه داشته باشید که این کار در اکسل امکان پذیر نیست، یعنی نمی‌توانستیم با یک دستور IF چندین شرط را بررسی کنیم و باید حتما از چندین دستور IF استفاده می‌کردیم.

مثال ۹ دستور IF – مقایسه دو عدد

برنامه‌ای بنویسید که ۲ عدد را از کاربر دریافت کند و مشخص کنید که عدد اول از عدد دوم بزرگتر است یا کوچکتر است و یا مساوی هم هستند.

```
Sub Chapter4_Ex15()
Dim Num1 As Double
Dim Num2 As Double

Num1 = InputBox("Enter Num1"): Num2 = InputBox("Enter Num2")

If Num1 > Num2 Then
    MsgBox "Num1 > Num2 "
ElseIf Num1 < Num2 Then
    MsgBox "Num1 < Num2 "
Else
    MsgBox "Num1 = Num2"
End If

End Sub
```

نکته ۱: ما لازم نیست که شرط Num1=Num2 را بررسی کنیم یعنی بنویسیم ElseIf Num1=Num2 چون اگر Num1 بزرگتر یا کوچکتر از Num2 نباشد، قطعا مساوی آن است و از Else استفاده می‌کنیم.

نکته ۲: خیلی از تازه کارها در برنامه نویسی به جای استفاده از ElseIF از چندین دستور IF مستقل استفاده می‌کند. به عنوان مثال برنامه بالا را به صورت زیر می‌نویسند:

```
If Num1 > Num2 Then MsgBox "Num1 > Num2 "
If Num1 < Num2 Then MsgBox "Num1 < Num2 "
If Num1 = Num2 Then MsgBox "Num1 = Num2 "
```

این روش همان پاسخ را می‌دهد اما مشکلی دارد و مشکل آن این است که دستورات بهینه نیستند. مثلا فرض کنید که عدد اولی از دومی بزرگتر است و پیغام Num1>Num2 نمایش داده می‌شود و بدیهی است که لازم نیست سایر دستورات IF که در سطرهای بعدی نوشته شده است بررسی شوند، اما هر IF به صورت مستقلی نوشته شده است و از نتیجه دستور قبلی آگاه نیست، تمامی IF ها اجرا خواهند شد.

البته در نظر داشته باشید که در این برنامه ساده واقعا بهینه بودن و سرعت اصلا مهم نیست اما در سایر جاها که ممکن است شما هزاران محاسبه برای بررسی یک شرط داشته باشد، حتی یک IF اضافه هم مهم خواهد شد.

مثال ۱۰ دستور IF - جایزه سیب

برنامه‌ای بنویسید که از کاربر سوال کند که چند تا سیب می‌خواهد خریداری کند، اگر تعداد سیب‌ها بیش از ۱۰ عدد بود به او ۲ سیب جایزه بدهیم و اگر بیش از ۲۰ عدد بود به او ۵ سیب و اگر بیش از ۱۰۰ عدد بود، به او ۱۲ سیب جایزه بدهید.

```
Sub Chapter4_Ex16()

Dim appleQty As Long
appleQty = InputBox("How many Apple do you Want?")

If appleQty > 100 Then
    Debug.Print "Bounes = 12"
ElseIf appleQty > 20 Then
    Debug.Print "Bounes = 5"
ElseIf appleQty > 10 Then
    Debug.Print "Bounes = 2"
Else
    Debug.Print "Bounes = 0"
End If

End Sub
```

نکته ۱: با من موافق هستید که با توجه به صورت مساله اگر بخواهیم ۲ عدد سیب جایزه بدهیم باید تعداد سیبی که فرد خریده است بین ۱۰ تا ۲۰ باشد و اگر خاطراتان باشد گفتیم که برای بررسی شرط هایی که عدد بین دو مقدار باید باشد، باید از And برای این استفاده کنیم مثلا بنویسیم:

```
IF appleQty > 10 And appleQty <20 Then
```

برای اینکه کار کمی کمتر شود و برنامه خلوت تر معمولا اینگونه IF را از آخر به اول می‌نویسند تا نیازی به And نباشد. به همین دلیل ما برنامه را از شرط آخر یعنی ۱۰۰ سیب شروع کردیم.

تمرین: این برنامه را با کلید F8 چندین بار اجرا و مقادیر مختلفی را وارد کنید.

سوال: اگر دقیقا تعداد سیب ها ۲۰ عدد باشد، چند سیب به عنوان جایزه نمایش داده می‌شود؟

سوال: برنامه را طوری تغییر دهید که اگر کاربری به اشتباه عدد منفی یا صفر وارد کرد، یک پیغام خطا به او نمایش داده شود و سپس مجددا از او تعداد سیب ها پرسیده شود.

سوال: اگر کاربر هنگام تایپ تعداد سیب‌ها عددی را وارد نکند و پنجره را ببندد چه اتفاقی می‌افتد؟ برای یافتن راهکار حل مشکلی که ایجاد شده است از گوگل کمک بگیرید.

IF های تو در تو

احتمالا تجربه نوشتن یک IF در داخل یک دستور IF دیگر را در اکسل داشته‌اید و البته در VBA اینکار نیز امکان پذیر است. دقت داشته باشید که می‌توانیم ما چندین لایه IF در داخل یک If داشته باشیم اما معمولا چون درک و خطایابی برنامه پیچیده می‌شود، بیش از ۲ مرحله اینکار را نمی‌کنیم.

```
If Condition Then
      If Condition Then Statement Else Statement
Else
      If Condition Then Statement Else Statement
End If
```

آشنایی با Boolean

شاید بهتر بود این بحث را در قسمت DataType ها مطرح می‌کردیم و یا در ابتدای شروع IF . اما آنرا تا اینجا به تاخیر انداختیم که در واقع شما عملا با مفهوم آن آشنا شوید و سپس بحث عملی و تئوریک آنرا انجام دهیم.

همانطور که تا اینجا دیدید ما با مفهوم True و False خیلی سروکار داریم و به همین دلیل یک نوع DataType ویژه برای آنها ساخته شده است که به آن بولی یا به انگلیسی Boolean (بولی‌یَن) گفته می‌شود.

بنابراین ما می‌توانیم متغیرهایی داشته باشیم که از نوع Boolean هستند و بنویسیم:

```
Dim x as Boolean
```

و آنها را مقدار دهی کنیم:

```
x = True
```

و آنکه یک عبارت Boolean بنویسیم:

```
x = (2 > 5)
```

منظور ما از عبارت Boolean یعنی چیزی که مقدار آن True یا False شود و در درس IF به آنرا با واژه Condition به شما معرفی کردیم. در واقع ما شکل کلی IF را اینگونه نوشتیم:

```
IF condition Then  statement_1 Else statement_2
```

و حال که به شما Boolean را معرفی کردم، شاید علمی‌تر آن باشد که بنویسیم :

```
IF BooleanExpression Then  statement_1 Else statement_2
```

یعنی در واقع ما در قسمت Condition (قسمت شرط دستور) انتظار یک عبارت Boolean داریم که مقدار آن True یا False باشد.

و حال اگر به شما بگویم که عملگرهای ...,AND,OR,Not ، عمگرهای Boolean هستند، منظورم را متوجه خواهید شد. یعنی انتظار داریم این عملگرها روی عبارت‌ها و یا مقادیری از نوع Boolean عملی را انجام دهید و در نهایت نتیجه آنها نیز یک عبارت Boolean خواهد شد.

نکته ۱: از قبل میدانیم که عملگرها نسبت به یکدیگر تقدم دارند مثلا در عبارت 4 / 2 + 2 ابتدا تقسیم انجام میشود و سپس جمع اتفاق می افتد و در اینجا باید یاد بگیریم که علمگرهای منطقی هم نسبت به هم تقدم دارند مثلا در عبارت x Or y And z نتیجه کاملا وابسته به این است که آیا ابتدا Or انجام میشود و یا اینکه And بنابراین من پیشنهاد میکنم که از علامت پرانتز استفاده کنید تا هم نیازی به حفظ کردن عملگرها نباشد و هم اینکه کد شما خواناتر شود.

تابع IIF

ما در VBA یک تابع هم برای نوشتن IF های کوتاه داریم به نام IIF . این تابع دقیقا مشابه تابع IF در اکسل است بدون هیچ تفاوتی.

```
IIF(Condition, statement_1, statement_2)
```

به عنوان مثال اگر کاربر عدد 0 را وارد کرد، پیغام صفر و در غیر اینصورت پیام غیر صفر نمایش داده می‌شود:

```
Sub Chapter4_Ex23()

    Num = InputBox("Enter a Number.")
    MsgBox IIf(Num = 0, "Zero", "Non Zero")

End Sub
```

همانطور که مشاهده میکنید از خروجی تابع به عنوان پیام MsgBox استفاده میکنیم.

نکته ۱: میتوانیم همانند Excel، ما در داخل IIF یک IIF دیگر هم بنویسیم.

نکته ۲: می‌توانیم از عملگرهای Boolean مانند And , Or در قسمت شرط/ Condition استفاده کنیم.

دستور Select Case

در همین ابتدا بگویم که کار دستور Select Case را کاملا می‌توانیم با IFهای بلوکی انجام دهیم و دقیقا کارش شبیه دستور IF است. حال اگر بپرسید که چه مزیت یا فایده‌ای دارد، به شما پاسخ می‌دهم استفاده از این دستور به خوانایی کد و تمیز نوشتن برنامه کمک می‌کند.

در واقع دستور IF بلوکی، دستور شلوغی است و اگر همان را با Select Case بنویسیم، خواهید دید که چقدر ساده است.

بگذارید دستور را با یک مثال شروع کنم و در انتها شکل کلی آنرا برای شما خواهم نوشت.

مثال ۲ دستور Select Case – فصل سال

برنامه‌ای بنویسید که از کاربر یک عدد از 0 تا 4 بگیرد و بگوید که آن عدد معادل کدام فصل سال است. اجازه دهید که این برنامه را ابتدا با دستور IF به شکل زیر بنویسیم.

```
Sub Chapter4_Ex17()

Dim Num As Long
Num = InputBox("Enter a Number between 1 to 4")

If Num = 1 Then
    Debug.Print "Bahar"
ElseIf Num = 2 Then
    Debug.Print "Tabestan"
ElseIf Num = 3 Then
    Debug.Print "Paeiz"
ElseIf Num = 4 Then
    Debug.Print "Zemestan"
Else
    Debug.Print "Invalid Number"
End If

End Sub
```

و حال همین برنامه با Select Case خواهیم نوشت.

```
Sub Chapter4_Ex18()

Dim Num As Long
Num = InputBox("Enter a Number between 1 to 4")

Select Case Num
    Case 1
        Debug.Print "Bahar"
    Case 2
        Debug.Print "Tabestan"
    Case 3
        Debug.Print "Paeiz"
    Case 4
        Debug.Print "Zemestan"
    Case Else
        Debug.Print "Invalid Number"
End Select

End Sub
```

و اگر از تکنیک یک خطی کردن دستورها استفاده کنید خواهیم داشت:

```
Sub Chapter4_Ex19()

Dim Num As Long
Num = InputBox("Enter a Number between 1 to 4")

Select Case Num
    Case 1: Debug.Print "Bahar"
    Case 2: Debug.Print "Tabestan"
    Case 3: Debug.Print "Paeiz"
    Case 4: Debug.Print "Zemestan"
    Case Else: Debug.Print "Invalid Number"
End Select

End Sub
```

امیدوارم که قانع شده باشیدکه آخرین کد به مراتب خواناتر و مرتب‌تر از کد اولی است. در واقع ما وقتی از Select Case استفاده می‌کنیم که می‌خواهیم در بین چند حالت (مثلا عدد ۱ تا ۴) انتخابی داشته باشیم و بر اساس آن انتخاب دستوراتی را بنویسیم.

همانطور که در مثال می‌بینید، در ابتدای دستور نام متغیر را یکبار نوشته‌ایم (Num) و در سایر خط‌ها مقادیری که آن متغیر ممکن است داشته باشد را نوشته‌ایم (Case 1) و به ازای هر یک مقدار، یک دستور نیز داریم .

شکل کلی دستور Select Case اینگونه است:

```
Select Case testexpression
      [Case expressionlist-n
            [instructions-n]]
      [Case Else
            [default_instructions]]
End Select
```

بله حق باشماست، از این شکل کلی خیلی چیزی دستگیرمان نمی‌شود بنابراین بیایید با هم قدم به قدم مثالی که برای فصل‌های سال زدیم را، مجددا بنویسیم.

متغیری به نام Num داریم که قرار است بر اساس مقدارش، پیغامی را چاپ کنیم.

بنابراین در قدم اول خواهیم نوشت بر اساس متغیر Num یکی از موارد (کیس‌ها) را انتخاب کن:

```
Select Case Num
```

و حال باید Case ها را بنویسیم:

```
Case 1
```

دستور Case 1 در حقیقت دارد مقدار Num را بررسی می کند که آیا برابر عدد ۱ است و اگر این شرط برقرار بود، دستوری را که در ادامه نوشتیم اجرا می‌کند و کلمه Bahar چاپ خواهد شد و دستور Select Case به پایان خواهد رسید و سایر Case ها بررسی نمی‌شود.

```
Debug.Print "Bahar"
```

حال سایر Caseها را هم می‌نویسیم که اگر متغیر Num با یکی از آنها برابر باشد، چه دستوری اجرا شود:

```
    Case 2
        Debug.Print "Tabestan"
    Case 3
        Debug.Print "Paeiz"
    Case 4
        Debug.Print "Zemestan"
```

و اگر هیچ مقدار Num با هیچ یک از مقادیری که در Case ها نوشته بودیم برابر نبود، قسمت Case Else اجرا خواهد شد:

```
    Case Else
        Debug.Print "Invalid Number"
```

در ضمن آنکه همانطور که قبلا گفتیم اگر دستوری چند سطری (بلوکی) بود، حتما باید انتهای آن دستور مشخص شود و انتهای دستور Select Case با End Select مشخص می‌شود:

```
End Select
```

تذکر: ما در IF می‌نوشتیم Num = 1 یعنی باید نام متغیر را می‌نوشتیم اما در دستور Case فقط می‌نویسیم Case 1 و خود VBA می‌داند که باید متغیر Num را با عدد ۱ مقایسه کند و توجه داشته باشید که کاربران تازه کار معمولا این نکته را فراموش می‌کنند و به اشتباه کدی مانند زیر را می‌نویسند که البته کاملا غلط است و برنامه خطا خواهد داد:

```
Select Case Num
    Case Num = 1
        Debug.Print "Bahar"
    Case Num = 2
        Debug.Print "Tabestan
```

مثال ۲ دستور Select Case – جایزه سیب

می‌خواهیم همان برنامه محاسبه تعداد جایزه سیب را Select Case بنویسیم.

برنامه‌ای بنویسید که از کاربر سوال کند که چند تا سیب می‌خواهد خریداری کند، اگر تعداد سیب‌ها بیش از ۱۰ عدد بود به او ۲ سیب جایزه بدهیم و اگر بیش از ۲۰ عدد بود به او ۵ سیب و اگر بیش از ۱۰۰ عدد بود، به او ۱۲ سیب جایزه بدهید.

```
Sub Chapter4_Ex21()
    Dim appleQty As Long
    appleQty = InputBox("How many Apple do you Want?")

Select Case appleQty
    Case Is > 100:   Debug.Print "Bounes is 12"
    Case Is > 20:    Debug.Print "Bounes is 5"
    Case Is > 10:    Debug.Print "Bounes is 2"
    Case Else:       Debug.Print "Bounes is 0"
End Select

End Sub
```

شرح برنامه: توسط علامت «:» دستورات را در یک سطر نوشتیم تا برنامه خواناتر شود و همانطور که می‌بینید می‌توانیم شرط بزرگتر را اینگونه پیاده کنیم.

انواع شرط‌های Select Case

فرض کنید که متغیری به نام x داریم که از از نوع Long است (یعنی یک عدد است) و می‌خواهیم انواع حالت هایی که برای قسمت شرط Case می‌توانیم استفاده کنیم را بررسی کنیم:

۱- اگر بخواهیم بررسی کنیم که مقدار متغیر x بین دو عدد ۱۰ و ۲۰ است از To استفاده می‌کنیم:

```
Case 1 To 5
```

۲- اگر بخواهیم که x بیشتر از ۱۰۰ باشد:

```
Case Is > 100
```

تذکر ۱: اگر Is را ننویسید، خود VBE آنرا اضافه می‌کند و اگر بخواهید بدانید که Is چه نقشی دارد، به شما خواهم گفت آن نیز یک Operator است یعنی یک عملگر مانند سایر عملگرها که کارش بررسی شرط‌ها است.

۳- اگر بخواهیم که بررسی کنیم x یکی از سه مقدار ۱۸ یا ۲۰ یا ۲۲ باشد:

```
Case 18, 20, 22
```

تذکر: نباید از Or استفاده کنید وعلامت «,» در اینجا کار Or را انجام می‌دهد.

۴- ترکیب شرط‌ها هم با یکدیگر صادق است مثلا می‌خواهیم بررسی کنیم که آیا x کمتر یا مساوی صف است و یا یکی از دو مقدار ۱۷ یا ۲۱ است و یا بین ۵۰ تا ۶۰ است:

```
Case Is <= 0, 17, 21, 50 To 60
```

و در نهایت می‌توان کدی شبیه زیر را داشته باشیم:

```
Sub Chapter4_Ex22()
Dim x As Long
x = InputBox("Enter a Number")

    Select Case x
        Case 1 To 5
            Debug.Print "x Between 1 to 5"
        Case Is > 100
            Debug.Print " x > 100"
        Case 18, 20, 22
            Debug.Print " x is 18 or 20 or 22"
        Case Is <= 0, 17, 21, 50 To 60
            Debug.Print " Multi Expresstion, Wow!"
    End Select

End Sub
```

تذکر ۲: اگر شرطی یکی از Case ها مقدار True شود (یعنی شرط برقرار باشد)، سایر Case ها که بعد از آن آمده‌اند بررسی نخواند شد.

تذکر ۳: برنامه نویسان حرفه‌ای یک عادت بسیار خوبی در نوشتن دستورات بلوکی (یا چند سطری) دارند و آن این است که اگر می‌خواهند یک Select Case بنویسید، ابتدا اول و آخر دستور را تایپ می‌کنند و سپس بدنه داخلی دستور را تکمیل می‌کنند.

```
Select Case x

End Select
```

این تکنیک به آنها کمک می‌کند تا فراموش نکنند که پایان دستور کجاست.

فصل پنجم – حلقه‌ها

در این فصل میخواهیم کار با دستورات تکرار را فرا بگیریم. قبل از شروع باید بگویم واژه انگلیسی عنوان این فصل Iteration Statement می‌باشد. مثلا فرض کنید که می‌خواهیم فرمان Debug.Print "Hi" را هزاران بار اجرا کنیم و آنوقت است که از دستورات Iteration کمک میگیریم.

واژه Iterations را به فارسی میتوانیم به «تکرار» و یا «حلقه‌ها» ترجمه کنیم که ما از هر دو ترجمه در متن این کتاب استفاده خواهیم کرد.

در ضمن این که می‌خواهیم تاکید ویژه‌ای بر روی این فصل کنم زیرا اساس اکثر برنامه‌هایی که شما می‌نویسید بر تکرارها است. مثلا می‌خواهید هزاران کالا را بررسی کنید و در نهایت چند تا از آن‌ها را که مشخصات خاصی دارند را استخراج نمایید بنابراین باید برای همه کالاها دستورات بررسی مشخصات را تکرار کنید.

نکته : در برخی از منابع شما به جای واژه Iteration ، واژه Loop را مشاهده می‌کنید که کاملا صحیح است و ما Loop را «تکرار» و یا «حلقه» ترجمه می‌کنیم.

ایجاد تکرار با GoTo- روشی اشتباه

فرض کنید که می‌خواهیم به تمامی افرادی که در کره زمین تا به حال زندگی کرده‌اند سلام بدهیم. بنابراین مجبوریم که فرمان Debug.Print "Hi" را بارها و بارها اجرا کنیم . در واقع چون نمیدانیم که چند نفر تا به حال زندگی کرده اند بنابراین به تعداد بی شماری باید این دستور اجرا شود.

در ضمن اینکه کامپیوتر اصلا از به تک تک افراد سلام کند اصلا خسته نمی‌شود و تا زمانی که به برق وصل است اینکار را انجام می‌دهد.

حالا اگر از شما سوال کنیم که آیا می‌توانید با دستوراتی که تاکنون در این کتاب آموخته‌اید، این کار را انجام دهید. یعنی بارها دستور Debug.Print "Hi" را اجرا کنید؟

بله می‌توانید! با دستور GoTo این کار به شکل زیر قابل انجام است:

```
Sub Chapter5_Ex1()

10:
    Debug.Print "Hi"
    GoTo 10

End Sub
```

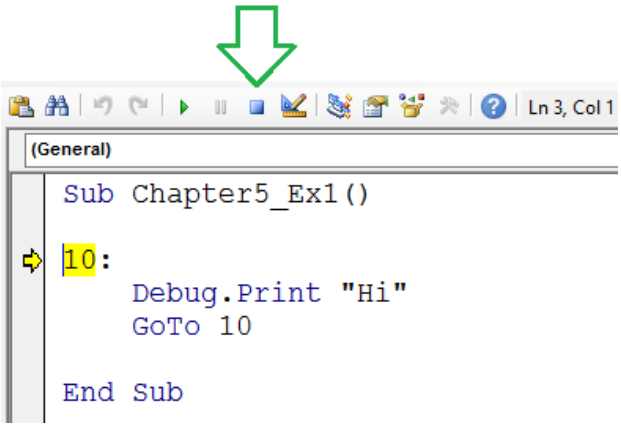
در واقع دستور GoTo 10 باعث می‌شود که برنامه مجددا به سطر شماره 10 برگردد و با این روش بارها و بارها دستور Debug.Print "Hi" اجرا می‌شود.

نکته ۱: اگر این برنامه را اجرا کنید، اکسل شما از کار خواهد افتاد یا اصطلاحا «هنگ» خواهد کرد و شما خواهید دید که اکسل در حالت Not Responding می‌رود. زیرا نمی‌تواند از پروسیجری که شما اجر کرده‌اید خارج شود و وظایف عادی خودش را انجام دهد.

هنگامی که برنامه‌ای باعث چنین حالتی می‌شود، یعنی کاری را بارها و بارها بدون داشتن یک پایان انجام می‌دهد، اصطلاحا می‌گوییم که «برنامه در Loop افتاده است».

نکته ۲: بهتر است که برنامه فوق را برای اینکه یک Loop بی نهایت ایجاد نشود، با کلید F8 اجرا کنید تا هر وقت که خواستید بتوانید با زدن دکمه Reset به کار برنامه خاتمه دهید.

تصویر دکمه Reset برای اتمام کار یک برنامه در حال اجرا



نکته ۳: اگر کلید F5 را بزنید و برنامه در Loop قرار گیرد، احتمالا بتوانید با زدن کلید CTRL+Break از برنامه خارج شوید. البته باید از این کلید خیلی سریع استفاده کنید تا اکسل در حالت هنگ یا همان Not Responding قرار نگیرید.

حال اگر بخواهیم که تکرار برنامه بی‌شمار نباشد، بلکه فقط 10 بار این پیام چاپ شود، آنوقت می‌توانید با تکنیک زیر این کار را انجام دهید:

```
Sub Chapter5_Ex2()

Dim x As Long

10:
    Debug.Print "Hi"
    x = x + 1
    If x < 10 Then GoTo 10

End Sub
```

ما در این برنامه از دستور x=x+1 استفاده کرده‌ایم و این دستور را حتما شما باید دقیقا متوجه شوید. زیرا این تکنیک در جاهای مختلفی کاربرد دارد.

به همین دلیل در ادامه به صورت ویژه‌ای این دستور را بررسی خواهیم کرد.

تفسیر دستور x = x +1

در ابتدا باید یک نکته را یادآوری کنم. اگر خاطرتان باشد ما در فصل «آشنایی با دنیای VBA» ،عنوانی داشتیم به نام « تخصیص یک مقدار به یک متغیر» . در آنجا به شما گفتم که برای تفسیر علامت «=» که به آن Assignment Oprator گفتیم، ابتدا سمت راست مساوی را محاسبه می‌کنیم و سپس نتیجه را در متغیری که سمت چپ علامت مساوی است، قرار می‌دادیم.

دقیقا برای تفسیر دستور x = x +1 نیز باید همین کار را انجام دهیم. یعنی ابتدا باید سمت راست مساوی را محاسبه کنید و سپس نتیجه محاسبه هر مقداری شد، آنرا مقدار در داخل x قرار دهیم.

پروسیجر Chapter5_Ex2 را در نظر بگیرید. در ابتدا با دستور زیر یک متغیر را از نوع Long (یعنی عدد صحیح) تعریف کرده‌ایم. هر گاه یک متغیر تعریف می‌شود، مقدار به صورت اتوماتیک به عنوان مقدار پیش فرض برای آن نیز در نظر گرفته می‌شود و پیش فرض متغیرهای عددی ، مقدار صفر است. حال اگر دستور زیر اجرا شود، نتیجه آن ایجاد یک متغیر با مقدار 0 در حافظه دستگاه به نام x خواهد بود.

```
Dim x As Long
```

بقیه دستورات پروسیجر سطر به سطر اجرا خواهند شد تا اینکه برسیم به دستور x = x +1 و حال باید ببینم که VBA این دستور را چگونه تفسیر می‌کند .

قبل از آنکه بخواهم تفسیر دستور x = x +1 را به شما توضیح دهیم بگذارید یک نکته را هم بررسی کنیم. اگر شما از یک نفر که ریاضی بلد است بپرسید که x = x +1 چیست و آنرا چگونه می‌فهمد، او به شما خواهد پاسخ می‌دهد که این یک نامعادله است. یعنی اصلا این معادله در دنیا امکان پذیر نیست. زیرا دستور x = x +1 را اگر به صورت ریاضی تفسیر کنیم ، نتیجه آن می‌شود 0 = 1 و این نیز امکان پذیر نیست. (امیدوارم که از دبیرستان یادتان باشد که چطور از x = x + 1 نتیجه می‌شود 0 = 1 و اگر خاطرتان نیست ، لطفا از یکی از دوستانتان کمک بگیرید.)

خواستم تاکید مجددی کنم که در VBA دستور x = x +1 تفسیر خاص خودش را دارد و نباید آنرا به صورت یک عبارت ریاضی تفسیر کنیم.

اگر VBA بخواهد x = x +1 را تفسیر کند ، همانطور که گفتیم ابتدا سمت راست علامت مساوی یعنی x + 1 محاسبه می‌شود و برای محاسبه آن VBA به حافظه کامپیوتر مراجعه می‌کند و به دنبال خانه‌ای به نام x می‌گردد و سپس نگاه می‌کند که در آن خانه چه عددی ذخیره شده است.

گفتیم که مقدار پیش فرض بعد از تعریف یک متغیر صفر است، بنابراین هنگامی که VBA به خانه x مراجعه می‌کند مقدار 0 را در آنجا خواهد دید و در نتیجه به جای مقدار x در عبارت x + 1 ، عدد صفر را قرار می‌هد و 1 + 0 را محاسبه می‌کند و نتیجه این جمع هم که کاملا مشخص است و مقدار 1 خواهد شد.

بنابراین سمت راست علامت مساوی عدد 1 خواهد شد و حال که تکلیف سمت راست مشخص شده است، می‌توانیم به جای x + 1 مقدارش را قرار دهیم و عبارت x = x + 1 خواهد شد x = 1.

عبارت x =1 را قبلا آموخته‌ایم و این یعنی مقدار 1 را به متغیر x اختصاص می‌دهیم و در حافظه دستگاه پس از اجرای این دستور مقدار خانه x عدد 1 خواهد بود.

حال نوبت دستور IF می‌رسد و از آنجایی که مقدار x برابر عدد 1 است، قسمت شرط True خواهد شد و دستور GoTo 10 اجرا می‌شود و برنامه به سطر 10 می‌رود.

سپس دستورات از سطر 10 و سطر به سطر اجرا خواهند شد تا برسیم به دستور x = x +1 .

این دستور همانند قبل تفسیر می‌شود یعنی VBA به حافظه دستگاه می‌رود تا خانه x را بیابد و مقدار آنرا ببیند. اینبار که VBA به خانه x مراجه می‌کند مقدار 1 را در آنجا می‌بیند، و بنابراین نتیجه دستور x+1 عدد 2 خواهد شد و در نهایت x = 2 می‌شود و سایر سطرها اجرا خواهند شد و با رسیدن به دستور IF ، مجددا برنامه به سطر 10 برخواهد گشت.

در واقع هر بار که برنامه به سطر 10 بر می‌گردد و سطر x = x +1 اجرا می‌شود، مقداری که در خانه x در حافظه قرار دارد 1 واحد افزایش پیدا می‌کند.

به این تکنیک یعنی استفاده از عبارت x = x +1 که باعث افزایش x به مقدار 1 واحد می‌شود اصطلاحا Incremental Assignment می‌گویند.

شاید بتوانیم آنرا به "تخصیص افزایشی" ترجمه کنیم که مطمئن باشید اینکار را نخواهیم کرد ، چون این ترجمه هیچ مفهومی در ذهن شما ندارد و ما به جای ترجمه از همان اصطلاح علمی و انگلیسی آن استفاده خواهیم کرد.

نکته ۱: بدیهی است که می‌تواند نام متغیر شما به جای x هر چیزی دیگری باشد. مثلا عبارت زیر باعث افزایش متغیر Counter به اندازه یک واحد می‌شود:

```
counter = counter + 1
```

نکته ۲: از تکنیک Assignment فقط منحصر به جمع نیست بلکه اگر عبارت y = y -1 را بنویسیم، این به معنای آن است که از متغیر y یک واحد کم کن و یا w = w * 2 یعنی متغیر w را به توان ۲ برسان.

نکته ۳: ما می‌توانیم z = z + 4 را بنویسیم و این به معنای اضافه شدن z به مقدار 4 واحد است.

```
Sub Chapter5_Ex2_I()

Dim x As Long: Dim y As Long: Dim z As Long
Debug.Print "x", "y", "z"
Debug.Print "----------------------------------"

10:
    Debug.Print x, y, z
    x = x + 1
    y = y - 1
    z = z + 4

    If x < 10 Then GoTo 10

End Sub
```

نکته ۲: متغیر ما می‌تواند از نوع «متنی» باشد و استفاده از این تکنیک (یعنی علامت مساوی) برای متغیرهای متنی هم متداول است . البته ما متغیرهای متنی را با هم جمع/ تفریق و ... نمی‌کنیم بلکه آنها را به یکدیگر می‌چسبانیم.

```
Sub Chapter5_Ex2_II()

Dim s As String
s = "Hi, Farshid."
s = s & "How Are You?"
Rem baray Enter (khat jadid) az Chr(13) estefadeh mikonim
s = s & Chr(13)
s = s & "I know that you Love Programming."
s = s & Chr(13)
s = s & " ;)"

Debug.Print s

End Sub
```

مشاهده مقدار متغیر با ToolTip

برای درک بهتر مقادیر x که در مثال قبلی دیدم، به شما توصیه می‌کنم که با کلید F8 برنامه را سطر به سطر اجرا کنید.

حال در این حالت (حالت Debug) هرگاه که شما موس را بر روی x نگه دارید، مقدار متغیر x در یک ToolTip به شما نمایش داده می‌شود و می‌توانیم بفهمیم که مقدار x چیست.

اینکار به شما کمک می‌کند تا درک بهتری نسبت به روند اجرای برنامه داشته باشید.

تصویر استفاده از ToolTip برای دیدن مقدار x

```
10:
    Debug.Print "Hi"
    x = x + 1
    x = 3

    If x < 10 Then GoTo 10
```

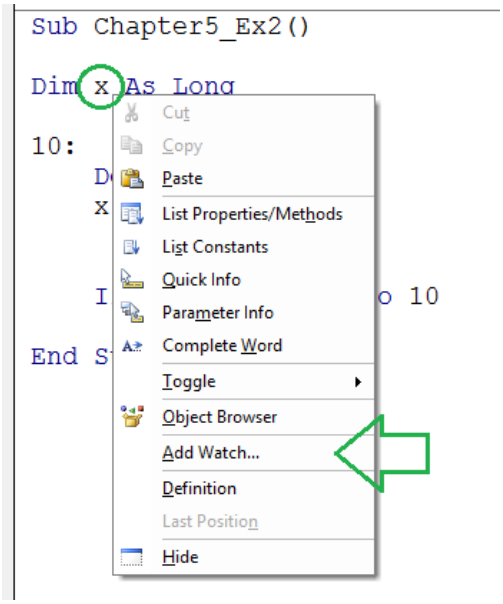
نکته: به پیام‌های کوچکی و زرد رنگی که با نگاه داشتن موس بروی یک چیز، موقتا ظاهر می‌شوند ToolTip می‌گوییم.

مشاهده مقدار متغیر با Watch

در مثال قبلی ما فقط یک متغیر داشتیم و برای مشاهده مقدار آن کافی بود که موس را یک لحظه روی نام متغیر نگاه داریم و مقدارش در یک ToolTip به ما نمایش داده می‌شد. حال در برنامه‌های واقعی ممکن است چندین متغیر داشته باشیم ما به عنوان برنامه نویس بخواهیم که آنها ار تحت نظر بگیریم و از تغییرات آنها باخبر شویم با ابزار Watch می‌توانیم اینکار را انجام دهیم.

برای گذاشتن Watch بر روی یک متغیر R-Click کنید و سپس گزینه Add Watch را انتخاب کنید.

تصویر قرار دادن Watch بر روی متغیر x

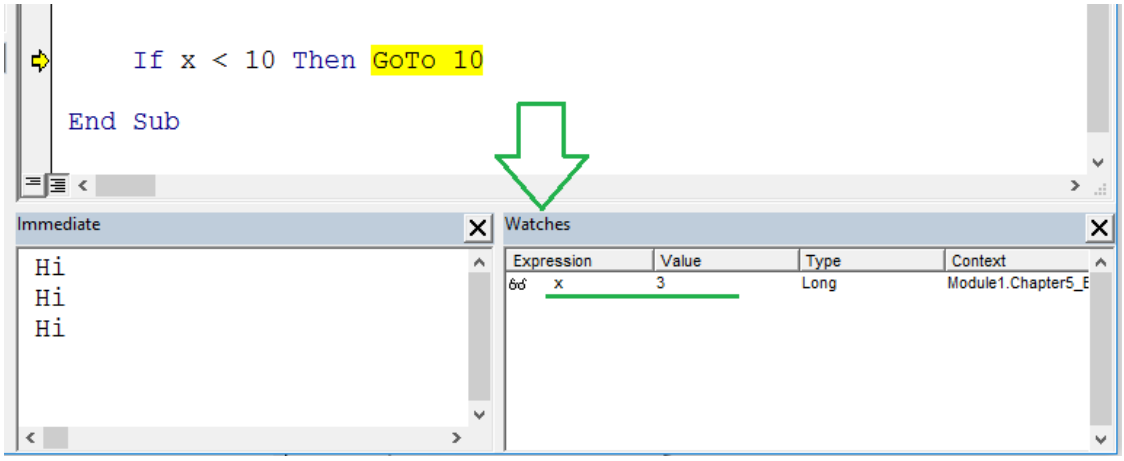


سپس یک پنجره باز خواهد شد که با انتخاب Ok ، برای شما یک Watch بر روی مقدار x خواهد گذاشت و از این پس می‌توانید در قسمت Watch از مقدار x مطلع شود.

حال اگر برنامه را در حالت Debug (یعنی سطر به سطر با کلید F8) اجرا کنید، می‌توانید متغیرها را تحت نظر بگیرد.

نکته ۱: قسمت Watch باید در کنار قسمت Immediate نمایش داده شود، اگر نمایش داده نشد می‌توانید از View → Watch Window را انتخاب کنید.

تصویر قسمت Watch و مقدار x



نکته ۲: حتما برای استفاده از Watch باید برنامه را مرحله به مرحله اجرا کنید مثلا از کلید F8 استفاده کنید و اگر کلید F5 را بزنید پروسیجر به یکبار با سرعت تا انتها اجرا خواهد شد و قسمت Watch چیزی را به شما نمایش نمی‌دهد.

چرا استفاده از GoTo اشتباه است

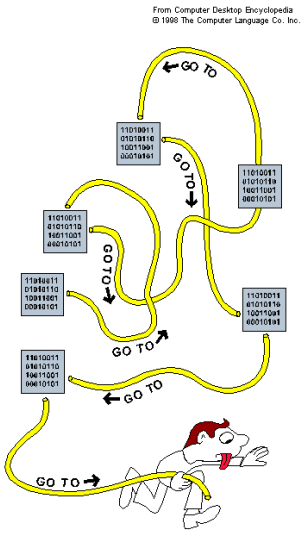
روش‌های برنامه نویسی متعددی در دنیا وجود دارد و یکی از آنها Structured programming است که در فارسی آنرا به «برنامه نویسی ساخت یافته» ترجمه کرده‌اند.

اساس این روش بر خوانا بودن کد است. در این روش برنامه به قسمت‌هایی کوچکی تقسیم می‌شود که به آنها پروسیجر می‌گوییم (در واقع برنامه به روتین ها و سپس رویتن ها به پروسیجر ها تقسیم می شوند) و هر پروسیجر یک کار کوچک و واضح را انجام می‌دهد.

یکی از قواعد «برنامه نویسی ساخت یافته» آن است که از GoTo استفاده نشود و یا بسیار کم استفاده شود. فرض کنید که یک برنامه صد خطی دارید که در آن از ۲۰ دستور GoTo استفاده شده است، حال اگر بخواهید مراحل اجرای برنامه را در ذهن خود تصور کنید، یک کلاف خواهید دید مانند ماکارانی و اصطلاحا «اسپاگتی کد» نامیده می‌شود.

استفاده از GoTo های فراوان درک روند اجرای برنامه را بسیار سخت خواهد کرد و به همین دلیل استفاده از GoTo برای ساختن حلقه، روشی اشتباه است.

تصویر اسپاگتی کد به دلیل استفاده از GoTo



ما در VBA برای ساختن تکرارها/حلقه‌ها بدون اینکه نیازی به استفاده از GoTo داشته باشیم دو دستور زیر را داریم که قرار است در ادامه این دستورات را با دقت و حوصله فراوانی یاد بگیریم:

- دستور For – Next برای ایجاد حلقه‌ها با تعداد تکرار مشخص
- دستور Do – Loop برای ایجاد حلقه‌ها تا هنگامی که شرایط مساعد است!

نکته ۱: این دو دستور به صورت بلوکی استفاده می‌شوند و اگر خاطرتان باشد هر دستور بلوکی یک ابتدا و یک انتها دارد. معمولا برنامه نویسان تازه کار فراموش می‌کنند که باید پایان بلوک دستورات را مشخص کنند.

نکته ۲: قبل از اینکه این عنوان را به اتمام برسانم ، لازم است که یک نکته دیگر را در خصوص «برنامه نویسی ساخت یافته» به شما بگویم و آن این است که معمولا در این نوع برنامه نویسی، کد ما در یک نقطه مشخص به پایان می‌رسد. یعنی اگر برنامه‌ای می‌نویسیم که ۱۰ سطر دارد، در آخرین سطر دستور End Sub است و همانجا پایان برنامه است و نباید در بدنه برنامه (مثلا سطر پنجم) شرطی داشته باشید که باعث شود برنامه بتواند در آنجا نیز به پایان برسد.

دستور For – Next

این دستور برای ایجاد یک حلقه با تعداد دفعات تکرار مشخص استفاده می‌شود. مثلا اگر بخواهیم که ۱۰۰ بار دستور Debug.Print "Hi" را اجرا کنیم ، چون مشخص است که دقیقا ۱۰۰ بار باید این دستور اجرا شود بنابراین باید از For – Next استفاده شود.

دستور For – Next برای اجرا شدن نیاز دارد که یک متغیر به آن بدهیم و سپس به For – Next بگوییم که این متغیر را از چه عددی تا چه عددی مقدار دهی کند.

مثلا اگر بخواهیم که یک حقله با تکرار ۱۰۰ بار بسازیم ، باید اینگونه بنویسیم:

```
For  x = 1  To  100
```

در ضمن اینکه چون دستور For – Next یک بلوک است، باید انتهای آنرا مشخص کنیم. بنابراین کل بلوک به شکل زیر می‌شود:

```
For  x = 1  To  100
Next
```

حال داخل این بلوک هر دستوری دوست دارید بنویسید :

```
For  x = 1  To  100
      Debug.Print "Hi"
Next
```

اگر این بلوک را در داخل یک پروسیجر بگذارید و سپس پروسیجر را اجرا کنید، نتیجه آن چاپ ۱۰۰ تا سلام خواهد بود:

```
Sub Chapter5_Ex3()

For x = 1 To 100
    Debug.Print "Hi"
Next

End Sub
```

نقش متغیر در دستور For – Next

ممکن است شما بپرسید که نقش آن x در دستور For – Next چیست؟ اصلا چرا دستور را اینگونه ننویسم:

```
For  1 To 100
      Statments
Next
```

یا شاید خلاصه تر هم پیشنهاد کنید:

```
For 100
      Statments
Next
```

قطعا سوال شما درست است، اگر قرار است کاری ۱۰۰ بار انجام شود، بهتر است که خود VBA آنرا مدیریت کند و از ما نخواهد که حتما یک متغیر را معرفی کنیم.

اما در پاسخ شما باید بگویم که در مثال قبلی ما هیچ نیازی به داشتن x نداشتیم اما همواره اینگونه نیست . بلکه در بسیاری از موارد ما می‌خواهیم که یک متغیر به نام x را از 1 تا 100 مقدار دهی کنیم، زیرا بر اساس آن متغیر می‌خواهیم محاسباتی را انجام دهیم.

به عنوان مثال به شما می‌گویند که جذر همه اعداد ۱ تا ۱۰۰ را چاپ کنید. حال در اینجا شما نیاز دارید که اعداد ۱ تا ۱۰۰ را داشته باشید تا بتوانید جذر آنها را بگیرید.

در اینجاست که نقش معرفی متغیر در For – Next دقیقا مشخص می‌شود. در واقع دستور For – Next از ما یک متغیر می‌گیرد و به ترتیب به آن مقادیری که را که تعیین کرده‌ایم را به ترتیب اختصاص می‌دهد.

هنگامی که یک For – Next به شکل زیر می‌سازیم،

```
For  x = 1  To  100
Next
```

ابتدا مقدار x برابر 1 خواهد شد و سپس هر آنچه در بدنه بلوک نوشته‌ایم اجرا می‌شوند تا به دستور Next برسیم. دستور Next دارد به VBA می‌گوید که حالا نوبت مقدار بعدی x رسیده است و مقدار بعدی x عدد 2 خواهد شد و سپس بدنه بلوک اجرا می‌شود تا مجددا به دستور Next برسیم و قاعدتا اجرا Next باعث می‌شود که مقدار x، عدد بعدی شود یعنی برابر 3 و همینطور الی آخر.

اتفاقا برای خوانایی برنامه خیلی بهتر است که بنویسیم Next x یعنی ما x بعدی را می‌خواهیم:

```
For  x = 1  To  100
Next x
```

مثال ۱ دستور For Next - جذر

برنامه‌ای بنویسید که اعداد یک تا صد را به همراه جذر آنها چاپ کند:

```
Sub Chapter5_Ex4()

Dim i As Long

For i = 1 To 100
    Debug.Print i, Sqr(i)
Next i

End Sub
```

شرح برنامه: یک متغیر به نام i از نوع عدد صحیح (Long) تعریف کرده‌ایم و سپس با دستور For – Next به این متغیر را از ۱ تا ۱۰۰ تغییر می‌دهیم.

در واقع بهتر است بگوییم که متغیر i ابتدا عدد 1 است و بدنه بلوک اجرا می‌شود بنابراین عدد 1 و جذر عدد 1 چاپ خواهند شد. سپس دستور Next i باعث می‌شود که مقدار متغیر i عدد 2 شود و در نتیجه عدد 2 به همراه جذرش چاپ می‌شود و همینطور تا اینکه مقدار i برابر 100 شود و بعد از اینکه عدد 100 و جذر آن چاپ شد، حلقه For-Next به کارش پایان می‌دهد زیرا i بعدی نداریم (در واقع قرار نیست که i برابر عدد 101 شود) و در ادامه سایر سطرهایی که بعد از Next نوشته شده‌اند اجرا خواهند شد.

تمرین: این برنامه را با کلید F8 و استفاده از Watchها اجرا کنید.

مثال ۲ دستور For Next – اعداد زوج

برنامه‌ای بنویسید که کلیه اعداد زوج بین یک تا صد را برای ما چاپ کند.

کافی است که با یک For – Next تمامی اعداد 1 تا 100 را تولید کنید و سپس با یک تابع IF باقی‌مانده تقسیم هر یک از آن اعداد را بر عدد 2 محاسبه کنیم و اگر باقی‌مانده 0 بود، یعنی آن عدد زوج است.

```
Sub Chapter5_Ex5()

Dim aNumber As Long

For aNumber = 1 To 100
    If aNumber Mod 2 = 0 Then Debug.Print aNumber
Next aNumber

End Sub
```

شرح برنامه: ما یک متغیر به نام aNumber از نوع عدد صحیح تعریف کرده‌ایم و سپس با عملگر Mod باقی‌مانده تقسیم آن عدد بر 2 را محاسبه کرده‌ایم و اگر عبارت aNumber Mod 2 = 0 مقدار True باشد، آنگاه آن عدد زوج است و ما آنرا چاپ می‌کنیم.

استفاده از Step در For – Next

در مثال قبلی خواستیم که اعداد زوج را چاپ کنیم و اینکار را با یک IF انجام دادیم. حال می‌خواهم به شما بگویم که دستور For – Next قابلیت دیگری را دارم و آنهم تنظیم Step است.

در حالت عادی دستور For – Next مقادیر متغیر را به اندازه 1 واحد در هربار اجرا زیاد می‌کند، یعنی در بار اول aNumber مقدار 1 است و سپس مقدار آن 2 خواهد شد.

حال می‌توانیم توسط Step تنظیم کنیم که مقادیر متغیر به اندازه 2 واحد زیاد شود مثلا دستور زیر را در نظر بگیرید:

```
Sub Chapter5_Ex6()

Dim aNumber As Long

For aNumber = 2 To 100 Step 2
    Debug.Print aNumber
Next aNumber

End Sub
```

متغیر aNumber در ابتدا مقدار 2 است و سپس 4 و همینطور الی آخر به مقدار 2 واحد در هربار اجرا زیاد خواهد شد.

اگر لازم است می‌توانید Step 0.5 را هم داشته باشیم و متغیر ما به اندازه پنج دهم در هر بار اجرا زیاد می‌شود.

```
Sub Chapter5_Ex7()

Dim aNumber As Double

For aNumber = 2 To 100 Step 0.5
    Debug.Print aNumber
Next aNumber

End Sub
```

نکته: متغیر را از نوع Double که اعشاری می‌پذیرد تعریف کرده‌ایم.

حال فرض کنید که می‌خواهیم از آخر به اول بیاییم، یعنی متغیری داشته باشیم که ابتدا 100 باشد و سپس 99 و بعد 98 و ... تا به عدد 0 برسیم، می‌توانیم در این حالت Step -1 هم داشته باشیم:

```
Sub Chapter5_Ex8()

Dim aNumber As Long

For aNumber = 100 To 0 Step -1
    Debug.Print aNumber
Next aNumber

End Sub
```

مثال ۳ دستور For Next – خروج از حلقه

برنامه‌ای بنویسید که از بین اعداد 1000 تا 2000 ، اولین عددی که بر 333 بخش پذیر است را پیدا کند.

```
Sub Chapter5_Ex9()

Dim counter As Long

For counter = 1000 To 2000
    If counter Mod 333 = 0 Then Exit For
Next counter

Debug.Print "First Number is :" & counter

End Sub
```

شرح برنامه: بازهم برای یافتن بخش‌پذیر بودن از عملگر Mod استفاده می‌کنیم و اگر باقی‌مانده 0 بود یعنی بخش پذیر است. اما نکته در این است که باید اولین عدد را بیابیم. در واقع همین که اولین عدد را یافتیم باید از For – Next خارج شویم و لازم نیست که حلقه را تا انتها ادامه دهیم و با این خارج شدن از اجرای بیهوده حلقه و در نتیجه اتلاف زمان جلوگیری خواهیم کرد. برای خارج شدن از یک حلقه For – Next از دستور Exit For استفاده کرده‌ایم.

مثال ۴ دستور For Next – جمع اعداد

تکنیکی که در این برنامه یاد می‌گیرید بسیار مهم و پر کاربرد است و مطمئن شوید که بر آن کاملا مسلط شده‌اید. شاید در اولین بار خواندن و اجرای این برنامه متوجه نحوه کار آن نشوید، نگران نباشید چندین بار آنرا آزمایش کنید و مطمئن باشید که چیزی پیچیده‌ای ندارد.

برنامه‌ای بنویسید که جمع اعداد 1 تا 100 را محاسبه کنید.

بدیهی است که ابتدا باید اعداد 1 تا 100 را با یک For – Next تولید کنیم و برای این منظور متغیر counter را تعریف می‌کنیم و از آنجایی که بالاخره جمع این اعداد باید در یک متغیری ذخیره شوند، متغیری به نام sum را هم خواهیم داشت:

```
Dim counter As Long
Dim sum As Long

For counter = 1 To 100

Next counter
```

اما مهم آن است که چگونه این اعداد را با هم جمع بزنیم! بازهم از همان علامت «=» استفاده می‌کنیم و عبارت زیرا را در بدنه For – Next می‌نویسیم:

```
sum = sum + counter
```

هنگامی که این برنامه اجرا شود و به پایان برسد، مقدار sum جمع اعداد یک تا صد خواهد شد.

حال بیایید برنامه را قدم به قدم از ابتدا در ذهنمان اجرا کنیم:

اولین بار که حلقه For - Next اجرا می‌شود:

- مقدار counter عدد 1 است و مقدار sum عدد صفر
- ابتدا سمت راست مساوی محاسبه می شود و مقدار sum + counter برابر 1 + 0 می-شود.
- سپس مقدار 1 + 0 که عدد 1 است در متغیر sum (سمت چپ مساوی) قرار داده می‌شود. (به عبارت دیگر می‌توانیم بگوییم که مقدار 1 به متغیر sum اختصاص داده می‌شود و در نتیجه مقداری که در متغیر sum ذخیره می‌شود عدد 1 خواهد بود)

دومین بار که حلقه For – Next اجرا می‌شود

- مقدار counter عدد 2 می شود.
- مقدار sum + counter برابر 2 + 1 خواهد شد . زیرا در مرحله قبل sum عدد 1 شده است.
- سپس مقدار 2+1 که عدد 3 است در متغیر sum (سمت چپ مساوی) قرار داده می‌شود.

سومین بار که حلقه For – Next اجرا می‌شود

- مقدار counter عدد 3 می شود.
- مقدار sum + counter برابر 3 + 3 خواهد شد . زیرا در مرحله قبل sum عدد 3 شده است.
- سپس مقدار 3+3 که عدد 6 است در متغیر sum (سمت چپ مساوی) قرار داده می‌شود.

چهارمین بار که حلقه For – Next اجرا می‌شود

- مقدار counter عدد 4 می شود.
- مقدار sum + counter برابر 4 + 6 خواهد شد . زیرا در مرحله قبل sum عدد 6 شده است.
- سپس مقدار 4+6 که عدد 10 است در متغیر sum (سمت چپ مساوی) قرار داده می‌شود.

و همینطور الی آخر. در واقع متغیر sum یک انبار است که در هر بار اجرا ، مقداری که آن انبار دارد را با counter جمع می‌زنیم و نتیجه هر چه شود را مجددا در همان انبار ذخیره می‌کنیم.

نکته ۱: امیدواریم که توانسته باشم این برنامه را خوب برای شما توضیح دهم و توصیه می‌کنیم که به کمک F8 برنامه را با حوصله سطر به سطر اجرا کنید و مقادیر متغیرها را ببینند:

```
Sub Chapter5_Ex10()

Dim counter As Long
Dim sum As Long

For counter = 1 To 100
    sum = sum + counter
Next counter

Debug.Print "Answer :" & sum

End Sub
```

نکته ۲: در این روشی که ما بکار بردیم ابتدا همه اعداد را تولید کردیم و سپس آن اعداد را با هم جمع زدیم و در نهایت به جوابی رسیدم. باید بدانید که ریاضیات برای ما تکنیک‌هایی را فراهم کرده است تا لازم نباشد که همه این اعداد را تولید کنیم و سپس با هم جمع بزنیم مثلا برای یافتن جمع اعداد 1 تا n می‌توانیم از فرمول ریاضی زیر استفاده کنیم و بدیهی است که این تکنیک بسیار سریعتر و بهینه تر از روشی است که ما بکار بردیم. البته باید بگویم که هدف ما بیشتر آموزش تکنیک sum = sum + counter و تشریح آن بود.

$$\text{Sum from 1 to n} = \frac{n(n+1)}{2}$$

$$\text{Sum from 1 to 100} = \frac{100(100+1)}{2} = (50)(101) = 5050$$

برنامه محاسبه جمع اعداد 1 تا n بدون نیاز به For – Next و با استفاده از فرمول ریاضی:

```
Sub Chapter5_Ex11()

  Dim number As Long
  number = InputBox("Enter a number?")
  Debug.Print (number * (number + 1)) / 2

End Sub
```

دستور Do – Loop

دستور For – Next یک حلقه را به تعداد مشخصی برای شما انجام می‌دهد. حال اگر بخواهید که یک حلقه داشته باشید که تعداد دفعات مشخصی نداشته باشد باید از دستور Do – Loop استفاده کرد.

قبل از اینکه دستور را معرفی کنم باید بدانید که این دستور شکل‌های مختلفی دارد اما در هر حال اصول و کلیات آن یکسان است.

شکل ساده دستور Do – Loop به صورت زیر است:

```
Do
      Statements
Loop
```

این دستور به صورت یک بلوک است و ابتدای آن با Do و انتهای آن با Loop مشخص می‌شود و هر عبارت/دستوری که در بدنه بلوک قرار گیرد به تعداد بی‌نهایت اجرا می‌شود.

مثال ۱ دستور Do – Loop – پیغام سلام

برنامه‌ای بنویسید که پیغام Hi را مرتبا چاپ کند.

در واقع ما در این برنامه انتهایی نداریم و باید پیغام Hi را بارها و بارها چاپ کنیم، بنابراین برنامه به شکل زیر خواهد شد:

```
Sub Chapter5_Ex12()

Do
    Debug.Print "Hi"
Loop

End Sub
```

تمرین: این پروسیجر را با F8 به صورت سطر به سطر اجرا کنید و ببنید که دستور Loop باعث اجرای مجدد بدنه بلوک می‌شود.

یادآوری: از آنجایی که این برنامه باعث ایجاد Loop می‌شود، قبل از اجرای آن، فایل اکسل را ذخیره کنید تا اگر احتمالا نتوانستید با زدن کلید CTRL+Break به برنامه پایان بدهید و اکسل هنگ کرد، برنامه‌ای که نوشته‌اید ذخیره شده باشد.

معرفی عبارت While در دستور Do - Loop

دقت داشته باشید که در اکسل ما هیچوقت به یک Loop بی‌نهایت مانند برنامه قبلی نیازی نداریم. باید برای یافتن حلقه‌ها و خارج شدن از آن راهکاری داشته باشیم. معمولا حلقه ها را تا جایی تکرار می‌کنیم که شرطی برقرار باشد و همینکه آن شرط برقرار نبود، از حلقه خارج خواهیم شد.

شکل کلی دستور به صورت زیر است:

```
Do
      Statements
Loop While Condition
```

مشاده می‌کنید که در قسمت Loop یک عبارت While اضافه شده است و سپس یک شرط گذاشته‌ایم و تا هنگامی که آن شرط برقرار است (یعنی True است) ، حلقه تکرار می‌شود

مثال ۲ دستور Do – Loop – برنده شدید

فرض کنید که می‌خواهیم از کاربر بخواهیم که یک عدد را به ما بدهد و اگر آن عدد برابر 5 به او بگوییم که برنده شده است و در غیر اینصورت به او فرصت دهیم تا مجدد عددی را وارد کند تا بالاخره او 5 را تایپ کند و برنده شود!

خلاصه این برنامه را به صورت زیر می‌توانیم بنویسیم:

- از کاربر بارها و بارها یک عدد پرسیده شود و چون مشخص نیست که این پرسش باید چند بار تکرار شود و برای همین باید از Do – Loop استفاده کنید.
- اگر عددی که کاربر می‌دهد برابر 5 نبود، حلقه تکرار شود.
- پایان حلقه وقتی اتفاق می‌افتد که کاربر عدد 5 را وارد کند.

حالا فرض کنید که عدد ورودی کاربر را در متغیر x می‌گذاریم و می‌خواهیم تکرار ما تا وقتی که x مخالف 5 است انجام شود بنابراین این شرط را یعنی x <> 5 را در قسمت Loop اینگونه اضافه می‌کنیم:

```
Loop While x <> 5
```

در واقع می‌توانیم این دستور را اینگونه بخوانیم:"Loop را تا وقتی که x مخالف 5 است را انجام بده" و کل برنامه به شکل زیر خواهد شد:

```
Sub Chapter5_Ex13()

Do
    x = InputBox("Enter a Number?")
Loop While x <> 5

MsgBox "You Win!"
End Sub
```

مثال ۳ دستور Do – Loop – اعداد دو رقمی

ما می‌دانیم که برای داشتن اعداد ۲ رقمی، یعنی اعداد بین ۱۰ تا ۹۹ به سادگی می‌توانیم از For – Next به شکل زیر استفاده کنیم.

```
For x = 10 to 99
Next x
```

حال برای تمرین می‌خواهیم که همین کار را با Do – Loop انجام دهیم:

```
Sub Chapter5_Ex14()
Dim x As Long
x = 10

Do
    Debug.Print x
    x = x + 1
Loop While x < 100

End Sub
```

شرح برنامه:

از قبل می‌دانیم که x = x +1 یعنی چه . در واقع یعنی مقدار متغیر x به اندازه ۱ واحد اضافه می‌شود و چون اولین عدد دو رقمی 10 است و نمی‌خواهیم که از عدد صفر شروع کنیم، به x مقدار اولیه 10 را می‌دهیم:

```
x = 10
```

و Do – Loop تا وقتی تکرار می‌شود که متغیر x کمتر از 100 است.

شرط در ابتدا یا انتهای بلوک Do - Loop

همانطور که دیدی در دستور Do – Loop می‌توانیم با While مشخص کنیم که تا چه هنگامی حلقه تکرار شود و همانطور که گفتیم تا «هنگامی که» شرطی که در While قرار داده‌ایم مقدار True داشته باشد، حلقه تکرار خواهد شد.

آیا به این نکته دقت کردید که ابتدا کل دستورات داخل بلوک اجرا می‌شود و در آخر بلوک است که شرط بررسی می‌شود؟ در واقع جالب است که این روشی که ما بکار بردیم، حداقل یکبار دستورات داخل بلوک اجرا می‌شوند در سپس شرط بررسی خواهد شد و اگر مقدار شرط True بود، آنگاه مجددا دستورات داخل بلوک اجرا خواهند شد.

حال می‌خواهم به شما بگویم که می‌توانید شرط را در همان ابتدای حلقه هم بررسی کنید و اگر شرط برقرار نبود، حلقه حتی یکبار هم اجرا نخواهد شد:

```
Do While Condition
      Statements
Loop
```

اگر بخواهیم با این روش برنامه قبلی را بنویسیم، نتیجه این خواهد شد:

```
Sub Chapter5_Ex15()
Dim x As Long
x = 10

Do While x < 100
    Debug.Print x
    x = x + 1
Loop

End Sub
```

نکته ۱: بررسی اینکه شرط را در ابتدا بررسی کنیم و یا در انتهای بلوک، در این مثال تفاوت خاصی ایجاد نشده است و فقط جای While از انتهای بلوک به ابتدای بلوک منتقل شده است.

نکته ۲: در بسیاری از مواقع است که ما می‌خواهیم ابتدا شرطی بررسی شود، مثلا فرض کنید که می‌خواهید سطر به سطر یک فایل txt را با برنامه‌ای بخوانید و روی اطلاعات آن فایل کاری را انجام دهید. احتمال دارد که آن فایل کاملا خالی باشد و به همین دلیل باید در ابتدا بررسی کنیم که آیا آن فایل دارای اولین سطر است و اگر موفق شدیم که اولین سطر فایل را بخوانیم، سپس بدنه بلوک اجرا شود.

البته موقعیت‌هایی هم وجود دارد که حتما باید حداقل یکبار بدنه حلقه اجرا شود و در این مواقع شرط را در آخر بلوک قرار می‌دهیم.

مثال ۴ دستور Do – Loop – خواندن یک فایل

در برنامه زیر ما یک فایل متنی را باز می‌کنیم و سپس اگر آن فایل دارای حداقل یک سطر بود، آن سطر را چاپ می‌کنیم.

در واقع قبل از آنکه فرمان چاپ کردن سطر را بدهیم، باید اصلا بررسی کنیم که آیا آن سطر وجود داشته است یا خیر و برای اینکار از While Not EOF(1) در ابتدای شروع حلقه استفاده می‌کنیم.

```
Sub Chapter5_Ex16()
Dim filename As String

filename = ActiveWorkbook.Path & "\Data.txt"  'File to read
Open filename For Input As #1

Do While Not EOF(1)

    Line Input #1, txt
    Debug.Print txt

Loop
Close #1

End Sub
```

شرح برنامه:

قصد ندارم که جزئیات این برنامه رو به شما توضیح بدهم و فقط قسمت شرط برای ما مهم است. در واقع با تابع EOF ، بررسی می‌کنیم که آیا به قسمت انتهایی فایلی که باز کرده‌ایم، رسیده‌ایم؟ اگر انتهای فایل «نبودیم» (یعنی **Not** EOF) آنگاه دستورات داخل حلقه اجرا خواهند شد.

کاملا مهم است که ابتدا مطمئن شویم که انتهای یک فایل نیستیم و حداقل یک سطر وجود دارد و آنگاه آن سطر از فایل را بخوانیم. اگر شما این شرط را در انتهای بلوک یعنی قسمت Loop قرار دهید، اگر فایلی باشد که کاملا خالی باشد، آنگاه برنامه ابتدا اولین سطر آن فایل را باید بخواند و چون هیچ سطری وجود ندارد، برنامه با خطا مواجه می‌شود.

توجه: برای استفاده از یک کد واقعا لازم نیست که در ابتدا همه جزئیات آنرا بدانید و کاملا کاری متداول است که برنامه‌ای را از اینترنت کپی کنیم و با تغییرات جزئی از آن فقط استفاده کنیم. به عنوان مثال در برنامه بالا شما می‌توانید مسیر و نام هر فایل دلخواه متنی را در متغیر FileName قرار دهید و خیلی لازم نیست که مفهوم #1 و یا Close #1 را بدانید. البته اگر مجبور بودید که بارها و بارها از این تکنیک استفاده کنید و یا حالت‌های پیچیده‌تری را بنوسید، توصیه می‌شود که کار با فایلهای متنی را کاملا فرا بگیرد.

مثال ۵ دستور Do – Loop – فاکتوریل

برنامه‌ای بنویسید که یک عدد را از کاربر بگیرد و فاکتوریل آن عدد را محاسبه کنید.

یادآوری: اگر بخواهیم فاکتوریل عدد 7 را بگیرید باید اعداد 1 تا 7 را در هم ضرب کنیم که نتیجه عدد 5040 خواهد شد:

```
1*2*3*4*5*6*7  = 5040
```

امیدواریم که شروع نکنید به غر زدن که مگر ما ریاضی دادن هستیم و فاکتوریل به چه درد ما می‌خورد. در واقع من سعی دارم که با مثالهای متنوع، دستورات را برای شما شفاف تر سازم.

```
Sub Chapter5_Ex17()

Dim num As Long
Dim fact As Long
fact = 1
num = InputBox("Enter an Integer : ")

Do
    fact = fact * num
    num = num - 1
Loop While num > 0

Debug.Print "Factorial= " & fact

End Sub
```

شرح برنامه:

ما یک عدد از کاربر می‌گیریم و آنرا در داخل متغیر num می‌گذاریم و سپس در حلقه از تکنیک زیر برای محاسبه حاصلضرب عددها استفاده می‌کنیم:

```
fact = fact * num
```

این تکنیک دقیقا مانند تکنیک sum = sum + counter است که در جمع اعداد بکار گرفتیم. در واقع در آنجا قصد داشتیم اعداد را جمع بزنیم و در اینجا می‌خواهیم اعداد را ضرب کنیم.

نکته ۱: از آنجایی که قرار است اعداد با هم ضرب شوند ، حتما باید مقدار اولیه متغیر fact را برابر 1 قرار دهیم، زیرا اگر fact برابر صفر باشد ، بدیهی است که ضرب هر عدد در صفر ، صفر خواهد شد.

بنابراین ما از قبل با دستور زیر مقدار اولیه متغیر fact را برابر 1 کرده ایم:

```
fact = 1
```

همچنین دستور num = num -1 هم که دقیقا شبیه x = x +1 است با این تفاوت که در اینجا ما از مقدار متغیر num یک واحد در هر حلقه کم خواهیم کرد.

حال که کلیات دستورات را گفتیم بیایید برنامه را سطر به سطر در ذهنمان اجرا کنیم :

شروع اجرا:

- مقدار متغیر num عددی است که کاربر می‌دهد. فرض کنید که کاربر عدد 7 را وارد کرده
- مقدار متغیر fact عدد 1 است

اولین اجرای حلقه:

- ابتدا سمت راست علامت مساوی، یعنی fact * num محاسبه می شود و نتیجه 7 * 1 می‌شود و در نتیجه fact = 7 خواهد شد.
- حال نوبت به سطر num = num -1 می‌رسد و همانطور که می‌دانید این دستور یعنی از متغیر num یک واحد کم کن و در نتیجه num = 6 خواهد شد.
- حال شرط Loop بررسی می‌شود و چون مقدار متغیر num هنوز بیشتر از 0 است، حلقه تکرار خواهد شد

دومین اجرا حلقه:

- از مرحله قبلی می‌دانیم که مقدار متغیر num عدد 6 است و مقدار fact نیز عدد 7 است
- سمت راست علامت مساوی، یعنی fact * num محاسبه می شود و نتیجه 6 * 7 می‌شود و در نتیجه fact = 42 خواهد شد.
- حال نوبت به سطر num = num -1 می‌رسد و در نتیجه num = 5 خواهد شد.
- چون هنوز num بیشتر از 0 است، حلقه تکرار می‌شود

سومین اجرا حلقه:

- از مرحله قبلی می‌دانیم که مقدار متغیر num عدد 5 است و مقدار fact نیز عدد 42 است
- سمت راست علامت مساوی، یعنی fact * num محاسبه می شود و نتیجه 5 * 42 می‌شود و در نتیجه fact = 210 خواهد شد.
- حال نوبت به سطر num = num -1 می‌رسد و در نتیجه num = 4 خواهد شد.
- چون هنوز num بیشتر از 0 است، حلقه تکرار می‌شود

چهارمین اجرا حلقه:

- از مرحله قبلی می‌دانیم که مقدار متغیر num عدد 4 است و مقدار fact نیز عدد 210 است
- سمت راست علامت مساوی، یعنی fact * num محاسبه می شود و نتیجه 4 * 210 می‌شود و در نتیجه fact = 840 خواهد شد.
- حال نوبت به سطر num = num -1 می‌رسد و در نتیجه num = 3 خواهد شد.
- چون هنوز num بیشتر از 0 است، حلقه تکرار می‌شود

و همینطور الی آخر حلقه اجرا می‌شود تا زمانی که متغیر num > 0 است.

این برنامه را با کلید F8 با استفاده از Watch ها مرحله به مرحله اجرا و بررسی کنید و مطمئن شوید که تکنیک‌های آنرا کاملا یاد گرفته‌اید و هیچ ابهامی ندارید.

دستور Exit Do

با دستور Exit Do می‌توانیم از حلقه خارج شویم، شکل کلی این دستور اینگونه است:

```
Do
      Statements
      Exit Do
Loop
```

مثال ۶ دستور Do – Loop – ک م م

برنامه‌ای بنویسید که سه عدد را بگیرد و سپس کوچکترین مضرب مشترک آن سه عدد را بیابد.

یادآوری ۱: ک‌م‌م سه عدد یعنی کوچکترین عددی که بر هر ۳ عدد بخش پذیر باشد. مثلا فرض کنید که عدد ۳ و ۴ و۶ را داریم و اگر به بپرسیم که کوچکترین عددی است که بر هر سه این اعداد بخش پذیر باشد، چیست، شما باید بگویید ۱۲. یعنی ۱۲ کوچکترین عددی است که اگر بر هر یک از این اعداد تقسیم شود، باقی‌مانده‌اش صفر خواهد شد.

بازهم لطفا اعتراض نکنید که این چه مثال‌هایی است که زده می‌شود زیرا از این مثال‌ها ساده‌تر برای آموزش دستورات نداریم و لطفا با حوصله آنها را انجام دهید.

روش حل ما به صورت سعی و خطا است. یعنی ما روش‌های ریاضی یافتن ک‌م‌م را بلد نیستیم و به همین دلیل همه اعداد را به ترتیب تولید می‌کنیم و بر سه عدد تقسیم می‌کنیم و اگر باقی مانده همه تقسیم‌ها صفر شد، آنگاه می‌گوییم که آن عدد ک‌م‌م آن سه عدد است.

یادآوری ۲: برای یافتن باقی مانده تقسیم (باخارج قسمت صحیح) از عملگر Mod استفاده می‌کردیم.

نکته ۱: در واقع چون ما نمی‌دانیم که تا چه عددی باید جلو برویم ، به همین دلیل باید از Do – Loop استفاده کنیم.

نکته ۲: همین که ما ک‌م‌م را یافتیم باید از حلقه خارج شویم و بدیهی است که لازم نیست تا بی‌نهایت حلقه را ادامه دهیم و برای این منظور (خارج شدن از حلقه) از دستور Exit Do استفاده خواهیم کرد:

```
Sub Chapter5_Ex18()

Dim num1 As Long, num2 As Long, num3 As Long
Dim counter As Long

num1 = InputBox("Enter first  number: ")
num2 = InputBox("Enter second number: ")
num3 = InputBox("Enter third  number: ")

Do
  counter = counter + 1
  If (counter Mod num1 = 0) And _
    (counter Mod num2 = 0) And _
    (counter Mod num3 = 0) Then Exit Do
Loop

Debug.Print "Least common multiple : " & counter

End Sub
```

یادآوری ۳: با استفاده از Space و یک «_» ما قسمت شرط IF را در چند سطر شکاندیم تا خواناتر باشد.

نکته ۳: شما می‌توانید از دستور IF و Exit Do استفاده نکنید و فقط شرط‌ها را در قسمت Loop اینگونه بنویسید:

```
Sub Chapter5_Ex19()

Dim num1 As Long, num2 As Long, num3 As Long
Dim counter As Long

num1 = InputBox("Enter first  number: ")
num2 = InputBox("Enter second number: ")
num3 = InputBox("Enter third  number: ")

Do
    counter = counter + 1
Loop While (counter Mod num1 <> 0) Or (counter Mod num2 <> 0) Or (counter Mod num3 <> 0)

Debug.Print "Least common multiple : " & counter

End Sub
```

نکته ۴: دقت کنید که در اینجا از تابع OR استفاده کرده‌ایم . یعنی ما باید Loop تا هنگامی که «حتی یکی از باقی‌مانده‌ها» مخالف صفر است را ادامه دهیم.

سوال: فرض کنید که می‌خواهیم ک‌م‌م سه عدد 11245 , 2048 , 1017 را بیابیم، آیا می‌توانید روشی را پیشنهاد کنید که برنامه ما کمی بهینه شود. (منظور از بهینه شدن یعنی سریعتر پاسخ را بیابیم و از اجراهای بیهوده، حلقه‌های اضافه و یا شرط‌ها اضافه جلوگیری کنیم.)

نکته ۵: ما در اکسل یک تابع به نام LCM داریم که برای یافتن ک‌م‌م است و کاملا توصیه می‌شود که به جای این برنامه که نوشته‌ایم، از آن تابع استفاده کنیم. در واقع ممکن است که تابع LCM اکسل، از روش‌های ریاضی بسیار سریعی برای یافتن ک‌م‌م استفاده کند و نه روشی مانند روش ما که غیربهینه ترین روش است. در برنامه زیر استفاده از توابع اکسل را در VBA می‌بینید که البته در فصول آینده قرار است به صورت کامل آنرا آموزش دهیم.

```
Sub Chapter5_Ex20()

Dim num1 As Long, num2 As Long, num3 As Long
Dim counter As Long

num1 = InputBox("Enter first  number: ")
num2 = InputBox("Enter second number: ")
num3 = InputBox("Enter third  number: ")

Debug.Print WorksheetFunction.Lcm(num1, num2, num3)

End Sub
```

آشنایی با عبارت Until در دستور Do – Loop

قبلا گفتم که در دستور Do – Loop می‌توانیم از While برای قسمت شرط استفاده کنیم و در اینجا می‌خواهیم بگوییم که علاوه بر While ، عبارت Until هم وجود دارد و می‌توانید از آن نیز استفاده کنید.

می‌توانیم بگوییم که عبارت Until ، همان While است اما از یک نگاه دیگر. معمولا دستور While و Until برنامه نویسان تازه کار را کمی گیج می‌کند و به همین دلیل کافی است که دستور While را یاد بگیرید و فقط از آن استفاده کنید.

البته من Until را هم یک توضیح مختصر خواهم داد تا اگر با آن در کدهای اینترنتی یا سایر کتاب‌ها مواجه شدید، بدانید که تفسیر آن چگونه است.

بگذاریم یک مثال بزنم، "تاوقتی که هوا بارانی باشد من در خانه خواهم بود" و "به محض اینکه باران قطع شود، در خانه نخواهم ماند". ما در این دو جمله یک معنی را با دو شکل متفاوت بیان می‌کرده‌ایم. دقیقا فرق دوعبات While و Until اینگونه هستند.

مثلا ما شرط «تا وقتی که عدد کمتر از ۱۰۰ باشد» را با While اینگونه نوشتیم و «تا هنگامی که» این شرط برقرار است حلقه تکرار خواهد شد:

```
Loop While x < 100
```

حال می‌توانیم همین شرط را اینگونه بنویسم :

```
Loop Until x = 99
```

یعنی حلقه را به محض اینکه عدد x برابر 99 شد، خاتمه بده و «به محض اینکه» این شرط برقرار شود، یعنی x برابر 99 شود، از حلقه خارج می‌شویم.

نکته : من واژه While را به «تا وقتی که» ترجمه کرده‌ام و Until را «به محض اینکه».

جمع بندی دستور Do – Loop

شکل کلی دستور به یکی از دو صورت زیر است:

```
Do [{While | Until} condition]
      [statements]
      [Exit Do]
      [statements]
Loop
```

و یا این شکل:

```
Do
      [statements]
      [Exit Do]
      [statements]
Loop [{While | Until} condition]
```

نکته ۱: هر جا که دستوری در بین علامت [ ] گذاشته شده است یعنی آن قسمت اختیاری است و می‌توانید از آن استفاده نکنید. مثلا دستور Exit Do در این علامت گذاشته شده است.

نکته ۲: اگر بخواهید که شرط بگذارید باید یکی از عبارت های While و یا Until را استفاده کنید و اگر دقت کنید بین این دو عبارت علامت «|» را گذاشته است که به معنای «یا» است.

یادآوری: قبلا گفتم که منظور از Statement یعنی هر عبارت و یا دستوری که VBA آنرا بتواند بفهمد و تفسیر کنید مثلا یک دستور IF ، یا یک دستور Debug.Print و البته حتی می‌توانید از For – Next هم در داخل Do – Loop استفاده کنید و یا حتی یک Do – Loop دیگر و اصلا هیچ محدودیتی نداریم که چه چیزی را در داخل بدنه Do – Loop بنویسیم.

حلقه‌های تو در تو

تا اینجا با حلقه‌ها و دو دستور اصلی آن یعنی دستور For – Next و دستور Do – Loop آشنا شدیم. در ادامه قصد داریم چند مثال و تکنیک کاربردی را از ترکیب‌های این دستورات را مطرح کنیم زیرا در بسیاری از موارد شما در داخل بدنه یک دستور For – Next از یک حلقه دیگر استفاده می‌کنید.

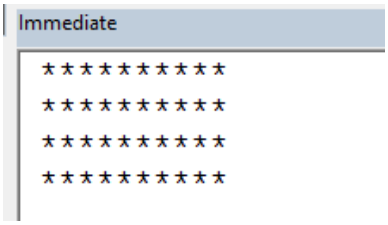
به عنوان مثال از یک For – Next در داخل For – Next استفاده شود که صورت کلی آن به شکل زیر است:

```
For x = 1 to 10
      For y = 1 to 10

      Next y
Next x
```

مثال ۱ حلقه‌های تو در تو – مستطیل

برنامه‌ای بنویسید که یک مستطیل با طول ۱۰ ستاره و ارتفاع ۴ ستاره، مانند شکل زیر در محیط Immediate چاپ کند.

تصویر رسم مستطیل با For - Next




```
Sub Chapter5_Ex21()

Dim row As Long, column As Long
For row = 1 To 4
  For column = 1 To 10
    Debug.Print "*";
  Next column
  Debug.Print
Next row

End Sub
```

شرح برنامه:

ما به ۱۰ ستاره پیاپی نیاز داریم و برای تولید آنها کافی است که فرمال Debug.Print "*"; را 10 بار اجرا کنیم و اینکار را با حلقه زیر انجام می‌دهیم:

```
For column = 1 To 10
    Debug.Print "*";
Next column
```

چون به ۴ سطر نیاز داریم، باید این حلقه را ۴ بار تکرار کنیم، بنابراین یک حلقه دیگر به تعداد ۴ تکرار در نظر می گیریم و این حلقه را در داخل آن قرار می دهیم:

```
For row = 1 To 4

  For column = 1 To 10
    Debug.Print "*";
  Next column

Next row
```

یادآوری:هر دستور Debug.Print باعث چاپ هر مقدار در یک سطر جدید می شود و اگر بخواهیم که مقادیر پشت سر هم چاپ شوند باید از علامت «;» (سمیکالن یا نقطه ویرگول) در انتهای آن استفاده کنیم و اگر بخواهیم که اطلاعات در سطر بعدی چاپ شوند کافی است که بنویسیم Debug.Print.

تمرین: توصیه می‌شود که این برنامه را با کلید F8 به صورت سطر به سطر اجرا کنید تا متوجه نحوه کارکرد آن شوید.

مثال ۲ حلقه‌های تو در تو – جدول ضرب

برنامه‌ای بنویسید که جدول ضرب را چاپ کند.

```
Sub Chapter5_Ex22()

Dim x As Long, y As Long
For x = 1 To 9
    For y = 1 To 9
        Debug.Print x * y,
    Next y
    Debug.Print
Next x

End Sub
```

شرح برنامه:

یک متغیر به نام x تعریف می‌کنیم و سپس با یک حلقه For – Next به x اعداد 1 تا 9 را می‌دهیم و در داخل این حلقه، یک حلقه دیگری با متغیری به نام y تعریف می‌کنیم.

نکته مهم آن است که بدانید برای هر تغییر متغیر x یکبار کل حلقه y اجرا می‌شود. یعنی هنگامی که x = 1 است ، حلقه y اجرا خواهد شد و بنابراین متغیر y از عدد 1 تا 9 تغییر می‌کند و در اجرای بعدی حلقه x یعنی هنگامی که x = 2 شود، مجددا متغیر y از مقدار 1 تا 9 تغییر خواهد کرد.

بگذارید برنامه را در ذهنمان اجرا کنیم:

برنامه که اجرا می‌شود اولین مقدار x برابر عدد 1 خواهد شد .

سپس حلقه داخلی اجرا می‌شود و در حلقه داخلی ما متغیر y را داریم که از عدد 1 شروع می شود.

بنابراین حاصل عبارت x * y مقدار 1 * 1 چاپ خواهد شد.

سپس مقدار y عدد 2 می‌شود و حاصل عبارت x * y چاپ خواهد شد و سپس y عدد 3 می‌شود و همینطور الی آخر تا حلقه y به اتمام برسد.

هنگامی که حلقه y تمام شود، نوبت به x بعدی یعنی عدد x = 2 خواهد رسید و در اینجاست که مجددا حلقه y اجرا خواهد شد یعنی y عدد 1 می شود و حاصل عبارت x * y مقدار 1 * 2 خواهد شد و نوبت به y بعدی یعنی عدد 2 می‌رسد و در نتیجه 2 * 2 چاپ می‌شود و سپس 3 * 2 و سپس 4 * 2 تا هنگامی که حلقه y تمام شود.

بعد از اتمام حلقه y نوبت به x بعدی یعنی 3 خواهد رسید و همینطور الی آخر.

تمرین: توصیه می‌شود که این برنامه را با کلید F8 به صورت سطر به سطر اجرا کنید تا متوجه نحوه کارکرد آن شوید.

مثال ۳ حلقه‌های تو در تو – اعداد اول

برنامه‌ای بنویسید که کلیه اعداد اول بین ۲ تا ۱۰۰۰ را چاپ کند.

یادآوری: اعداد اول (Prime Numbers) اعدادی هستند که بر هیچ عددی بخش (بجز ۱ و خودشان) بخش پذیر نیستند. پس اگر عددی به یک عدد دیگر بخش پذیر بود، اول نیست.

```
Sub Chpater5_Ex23()
Dim x As Long
Dim flag As Boolean

For x = 2 To 1000

    flag = True

    For y = 2 To x - 1

        If x Mod y = 0 Then
            flag = False
            Exit For
        End If

    Next y

        If flag Then Debug.Print x
Next x

End Sub
```

شرح برنامه:

اول بگویم که شاید برای برنامه نویسان تازه کار این برنامه کمی سخت به نظر برسد و در واقع چنین هم است و خیلی انتظار ندارم که در اولین نگاه و توضیح شما آنرا دقیقا متوجه شوید. با این حال یک تکنیک جالب در آن بکار رفته است که پر کاربر می‌باشد و از این جهت این برنامه مهم است.

در ابتدا با ساختن یک حلقه ما کلیه اعداد بین ۲ تا ۱۰۰۰ را تولید می‌کنیم (متغیر x) .

در داخل این حلقه می‌خواهیم که اول بودن x را آزمایش کنیم بنابراین یک نشانه یا یک علامت برای اینکه مشخص کنیم که x اول است یا نه، در نظر می‌گیریم و نام آنرا flag می‌گذاریم و فرض می‌کنیم که x اول است و مقدار اولیه آن flag را برابر True می‌گیریم.

حال باید x را برهمه اعدادی که ممکن است بخش پذیر باشد، تقسیم کنیم بنابراین متغیر y را می‌سازیم که از عدد ۲ تا عدد x-1 است. (لازم نیست که بخش پذیری عدد x را بر عدد ۱ و یا خودش بررسی کنید و به همین دلیل از عدد ۲ شروع می‌کنیم و تا x-1 آزمایش خودمان را انجام می‌دهیم.)

ما یک دستور IF می‌نویسیم که اگر حاصل تقسیم عدد x بر عدد y برابر صفر شد، دو کار انجام شود:

- اول آنکه نشانه ما را false کند.
- دوم آنکه دیگر لازم نیست x را بر ادامه مقادیر y تقسیم کند، چون همین که یک y پیدا شود که x بر آن بخش پذیر باشد، قطعا x عددی اول نیست.

پس از آزمایش تقسیم پذیری x بر همه اعداد و خروج از حلقه y ، ما به آن نشانه نگاه می‌کنیم،اگر نشانه ما هنوز True بود، یعنی فرض ما درست بوده است و x عددی اول است و اگر flase شده بود، یعنی اینکه عددی y پیدا کرده ایم که باعث شده است flag ما false شود و بنابراین اول نیست.

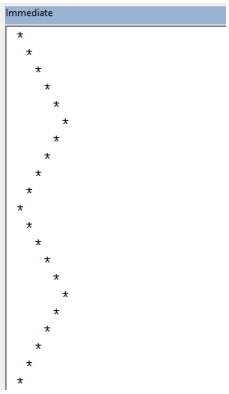
نکته ۱: از این تکنیک یعنی گذاشتن یک نشانه در حلقه‌ها که ما در اینجا با متغیر flag آنرا دیدیم، در بسیاری از کدهای استفاده می‌شود و معمولا به این نشانه (صرفنظر از نامی که برای آن متغیر استفاده می‌کنیم) flag می‌گویند. معمولا flag را از نوع Boolean می‌گیرند و البته ممکن است که از نوع عدد باشد و مقادیر 1 / 0 را قبول کند.

سوال: چگونه می‌توان این برنامه را بهینه کرد و از اجرای شدن بیهوده حلقه y تا حد ممکن جلوگیری کرد؟

مثال ۴ حلقه‌های تو در تو – چاپ آرام زیگزاگ

برنامه‌ای بنویسید که به صورت آرامی باعث چاپ شکل زیر در محیط Immediate شود:

تصویر زیگزاگ با ستاره در محیط Immediate



```
Sub Chpater5_Ex24()
Dim i As Long, flag As Long, y As Long, delay As Long
flag = 1

For i = 0 To 100

    Debug.Print VBA.Space(y) & "*"
    y = y + flag
    If y Mod 5 = 0 Then flag = -flag

    For delay = 1 To 100000: Next delay
    'Application.Wait (Now + TimeValue("00:00:01"))
Next i

End Sub
```

شرح برنامه: ما ۱۰۰ عدد ستاره را چاپ می‌کنیم و قبل از هر ستاره به تعداد y ، جای خالی (space) می‌گذاریم. برای گذاشتن Space ها از تابع Space استفاده می‌کنیم:

```
Debug.Print Space(y) & "*"
```

گفتیم که متغیر y بیانگر تعداد Space قبل از هر علامت ستاره است و به همین دلیل باید متغیر y در هر بار اجرای حلقه اصلی، یک واحد تغییر کند. ما متغیر flag = 1 در ابتدا تعریف می‌کنیم و بنابراین دستور y = y + flag یعنی y = y + 1 و این هم یعنی افزایش یک واحدی متغیر y.

حال پس از گذاشتن پنج ستاره اول، شکل زیر را در Immediateخواهیم داشت یعنی در اولین ستاره هیچ Space قبلش نیست و دومین ستاره ، یک Space دارد و سومین ستاره، دو Space تا پنجمین ستاره که چهار Space قبلش موجود است:

```
*
 *
  *
   *
    *
     *
```

پس از پنجمین ستاره، چون می‌خواهیم که حالت زیگزاگ داشته باشیم بنابراین برای ششمین ستاره باید از آخرین تعداد Space ها، یک واحد کم کرده باشیم به همین دلیل کافی است که متغیر flag عدد 1- شود و در نتیجه عبارت y = y + flag خواهد شد y = y + (-1) و این یعنی از y یک واحد کم شود و این IF را می‌نویسیم که اگر مقدار y بر عدد ۵ بخش پذیر بود، علامت متغیر flag را تغییر بده:

```
    If y Mod 5 = 0 Then flag = -flag
```

نکته ۱: این برنامه کمی پیچیده است و باید چند بار با حوصله و صبر با کلید F8 آنرا اجرا کنید و متغیرها را با watch بررسی کنید تا بتوانید تکنیک آنرا عمیقا متوجه شوید.

ما در صورت مساله گفته بودیم، آرام چاپ شود. زیرا اگر F5 را بزنیم بسیار سریع این زیگزاگ چاپ می‌شود و برای آنکه بتوانیم چاپ شدن ستاره‌ها را ببینیم، باید بین چاپ هر ستاره تا چاپ ستاره بعدی کمی زمان داشته باشیم. برای دادن این زمان اضافه، ما یک حلقه به شکل زیر ساختیم:

```
For delay = 1 To 100000: Next delay
```

از آنجایی که کامپیوتر مجبور است این حلقه را انجام دهید باعث ایجاد وقفه‌ای کوتاه می‌شود.

نکته ۲: سرعت اجرای حلقه‌ها بستگی به قدرت CPU دستگاه شما دارد و اگر در دستگاه شما این حلقه خیلی کند است، تعداد آنرا کمتر کنید مثلا تا عدد 1000 بیشتر حلقه اجرا نشود.

نکته ۳: با دستور زیر هم می‌توانید وقفه‌های یک ثانیه‌ای ایجاد کنید اما یک ثانیه برای اینکه چاپ زیگزاگ‌های ما زیبا شود، کمی زیاد است و از آنجایی که نمی‌توانیم کمتر از یک ثانیه وقفه با این دستور بسازیم، برای همین شاید استفاده از یک For – Next برای کار ما در اینجا بهتر باشد:

```
Application.Wait (Now + TimeValue("00:00:01"))
```

فصل ششم – خطاها و مدیریت آنها

خطاها اجتناب ناپذیر هستند. لاستیک اتومبیل پنجر می‌شود و نوسان برق باعث سوختن یخچال می‌شود، پژوی ۴۰۵ و یا گوشی سامسونگ آتش می‌گیرند. در دنیای برنامه نویسی هم ما با خطاهایی مواجه می‌شویم و باید آنها را به گونه‌ای مدیریت کنیم. در این فصل می‌خواهیم درباره خطاها و مدیریت آنها صحبت کنیم و باید به شما بگویم که تقریبا هر برنامه‌ای که می‌نویسیم باید خطاهای آنرا مدیریت کنیم و قطعا نادیده کردن این مبحث مشکلات جدی را برای شما و استفاده کنندگان از برنامه‌تان بوجود خواهد آورد.

قبل از شروع به شما بگویم که عنوان انگلیسی این بحث در کتابهای برنامه نویسی VBA، « Error Handling» است . البته در برخی از زبان‌های برنامه نویسی مانند C# به آن Exception Handling یعنی مدیریت استثناءها نیز می‌گویند .در ضمن واژه کلی Debugging به معنی خطایابی هم برای این بحث بکار می‌رود و منوی Debug محیط VBE برای خطاها و یافتن آنها کاربرد دارد.

در ضمن به ترجمه دو واژه Bug و Error را در فارسی باید اشاره کنم که ما هر دوی این واژه‌ها را «خطا» ترجمه می‌کنیم اما این ترجمه خوبی نیست زیرا این دو واژه کاملا بار معنایی متفاوتی را دارند.

وقتی که می‌گوییم Bug داریم، در واقع ما از وجود خطایی صحبت می‌کنیم که اساسا از چرایی، دلیل و چگونگی بوجود آمدن آن اطلاعی نداریم و با وجود یک Bug برنامه می‌تواند سال‌ها به کار خودش ادامه دهد و متوقف نمیشود اما در نهایت باعث عملکرد غیرقابل پیش بینی می‌شود. خیلی از مواقع یک Bug خطایی Logical است. مثلا برنامه نویس اشتباها به جای آنکه عددی را بر ۲ تقسیم کند، آنرا در ۲ ضرب کرده است.

اما در مورد Error‌ها وضعیت کاملا فرق می‌کند، برای ما Error خطایی است که دلیل آن در اکثر مواقع مشخص است و در بسیاری از موارد Error‌ها اجازه نمی‌دهند که برنامه اجرا شود.

در این فصل ما در خصوص Error‌ها صحبت خواهیم کرد.

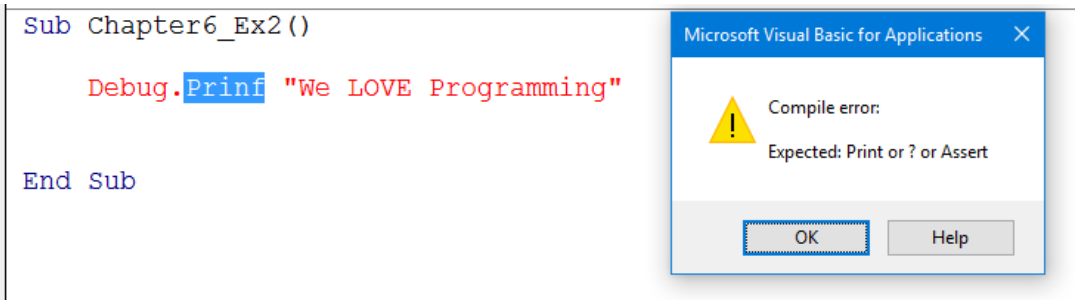
تذکر ۱: پیام خطاهای VBA بسیار کلی هستند و معمولا راهنمایی چندانی برای خطایابی به شما نمی‌کنند.

خطاهای نوشتاری – Syntax Error

این نوع از خطاها دوست شما هستند و هنگامی رخ می‌دهند که شما قوانین زبان VBA را رعایت نکرده باشید. قبلا گفتیم که VBA دارای قوانین و اصولی است و اگر قواعد نگارشی آنرا رعایت نکنید، با Syntax Error ها (خطاهای نگارشی / خطاهای دستوری) مواجه می‌شوید.

البته در خیلی از Syntax Error ها لازم نیست که حتما برنامه را اجرا کنید، زیرا VBE دارای Auto Syntax Check است یعنی همین که شما کلید Enter را بزنید ، دستورات شما را بررسی می‌کند و اگر یک Syntax Error را تشخیص دهد، بلافاصله آن سطر را قرمز خواهد کرد و پیام خطا را نمایش می‌دهد:

تصویر Auto Syntax Chek

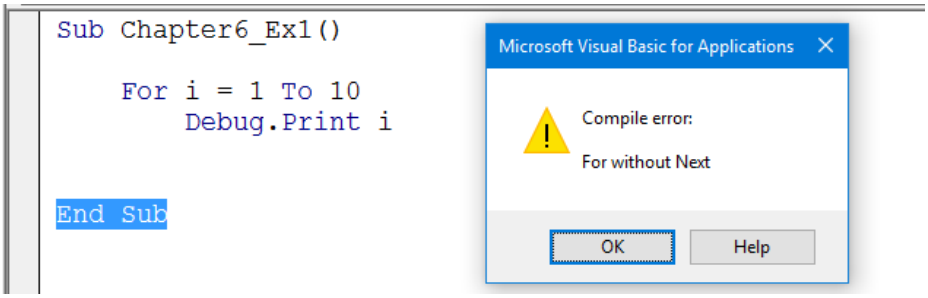


نکته ۱: معمولا دکمه Help در پنجره خطاها هیچ کمکی به شما نمی‌کند.

نکته ۲: شما می‌توانید حالت Auto Syntax Check را از Tools → Options → Editor (tab) غیر فعال نمایید.

می‌دانیم که در دستورات بلوکی باید انتهای بلوک مشخص شود و اگر شما یک For بنویسید و فراموش کنید که Next را بنویسید، بازهم یک Syntax Error خواهید داشت:

تصویر پیغام خطا به دلیل Syntax Error



البته این خطا توسط Auto Syntax Check قابل یافتن نیست، زیرا Auto Syntax Check فقط می‌تواند یک سطر را بررسی کند و نه کل سطرها را .

به همین دلیل این خطا تا هنگامی که شما کلید F5 را نزنید، آشکار نخواهد شد . هنگامی که کلید F5 را برای Run کردن برنامه بزنید، قبل از اجرا یک بررسی برای یافتن خطاهای شما انجام می‌شود و به شما خطا خواهد داد که دستور Next را ننوشته‌اید.

در این لحظه اگر شما کلید Ok را بزنید، VBE «سعی» می‌کند که دستور یا خطی را که منجر به خطا شده است را برای شما مشخص کند اما در اینجا VBA نمی‌تواند بگوید که شما باید Next را کجا قرار می‌دادید و به همین دلیل VBE نام پروسیجر را به عنوان جایی که خطا رخ داده است مشخص می‌کند و آنرا زرد رنگ خواهد کرد:

تصویر زرد رنگ شدن خطی که احتمالا منجر به خطا شده است

```
Sub Chapter6_Ex1()

    For i = 1 To 10
    Debug.Print i
    Debug.Print "hi"
    Debug.Print "bye"

End Sub
```

نکته ۳: در این حالت که سطری برای شما زرد رنگ می‌شود، اصطلاحا حالت Debuging می‌گوییم. یعنی شما قادر هستید که از ابزارهای خطایابی مانند Tool Tip و یا Watch و یا کلید F8 استفاده کنید و خطا را بیابید، آنرا اصلاح کنید و اجرای برنامه را ادامه دهید.

نکته ۴: ما تاکنون از کلید F8 برای درک نحوه اجرای برنامه‌ها استفاده کرده‌ایم و حال باید دقیق تر به شما بگویم که کلید F8 و اجرای سطر به سطر برنامه، استفاده از ToolTip و یا Watch برای مشاهده مقدار یک متغیر، استفاده از دستور Debug.Print و قسمت Immediate همگی از جمله ابزارها و روش‌های Debug هستند.

نکته ۴: هنگامی که شما در حالت Debugکردن هستید، نمی‌توانید سایر پروسیجرها را اجرا کنید و اگر بخواهید که از حالت Debug خارج شوید باید دکمه Reset را بزنید:

تصویر دکمه Reset و خارج شدن از حالت Debug

قانون یکم خطایابی

همواره قبل از تحویل و یا ارسال یک برنامه آنرا اجرا کنید تا متوجه خطاهای Sytax Error شوید. البته حتما لازم نیست که برنامه را اجرا کنید، بلکه اگر برنامه را Compile کنید ، Syntax Error ها مشخص می‌شوند.

برای Complie یک برنامه از منوی Compile VBA Project → Complie را می‌زنیم.

نکته : هر برنامه قبل از اینکه اجرا شود، باید به زبان ماشین ترجمه شود و سپس اجرا خواهد شد، به فرآیند ترجمه به زبان ماشین اصطلاحا Complie شدن می‌گوییم.

قانون دوم خطایابی

همواره Auto Syntax Check را در حالت فعال قرار دهید. البته ممکن است که برخی از مواقع مثل موقعی که می‌خواهید کدهایی را کپی کنید، این حالت را موقتا خاموش کنید اما بلافاصله آنرا به حالت روشن درآورید.

خطاهای درهنگام کار برنامه– Run Time Error

معمولا در هنگام کارکردن با برنامه با این خطاها مواجه می‌شویم و به همین دلیل به آنها Run Time میگوییم.

- مثلا از کاربر می‌خواهیم که یک عدد مثبت صحیح وارد کند و او به اشتباه یک عدد اعشاری منفی را تایپ می‌کند.
- یک عدد را بر صفر تقسیم می‌کنیم و خطای Division by Zero (تقسیم بر صفر) رخ می‌دهد.
- میخواهیم یک فایل را باز کنیم در حالی که آن فایل وجود ندارد.
- میخواهیم یک سلول را ویرایش کنیم در حالی که آن شیت قفل (Protect) شده است.

معمولا پیش بینی این خطاها دشوار است زیرا در خیلی از مواقع ما واقعا نمی‌دانیم که در هنگام اجرای یک برنامه چه حالت‌های خاصی ممکن است پیش بیاید.

قانون سوم خطایابی

بهترین روش مقابله با این خطاها (یعنی خطاهای Run Time Error) ، پیش گیری است . در واقع یک برنامه نویس با تجربه سعی می کند که تمامی حالت های محتمل را که ممکن است منجربه خطا شود را حدس بزند و با نوشتن شرطها و دستوراتی از رخ دادن آنها جلوگیری کند.

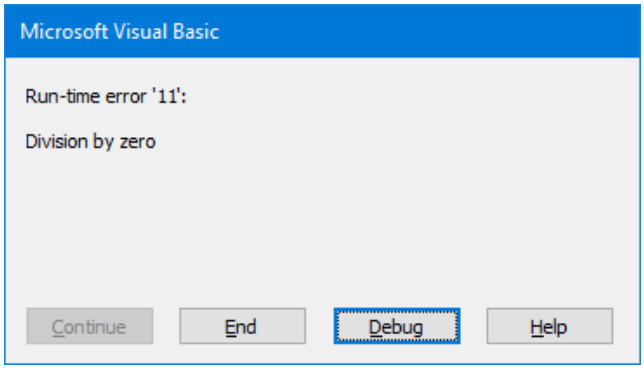
مثال ۱ Error Handling – تقسیم بر صفر

برنامه‌ای بنویسید که دو عدد را از کاربر بگیرد و عدد اول را بر عدد دوم تقسیم کند و نتیجه را نمایش دهد.

```
Sub Chapter6_Ex3()
Dim num1 As Long, num2 As Long

    num1 = InputBox("Enter First Number :")
    num2 = InputBox("Enter Second Number :")
    MsgBox num1 / num2

End Sub
```

قطعا این برنامه از نظر منطقی کاملا صحیح است اما ممکن است که کاربر عدد num2 را صفر وارد کند و در نتیجه خطای Division By Zero رخ دهد:

تصویر خطای تقسیم بر صفر



بنابراین بهتر است که با دستور IF بررسی کنیم که متغیر دومی یک عدد غیر صفر است:

```
Sub Chapter6_Ex4()
...
    If num2 <> 0 Then MsgBox num1 / num2
End Sub
```

و از این پس قطعا خطای Division By Zero هرگز پیش نخواهد آمد. اما خوشحال نشوید. بازهم این برنامه ساده ممکن است با خطاهایی مواجه شود.

یک لحظه خواندن این کتاب را متوقف کنید و حدس بزنید که چه خطاهایی محتمل است پیش بیاید؟

یکی از خطاهایی که کاملا محتمل است رخ دهد آن است که کاربر اصلا عدد وارد نکند و مثلا یک متن بدهد و آنوقت شما در سطرهای مقدار دهی متغیر num1 و یا num2 با خطا مواجه خواهید شد.

در ضمن اینکه اگر کاربر هنگام وارد کردن یکی از متغیرها کلید Cancel را بزند، مجددا در مقدار دهی متغیرها خطایی را خواهیم داشت.

برای آنکه همه این حالت را پوشش دهیم، متغیرهای num1, num2 را از نوع متنی تعریف می‌کنیم:

```
Dim num1 as String , num2 as string
```

و با تابع IsNumeric بررسی می‌کنیم که آیا متغیرها از نوع عددی هستند و سپس در صورتی که هر دو متغیر عدد بودند و مقدار عددی num2 نیز غیر صفر بود، آنگاه محاسبه تقسیم انجام شود:

```
Sub Chapter6_Ex5()
Dim num1 As String, num2 As String

    num1 = InputBox("Enter First Number :")
    num2 = InputBox("Enter Second Number :")

    If IsNumeric(num1) And IsNumeric(num2) And Val(num2) <> 0 _
                        Then MsgBox Val(num1) / Val(num2)

End Sub
```

نکته ۱ : با تابع val می‌توانیم ارزش عددی یک متغیر از نوع متن را استخراج کنیم. این تابع دقیقا مانند تابع Value در اکسل است.

نکته ۲: از آنجایی که VBA یک نرم افزار سخت گیر نیست، می‌توانید مستقیما از num1/num2 برای محاسبه تقسیم استفاده کنید و لازم نیست که حتما val(num1) / val(num2) شود. اگر چه استفاده از تابع val کاری حرفه‌ای است و همواره در کد به شما این تذکر را می‌دهد که این متغیرها از نوع عددی نمی باشند.

چند توصیه برای پیشگیری از خطاها

- کاری کاملا حرفه‌ای است که با Options Explicit ، تعریف متغیرها را اجباری کنید تا اگر در تایپ آنها دچار اشتباهی شدید، قبل از Complie شدن به شما هشدار داده شود. در زبانهایی مانند C, C# و ... تعریف متغیرها کاملا اجباری است.
- از تعریف متغیرهایی از نوع‌های کلی مانند Variant و یا Object خودداری کنید و نوع متغیر را دقیقا مشخص کنید.
- همواره مقدار ورودی کاربر را بررسی کنید . به این کار Data Validation می گویند و به او اجازه ندهید که با ورودی اشتباه، باعث Run Time Errorها شود.
- کتاب های خوب را مطالعه کنید تا لازم نباشد همه چیز را با سعی و خطا یاد بگیرید و با دانش خود بتوانید از رخ دادن خطاها جلوگیری کنید.

مدیریت خطاها با دستور On Error GoTo

دیدم که در حالت عادی هر گاه خطایی رخ دهد، برنامه متوقف شده و در حالت Debug قرار می‌گیرد. حال به شما دستوراتی را معرفی می‌کنیم تا اگر خطایی رخ داد بتوانید دستور دهید تا کدام قسمت پروسیجر را اجرا شود.

به عنوان مثال با دستور زیر می‌گوییم که اگر هر خطایی در هر یک از سطرها (که بعد از این دستور نوشته‌اند)، رخ داد، به سطر شماره 10 برو:

```
On Error Goto 10
```

البته معمولا نام سطر را چیزی مانند ErrorHandler و یا ErrHandler می‌گذاریم:

```
On Error Goto ErrHandler
```

مثال ۲ Error Handling – دستور On Error GoTo

برنامه ای بنویسید که دو عدد را گرفته و آنها بر هم تقسیم کند.

البته ما قبلا با دستور IF توانستیم این برنامه را بنویسیم و اکنون می‌خواهیم که آنرا با این روش پیاده‌سازی کنیم.

```
Sub Chapter6_Ex6()
Dim num1 As Long, num2 As Long

    num1 = InputBox("Enter First Number :")
    num2 = InputBox("Enter Second Number :")

On Error GoTo ErrHandler:
MsgBox num1 / num2
Debug.Print "Done!"

ErrHandler:
    MsgBox "Error in your code!", vbCritical


End Sub
```

شرح برنامه، سناریو ۱:

فرض کنید که کاربر یک عدد برای Num1 و سپس num2 را صفر وارد می‌کند. حال دستور MsgBox num1/num2 با خطا تقسیم بر صفر مواجه می‌شود و چون در سطر قبلی آن دستور On Error GoTo ErrHandler را نوشتیم، خطایی به کاربر نمایش داده نمی‌شود، و برنامه به سطر ErrHandler می‌پرد و Error in your code! چاپ خواهد شد.

شرح برنامه، سناریو ۲:

فرض کنید که کاربر برای num1 و num2 دو عدد مثلا ۱۰ و ۲۰ را وارد می‌کند و طبیعتا هیچ خطایی نخواهیم داشت. اما قبلا گفتیم که همه سطرهای پروسیجر پشت سر هم و یکی پس از دیگری اجرا خواهند شد.

بنابراین ابتدا حاصل تقسیم نمایش داده می‌شود و سپس Done! در Immediate چاپ می‌شود و سپس سطر MsgBox "Error in your code!" اجرا می‌شود!

در واقع قطعا ما نمی‌خواهیم که این پیام چاپ شود، اما بالاخره این سطر در یک جای این پروسیجر باید نوشته شود و قطعا اجرا هم خواهد شد.

راه حل این کار آن است که ما قبل از سطر ErrHandler با دستور Exit Sub از پروسیجر خارج شویم و در واقع همواره باید قبل از سطر ErrHandler از پروسیجر خارج شوید:

```
Sub Chapter6_Ex7()
Dim num1 As Long, num2 As Long

    num1 = InputBox("Enter First Number :")
    num2 = InputBox("Enter Second Number :")

On Error GoTo ErrHandler:
MsgBox num1 / num2
Debug.Print "Done!"

Exit Sub

ErrHandler:
    MsgBox "Error in your code!", vbCritical

End Sub
```

طبیعی است که اگر خطایی رخ دهد، برنامه به سطر ErrHandler می‌پرد و دستور Exit Sub در صورت وجود خطا هیچگاه اجرا نخواهد شد.

شرح برنامه، سناریو ۳:

حال فرض کنید که کاربر به جای یک عدد، یک متن (مثلا حرف x) را برای num1 وارد کند و چون متغیر num1 و num2 از نوع Long (عددی هستند)، بنابراین ما در سطر زیر دچار خطایی خواهیم شد:

```
num1 = InputBox("Enter First Number :")
```

و چون دستور On Error GoTo را ما پس از این دستور نوشته‌ایم، این خطا را مدیریت نکرده‌ایم و از طرف VBA خطایی رخ می‌دهد و برنامه به حالت Debug می‌رود.

اگر بخواهیم که خطایابی در هنگام مقدار دهی متغیرها را با دستور On Error GoTo مدیریت کنیم، حتما باید آنرا قبل از این فرمان‌ها بنویسیم:

```
Sub Chapter6_Ex8()
Dim num1 As Long, num2 As Long

On Error GoTo ErrHandler:

    num1 = InputBox("Enter First Number :")
    num2 = InputBox("Enter Second Number :")

MsgBox num1 / num2
Debug.Print "Done!"

Exit Sub

ErrHandler:
    MsgBox "Error in your code!", vbCritical

End Sub
```

دستور On Error GoTo 0

دیدیم که با دستور On Error GoTo ErrHandler مدیریت خطاها را فعال می‌کنیم و اگر بخواهیم که این حالت غیر فعال شود، یعنی مواجه با خطاها به حالت عادی برگردد و در صورت وقوع خطایی برنامه در حالت Debug برود، از دستور زیر استفاده می‌کنیم:

```
On Erro GoTo 0
```

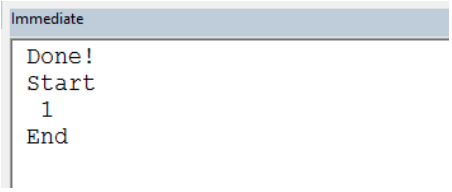
بی‌خیال خطاها – On Error Resume Next

بله، ما دستوری هم داریم که اگر برنامه در هنگام اجرا با خطایی مواجه شد، فقط آن سطر یا سطرها که از برنامه که خطا داده است، اجرا نمی‌شود و سایر سطرها اجرا خواهند شد:

```
Sub Chapter6_Ex9()
On Error Resume Next
    Debug.Print "Start"
    Debug.Print 0 / 0
    Debug.Print 2 / 2
    Debug.Print "End"
End Sub
```

دستور Debug.Print 0 / 0 چون خطای تقسیم بر صفر دارد، اجرا نمی‌شود و البته خطایی هم صادر نمی‌شود و سایر سطرها اجرا خواهند شد. خروجی این برنامه شکل زیر است:

تصویر نادیده گرفتن سطری که خطا دارد



مثال ۳ Error Handling – حذف شیت اکسل

می‌دانم که تا به حال دستور حذف یک شیت در اکسل را نگفته‌ایم اما کار سختی نیست و چون مثال جالبی است آنرا در اینجا مطرح کردیم. صورت مساله ما این است که باید شیتی به نام Test را حذف کنیم اما مشکل این برنامه آن است که ممکن است شیتی به نام Test نداشته باشیم بخواهیم که آنرا حذف کنیم و در این دستور حذف شیت باعث بوجود آمدن خطایی خواهد شد.

بنابراین با دستور On Error Resume Next به VBA می‌گوییم که اگر دستور حذف شیت Test خطا داد، بی خیال آن خطا شو! و آن خطا را نادیده بگیر و به سراغ سطرهای بعدی برو و آنها را اجرا کن.

```
Sub Chapter6_Ex10()

On Error Resume Next
    Worksheets("Test").Delete

End Sub
```

نکته : لازم نیست نگران دستوراتی باشید که به شما آنها را نگفته‌ام، در فصل‌های آینده این کتاب قطعا تک به تک این دستورات را برای شما شرح خواهم داد.

مروری بر ابزارهای Debugging در VBA

زبان VBA و محیط VBE برای برنامه نویسان ابزارهای گوناگونی جهت یافتن خطاها را پیش بینی کرده اند و تا کنون ما ابزارهای زیر را معرفی کرده ایم:

- دستور Debug.Print
- کلید F8
- ابزار Watch برای مشاهده مقادیر یک متغیر در هنگام اجرا
- استفاده از ToolTip برای مشاهده مقادیر یک متغیر در هنگام اجرا

در ادامه می‌خواهیم چند تکنیک دیگر را نیز یاد بگیریم.

دستور Stop

بعد از اجرای برنامه، استفاده از این دستور باعث توقف برنامه اجرا می‌شود و سپس حالت Deubg را خواهیم داشت . در حالت Debug می‌توانیم از مقادیر متغیرها مطلع شویم:

در برنامه زیر یک عدد تصادفی تولید کرده‌ایم و سپس اعداد ۱ تا ۱۰۰۰ را بر این عدد تصادفی تقسیم کرده‌ایم. حال در دو جای برنامه دستور Stop را قرار می‌دهیم تا هنگامی که برنامه به آن سطر رسید، متوقف شود تا ما امکان آنرا بیابیم که مقادیر متغیرها را بررسی کنیم.

تصویر دستور Stop و مشاهده متغیر با ToolTip

```
Sub Chapter6_Ex11()
Dim x As Long

  x = Rnd() * 100
  x = 76
Stop

For counter = 1 To 1000
    If counter / 2 = x Then Stop
Next

End Sub
```

نکته ۱: تابع Rnd یک عدد «تصادفی» بین 0 تا 1 تولید می‌کند و اگر آن را در 100 ضرب کنیم و اعشار آنرا حذف کنیم ، یک عدد طبیعی تصادفی دو رقمی خواهیم داشت.

نکته ۲ : فکر کنم که برای شما هم بدیهی است که حتما باید قبل از تحویل نهایی برنامه، Stop ها را پاک کنید.

نکته ۳: بعد از Stop شدن می‌توانیم برای ادامه برنامه کلید F5 و یا برای اجرا سطر به سطر کلید F11 را بزنیم.

یادآوری:

من از واژه عدد «طبیعی» استفاده کردم و برای مرور و یادآوری تعریف عدد «طبیعی» و سایر تعاریف را برای شما اینجا خلاصه می‌کنم.

- به اعداد ۱ و۲ و ۳ و ... اعداد طبیعی گفته می‌شود.
- به اعداد ۰ و ۱ و ۲ و ۳ و.. اعداد حسابی گفته می‌شود.
- به اعداد ... و ۳- و ۲- و ۱- و ۰ و ۱ و ۲ و ۳ و ... اعداد صحیح گفته می‌شود.
- اعداد گویا یعنی همه اعداد صحیح به همراه همه اعداد کسری
- اعداد گنگ و اعدادحقیقی را خودتان لطفا جستجو کنید.
-

استفاده از Break Point

نقاط Break دقیقا شبیه همان دستور Stop هستند و هر جایی که Break قرار داده باشیم، برنامه در آنجا متوقف و حالت Debug فعال می‌شود. با زدن کلید F5 برنامه تا Break Point بعدی اجرا خواهد شد.

برای گذاشتن Break Point از کلید F9 استفاده می‌شود و یا می‌توانید از منوی Debug → Toggle Break Point را انتخاب نمایید.

تصویر Break Poin

```
Sub Chapter6_Ex12()
Dim x As Long
    x = Rnd() * 100

Debug.Print x
Debug.Print 1 / x
Debug.Print x * x

End Sub
```

تذکر ۱ : با کلید بر روی حاشیه کناری قسمت کد، که در شکل زیر نمایش داده شده است، می‌توانیم یک Break Point ایجاد کنیم و برای پاک کردن Break Point ها ، کافی است که بر روی نقطه قرمز رنگ، کلیک کنیم.

تصویر ایجاد Break Point با کلیک بر روی حاشیه کناری

```
Sub Chapter6_Ex12()
Dim x As Long
    x = Rnd() * 100

Debug.Print x
Debug.Print 1 / x
Debug.Print x * x

End Sub
```

فصل هفتم – پروسیجرها و توابع

در این فصل در مورد پروسیجر ها بسیار کلی و خلاصه صحبت می‌کنیم و سپس به توابع خواهیم پرداخت و جزئیات بسیاری را در خصوص توابع خواهیم گفت.

بهتر است که اینجا بگویم که پروسیجرها و توابع قواعد و رفتارهای مشابه زیادی را دارند و حتی می‌توان با کمی اغماض آنها را یکی دانست.

پروسیجرها و برنامه نویسی ساخت یافته

طبیعی است که در روزهای اول برنامه نویسی، شما برنامه های کوتاهی بنویسید و کارهایتان را با آن برنامه کوتاه انجام دهید، اما به تدریج کارهای بیشتری را با کد نویسی انجام دهید و از این به بعد برنامه های شما بزرگتر و پیچیده‌تر خواهند شد.

حال باید ما خودمان را برای مدیریت برنامه های بزرگ تری مثلا یک برنامه ۱۰۰ یا ۲۰۰ خطی آماده کنیم و همانطور که قبلا در بحث «Structured programming» گفتیم باید یک برنامه بزرگ را شما به برنامه های کوچکی تقسیم کند.

در واقع ما همان کاری را می‌کنیم که در دنیای واقعی هم داریم. یک ابزار پیچیده را از قطعات کوچکتری می سازیم تا اگر یک قطعه خراب شد ، لازم نباشد همه قطعات را دور بریزیم و فقط قطعه خراب را تعویض کنیم و اگر خواستیم که یک قطعه را ارتقاء دهیم، لازم نباشد کل آن ابزار را از نو بسازیم و فقط یک قطعه خاص را تغییر خواهیم داد.

در دنیای برنامه نویسی هم ما دقیقا اینکار را انجام می دهیم یعنی یک برنامه بزرگ را به قطعات کوچکتری تقسیم میکنیم . بگذارید یک مثال ساده بزنم.

فرض کنیم که کل اطلاعات فروش شرکت را دارید و قرار است به مدیر هر بخشی گزارشی از فروش کالاهای تولید شده توسط آن بخش را بدهید. حال این برنامه را می توانیم به پروسیجرهای زیر تقسیم کنیم:

۱- خواندن اطلاعات کل فروش شرکت از دیتابیس اصلی
۲- فیلتر کردن اطلاعات بر اساس کالاهایی که هر واحد تولید می کند
۳- آپدیت کردن گزارش ها ، چارت ها و ....
۴- ساخت فایل PDF گزارش ها
۵- ایمیل کردن فایل PDF گزارش ها

و در نهایت همه پروسیجرها را توسط یک پروسیجر اصلی، اجرا کنیم:

```
Sub Main()
      Read_Data_Base
      Filter_Data
      Updata_Report
      Send_Email
End Sub
```

با اینکار نظم و ساماندهی برنامه ما بهتر می شود. اگر خواستیم یک نیاز اضافه کنیم به راحتی امکان پذیر است و لازم نیست که کل برنامه را عوض کنیم و اگر لازم شد قسمتی بهبود یابد، فقط کافی است که آن پروسیجر را تغییر دهیم.

یادآوری مطالبی قبلی

در فصل دوم ابتدا پروسیجر را تعریف کردیم و گفتیم که یک پروسیجر را می‌توان از چندین روش اجرا کرد:

۱- کلید F5
۲- دکمه Run
۳- منوی Run
۴- از داخل Excel و پنجره ماکروها
۵- از طریق تعریف یک کلید میانبر
۶- از داخل یک پروسیجر

همچنین در فصل دوم دیدیم که نامگذاری پروسیجر ها قواعدی دارند به عنوان مثال نمی‌توانیم از Space استفاده کنیم و نام پروسیجر نمی‌تواند با یک عدد شروع شود. از کلمات و دستورات استاندارد VBA نمی توانیم در نامگذاری استفاده کنیم.

در ضمن در فصل سوم در خصوص Scope یا میدان دید متغیرها نیز مطالبی را به شما گفتیم. حال می‌خواهم به شما بگویم که پروسیجرها نیز دارای Scope هستند.

میدان دید پروسیجرها

پروسیجرها در ماژول‌ها نوشته می‌شوند و در اینجا می‌خواهیم توضیح کوتاهی دهیم که آیا با دستور Call می‌توانیم سایر پروسیجرها را که در سایر ماژول‌ها هستند را اجرا کنیم.

تعریف پروسیجر از نوع Public

تمامی پروسیجرهایی که تا کنون نوشته‌ایم از نوع Public هستند و اگر پروسیجری Public باشد از هر پروسیجر دیگری، حتی اگر در یک ماژول دیگر باشد، قابل اجرا (call کردن) است.

به عنوان مثال در Module1 یک پروسیجر به نام Proc1 داریم. حال در Module2 می‌توانیم این پروسیجر را به شکل زیر اجرا کنید.

```
Sub Proc2()

 Call Proc1

End Sub
```

همچنین اگر مایل باشید، می‌توان نام ماژول را در ابتدا نام پروسیجر نوشت:

```
Call Module1.Proc1
```

در ضمن اگر مایل باشیم می توانیم کلمه Public را در جلوی نام پروسیجر های که میخواهیم Public باشند بنویسیم و در هر حال اگر هم ننویسیم باز آن پروسیجر Public است .

```
Public Sub Test()
      [Statements]
End Sub
```

تذکر : واژه Public را به عمومی و یا سراسری ترجمه کرد.

تعریف پروسیجر از نوع Private

اگر پروسیجری از نوع Private (خصوصی) باشد، توسط پروسیجرهایی قابل اجراست که در **همان** ماژول باشند. پروسیجر هایی که در ماژول های دیگری هستند، از وجود این پروسیجر کاملا بی اطلاع می باشند و نمی توانند آنرا اجرا کنند.

یک پروسیجر از نوع Private به شکل زیر تعریف می‌شود:

```
Private Sub Test()
      [Statements]
End Sub
```

نکته : اگر پروسیجری Private باشد از پنجره ماکرو Excel قابل دیدن و اجرا کردن نیست.

توجه: اگربپرسید که چه مواقعی باید یک پروسیجر از نوع Private باشد و چرا همیشه همه پروسیجرها را از نوع Public تعریف نکنیم، من پاسخ خیلی علمی و درستی را در دنیای VBA ندارم که به شما بدهم. شاید یکی از پاسخ‌هایی که به شما بدهم آن باشد که اگر پروسیجری Private باشد، کاربر دیگر در پنجره ماکروها آنرا نمی بینید و اینکار باعث می شود که کمی پنجره ماکروها خلوت تر شود. این راه هم اضافه کنم که در کتاب های مرجع برنامه نویسی VBA نیز تنها به تعریف Private و Public اکتفا کرده اند و هیچ اشاره ای به سناریوهای کاربردی استفاده از آنها نشده است.

در ضمن در سایر زبانهای برنامه نویسی مانند C# اینکه کدها و متغیرها را Private و ... کنیم دلایل جدی دارد که آن دلایل در حوزه VBA خیلی مصداق پیدا نمی کنند.

دادن ورودی به پروسیجر

هنگامی که یک پروسیجر می‌سازید در جلوی نام آن علامت پرانتز می آید و حال می خواهیم دلیل آن علامت پرانتز را بدانیم. این پرانتز یعنی اینکه پروسیجر می تواند ورودی بگیرد و سپس روی آن ورودی کاری را انجام دهد.

فرض کنید که یک می‌خواهیم پروسیجری بسازیم که یک عدد را بگیرد و سپس مکعب (توان سه) آن عدد را برای ما چاپ کند.

برای این منظور پروسیجر زیر را می سازیم که یک عدد به عنوان ورودی می گیرد:

```
Sub Cube(x As Double)
    Debug.Print "Answer :" & x * x * x
End Sub
```

حال کافیست که این پروسیجر را فراخوانی (Call) کنید و برای اجرای آن باید یک پروسیجر دیگری داشته باشیم و اگر پروسیجر زیر را اجرا کنید، باعث چاپ شدن مکعب عدد 3 خواهد شد:

```
Sub Chapter7_Ex1()
  Cube (3)
End Sub
```

نکته ۱: اگر پروسیجری ورودی بگیرد، شما نمی توانید آنرا با هیچ یک از سایر روش هایی که گفته شده است (مانند F5 و اجرا از داخل پنجره ماکرو) اجرا کنید. زیرا آن پروسیجر از شما ورودی می-خواهد و فقط می توانید از طریق یک پروسیجر دیگری به آن ورودی بدهید و آنرا اجرا کنید.

کاربرد پروسیجرها با ورودی

واقعیت آن است که دادن ورودی به پروسیجر خیلی ویژگی خاصی نیست، زیرا ما با تعریف یک متغیر از نوع Public یا Module Level به راحتی می‌توانستیم یک مقدار را بین دو پروسیجر به اشتراک بگذاریم بدون آنکه لازم باشد پروسیجر یک ورودی بگیرد. به عنوان مثال می‌توانیم مثال قبلی را اینگونه هم پیاده سازی کنیم:

تصویر تعریف متغیر و استفاده از آن در یک پروسیجر دیگر

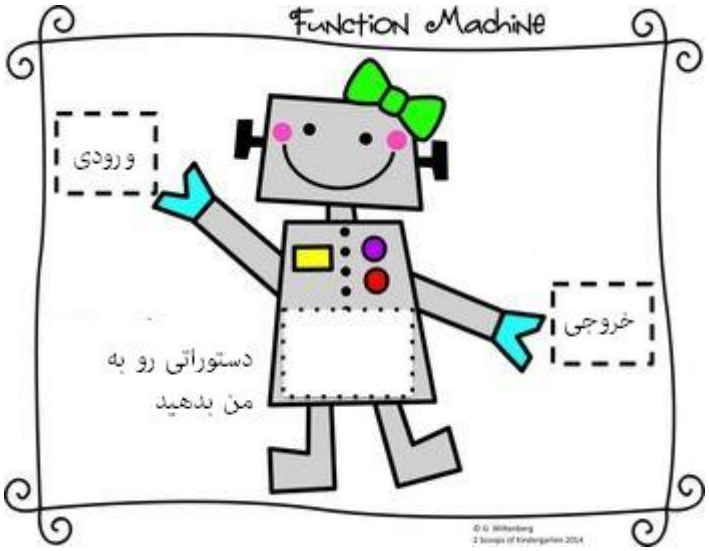
```
Dim x As Double

Sub Chapter7_Ex2()
    x = 3
    Call Cube
End Sub

Sub Cube()
    Debug.Print "Answer :" & x * x * x
End Sub
```

باید بگویم که استفاده از پروسیجرهایی که ورودی می‌گیرند آنچنان متداول نیست و برای این منظور ما از توابع استفاده می‌کنیم. توابع علاوه بر اینکه می توانند ورودی بگیرند ، می توانند به ما «خروجی» هم بدهند.

با توابع آشنا شوید

شما در اکسل با توابع (Functions) کار کرده اید و می دانید که توابع مانند یک ماشین، ورودی می‌گیرند و سپس روی ورودی ها کاری/فرآیندی/تغییراتی را انجام می دهند و در نهایت یک خروجی خواهند داشت.

تصویر عملکرد یک تابع



به عنوان مثال تابع Sum اکسل را در نظر بگیرد، از شما ورودی می گیرد و سپس تمامی ورودی ها را جمع می زند و در نهایت حاصل جمع را به شما خواهد داد.

ما در VBA با دو هدف زیر تابع هایی را می نویسیم:

۱- اضافه کردن یک تابع جدید به توابع اکسل
می‌دانیم که اکسل تابعی که بتواند عدد رو به حروف تبدیل کند را ندارد. بنابراین خود ما یک تابع برای اینکار در VBA خواهیم نوشت تا بتوانیم در سلولهای اکسل از آن تابع مشابه سایر توابع استاندارد استفاده کنیم.

۲- به منظور پیاده سازی روش «برنامه نویسی ساخت یافته »
همانطور که در ابتدای بحث پروسیجر ها گفتیم، در واقع یک برنامه بزرگ را به بخش های کوچکتری تقسیم می‌کنیم و معمولا این بخش ها را با توابع ایجاد می‌کنیم.

نکته : به توابع جدیدی که در Excel می‌نویسیم اصطلاحا Custom Function و یا User Defined Function و یا UDF می‌گویند.

مقایسه توابع و پروسیجر ها

قبل از شروع بحث توابع، بهتر است نگاهی به تفاوت‌های توابع و پروسیجرها بیندازیم:

۱- پروسیجر ها نمی توانند خروجی داشته باشند اما توابع خروجی دارند.
۲- از توابع می‌توان در سلول‌های اکسل استفاده کرد اما از پروسیجر ها نمی توانیم.
۳- می توانیم پروسیجرهای که ورودی ندارند را مستقیما اجرا کنیم (مثلا با کلید F5 و یا از طریق پنجره ماکرو) اما توابع اینگونه نیستند. در واقع حتما توابع باید فراخوانی (Call) شوند.
۴- می‌توانیم توابع را در سلول‌های اکسل بکار ببریم اما پروسیجرها را باید اجرا کرد.

نکته : عملکرد توابع و پروسیجر ها بسیار شبیه هم هستند و تفاوت های اندکی دارند که در زبانهای قدرتمندی مانند C# هر دوی آنها با هم ادغام شده اند و به نام متد (Method) شناخته می شوند.

کلیات ساخت و استفاده از یک تابع

برای اینکه یک تابع را بسازیم (تعریف کنیم) مانند پروسیجر ها عمل می کنیم و فقط واژه Sub را با واژه Function جایگزین خواهیم کرد.

در کد زیر یک تابع (بدون ورودی) به نام Test تعریف کرده‌ایم:

```
Function Test()
      Statements
End Function
```

نکته ۱: حتما نام توابع را بیش از چهار حرف بگذارید تا با نام سلولهای اکسل تداخل پیدا نکند. مثلا نام تابعی را Tax15 نگذارید زیرا Tax15 نام یکی از سلولهای اکسل است.

و اگر تابع ما ورودی هم بگیرد کافی است که در داخل پرانتز به ازای هر ورودی یک را تعریف کنیم.

در کد زیر تابع Test دو ورودی به نام‌های x و y لازم دارد:

```
Function Test(x , y)
      Statements
End Function
```

بگذارید من شکل کلی توابع را در اینجا برای شما بنویسیم :

```
[Public | Private ] [Static] Function name [(arglist)] [As type]
      [statements]
      [name = expression]
      [Exit Function]
      [statements]
      [name = expression]
End Function
```

ما با اکثر این حالت هایی که یک تابع می تواند داشته باشد، کار خواهیم کرد و کاربردهای آن را خواهیم دانست و تاکید میکنم که توابع در VBA کاربردهای بسیاری دارند و حتما باید بر آنها مسلط باشید.

Call کردن توابع

برای استفاده از یک تابع نمی‌توانیم مانند پروسیجر ها آنرا با کلید F5 مستقیما Run کنیم. در واقع باید یک تابع را صدا بزنیم که اصطلاحا آنرا Call کردن (صدا زدن) می‌گوییم.

برای Call کردن یک تابع ما چندین راه داریم:

۱- Call کردن تابع از داخل یک پروسیجر دیگر
۲- استفاده از Immediate
۳- در داخل یکی از سلول‌های اکسل

در مثال بعدی با این روش‌ها به صورت عملی آشنا خواهید شد.

مثال ۱ توابع - تابعی بدون ورودی و بدون خروجی

در اکثر نرم افزارها شما از منوی Help→ About می توانید در خصوص آن نرم افزار توضیحاتی را مشاهده کنید. حال می خواهم که تابعی بنویسم تا در مورد فرساران، این کتاب، سایت و .. اطلاعاتی را نمایش دهد:

```
Function About_Farsaran()
    Dim txt As String
    txt = "Excel VBA Programming v1"
    txt = txt & vbNewLine & "Author : Farshid Meidani"
    txt = txt & vbNewLine & "web site: www.farsaran.com"

    MsgBox txt
End Function
```

شرح برنامه: این تابع نه ورودی دارد (در داخل پرانتز هیچ چیزی ننوشته ایم) و نه خروجی. در ضمن آنکه اسم مناسبی برای آن انتخاب کرده‌ایم.

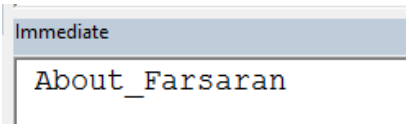
برای فراخوانی (یا بگوییم صدا زدن یا Call) این تابع می توانیم از سه روش زیر استفاده کنیم:

۱- اجرا از یک پروسیجر
کافی است که در هر پروسیجری نام تابع را (بدون پرانتز) بنویسیم:

```
Sub Chapter7_Ex3()
  About_Farsaran
End Sub
```

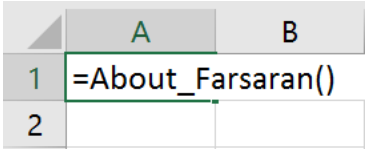
۲- می توانیم نام تابع را (بدون پرانتز) در Immediate بنویسیم و Enter را بزنیم:

تصویر فراخوانی تابع از Immediate



۳- و می توانید در هر سلولی از اکسل نام تابع را (حتما به همراه پرانتز) نویسید:

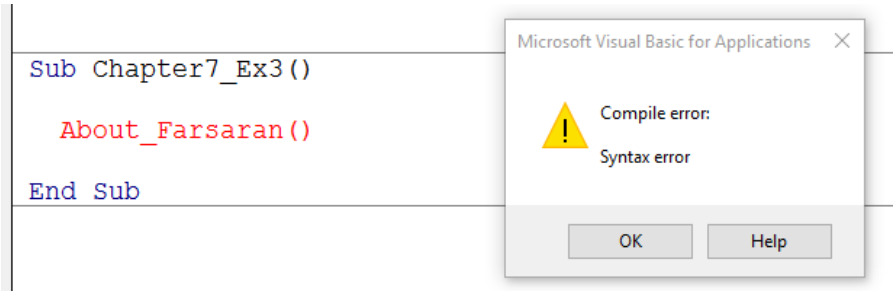
تصویر استفاده از تابع در یک سلول



تمرین ۱: این تابع را در یکی از سلول اکسل بنویسید و سپس آن سلول را در ۱۰ سلول دیگر کپی کنید، چه اتفاقی می افتد؟

نکته ۱: اگر نام تابع را در یک پروسیجر و یا Immediate با علامت پرانتز بنویسید، VBA فرض می کند که این تابع باید یک خروجی داشته باشد و خطای زیر را خواهید دید:

تصویر خطای استفاده از پرانتز در جلوی نام تابع
بدون آنکه تکلیف خروجی را مشخص کرده باشیم



در واقع از شما می خواهد که از آن خروجی استفاده کنید. مثلا خروجی را در یک متغیر بگذارید و یا آن خروجی را چاپ کنید:

```
Sub Chapter7_Ex4()

  x = About_Farsaran()
  Debug.Print About_Farsaran()

End Sub
```

میدان دید توابع

بگذارید در مورد تابع About_Farsaran یک سوال از شما بپرسم. حدس می‌زنید که آیا می‌توانید از این تابع در یک پروسیجر که در ماژول دیگری است، استفاده کنیم؟

در واقع می‌توانیم به زبان علمی تر از شما بپرسم که میدان دید (Scope) این تابع چیست؟

اگر میدان دید توابع را صراحتا تعریف نکنید به طور پیش فرض از نوع Public هستند یعنی در هر پروسیجری از هر ماژولی قابل Call هستند.

البته اگر دوست داشته باشید می‌توانید واژه Public را در ابتدای خط تعریف تابع بنویسیم:

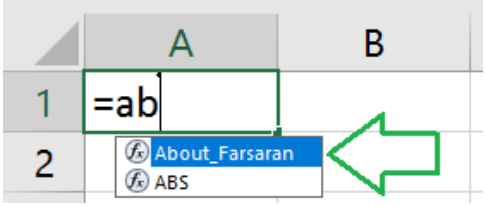
```
Public Function About_Farsaran()
```

اگر هم که بخواهید که Scope یک تابع را به یک ماژولی که تابع در آن نوشته شده است محدود کنید کافی است که کلمه Private را در ابتدای نام آن بگذارید و فقط پروسیجر هایی از این تابع اطلاع دارند که در همین ماژول باشند:

```
Private Function About_Farsaran()
```

نکته: توابع Public در لیست توابع اکسل ظاهر می شوند اما توابع Private را در این لیست نخواهیم داشت:

تصویر نمایش نام یک تابع Public در لیست توابع اکسل



تعریف خروجی یک تابع

می خواهیم که یک تابع به ما خروجی بدهد. شکل و یا دستور گرفتن خروجی از تابع شاید در ابتدا کمی عجیب به نظر برسد (که در واقع عجیب هم هست) اما به آن عادت می‌کنیم.

می‌دانیم که باید هر تابعی یک نام داشته باشد مثلا اسم تابعی را می توانیم MyName بگذاریم و اینگونه تابع MyName را تعریف می‌کنیم:

```
Function MyName()
```

و انتهای تابع هم با دستور End Function مشخص می‌شود و در بدنه تابع دستورات VBA را می نویسیم.

حال اگر بخواهیم که تابع به ما خروجی را بدهد، باید خروجی را در متغیری به نام، نام تابع بگذاریم. یعنی بنویسیم:

```
MyName = [expression]
```

در اینجا منظور ما از expression یعنی هر عبارتی که مقداری داشته باشد. اگر بخواهیم خروجی تابع MyName عدد 10 باشد، باید بنویسیم:

```
MyName = 10
```

و یا خروجی Farshid :

```
MyName = "Farshid"
```

و یا حتی یک محاسبه:

```
MyName = Len("Farshid") + 2 - Len("Fazad")
```

مثال ۲ توابع - اسم من

تابعی بنویسید که خروجی آن نام شما باشد.

پاسخ تابع زیر است:

```
Function MyName()
  MyName = "Farshid"
End Function
```

خیلی ساده بود نه؟ یک تابع به نام MyName تعریف کردیم و سپس مقدار خروجی این تابع را در MyName گذاشتیم.

اما بگذارید کمی آنرا حرفه ای تر هم کنیم. می دانیم که خروجی تابع MyName قرار است از نوع متنی (text) باشد، پس به دلایلی که قبلا در بحث متغیرها گفتیم، بهتر است که نوع تابع را مشخص کنیم. برای اینکار به شکل زیر عمل می‌کنیم:

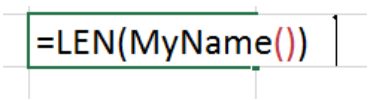
```
Function MyName() As String
  MyName = "Farshid"
End Function
```

و اگر بخواهید از این تابع استفاده کنید، کافی است که در یک پروسیجر بنویسیم:

```
MsgBox MyName()
```

و البته که می توانید این تابع را با سایر توابع اکسل و یا VBA ترکیب کنید:

تصویر استفاده از تابع MyName در تابع LEN



مثال ۳ توابع - فقط عدد طبیعی

تابعی بنویسید که یک عدد مثبت را به عنوان ورودی از کاربر بگیرد.

```
Function GetInt() As Long

    Do
        GetInt = InputBox("Give me a Positive Number :")
    Loop While GetInt <= 0

End Function
```

شرح برنامه: خروجی تابع از نوع Long است و اگر کاربر عددی مثبت را ندهد، مجدد سوال از او پرسیده می‌شود تا سرانجام یک عدد مثبت را وارد نماید.

نکته : در این برنامه خواستم به شما نشان بدهم که توابعی که با VBA می نویسید می تواند از توابع اکسل کامل تر باشد، مثلا این تابع به کاربر یک پنجره را نمایش می دهد در صورتی که هیچ تابعی در اکسل یک پنجره به کاربر نمایش نمی دهد.

ورودی توابع

حال که توانستیم از توابع خروجی بگیریم، چرا به تابع ورودی ندهیم که بر اساس آن ورودی‌ها بتوانیم کارهای جالبی را انجام دهیم و خروجی های متنوع تری را بگیریم.

دقیقا تعریف ورودی یک تابع مانند تعریف ورودی یک پروسیجر است یعنی در داخل پرانتز یک متغیر تعریف می‌کنیم و آن متغیر ورودی تابع ما خواهد بود و در بدنه تابع می توانیم از آن استفاده کنیم.

نکته : به ورودی توابع اصطلاحا Argument (آرگومان) می گویند. همچنین اصطلاحا می‌گوییم که ورودی‌ها را به تابع Pass می دهیم.

مثال ۴ توابع - مکعب

فرض کنید که می‌خواهیم تابعی بنویسیم که مکعب یک عدد را محاسبه کند:

```
Function Cube2(number) As Double
    Cube2 = number ^ 3
End Function
```

شرح : نام تابع Cube2 است و یک ورودی به نام Number دریافت می کند. خروجی تابع از نوع Double است و خروجی تابع بر اساس ورودی که گرفته است با دستور زیر محاسبه می شود:

```
Cube2 = number ^ 3
```

برای استفاده از این تابع کافی است که در یک پروسیجر مثلا بنویسید:

```
Debug.Print Cube2(25)
```

در ضمن بهتر است که نوع ورودی تابع را (به همان دلایل قبلی) مشخص کنیم، بنابراین اینگونه می نویسم:

```
Function Cube2(number As Double) As Double
    Cube2 = number ^ 3
End Function
```

نکته ۱: شما می‌توانید از این تابع مانند سایر توابع اکسل استفاده کنید و یک سلول را به عنوان ورودی به آن بدهید. آنرا مانند سایر توابع کپی کنید :

تصویر استفاده از تابع و مقدار یک سلول به عنوان ورودی

Cube:	2	=Cube2(C7)
	3	27
	4	64
	5	125

نکته ۲: چون در فایل مثال های این فصل یک پروسیجر به نام Cube ساخته بودیم، دیگر نمی-توانستیم در همان فایل یک تابع به نام Cube بسازیم، نام تابع را Cube2 گذاشتیم تا از تداخل نام ها جلوگیری کنیم.

مثال ۵ توابع - طول وتر مثلث

برنامه ای بنویسید که سه ضلع یک مثلث را بگیرد و سپس مساحت مثلث را از فرمول زیر (فرمول هرون) محاسبه کنید:

$$S = \sqrt{P(P-a)(P-b)(P-c)}$$

که در فرمول بالا مقدار P برابر است با:

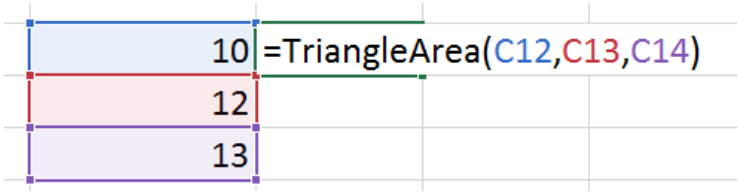
$$P = \frac{a+b+c}{2}$$

```
Function TriangleArea(a As Double, b As Double, c As Double) As Double
Dim p As Double

p = (a + b + c) / 2
TriangleArea = Sqr(p * (p - a) * (p - b) * (p - c))

End Function
```

توضیح خاصی ندارد. دیدیم که می توانیم به سادگی چندین ورودی بگیریم و روی آنها محاسبه ای را انجام دهیم.

نکته: تابع Sqr همان جذر است.

تصویر استفاده از فرمول مساحت در اکسل



ورودی اختیاری توابع

در توابع اکسل شما بارها با ورودی‌های اختیاری در توابع مواجه شده‌اید مثلا تابع LEFT دارای دو ورودی است که اولین ورودی یک متن است و اجباری است و ورودی دوم یک عدد است و اختیاری می باشد.

همانطور که در زیر مشاهده می کند اگر ورودی دوم تابع LEFT را تعیین نکنید، فقط یک حرف از ابتدای متن را به ما می دهد.

```
LEFT("Farsaran") → F
LEFT("Farsaran", 2 ) → Fa
LEFT("Farsaran", 4 ) → Fars
```

ما نیز به سادگی می‌توانیم ورودی یک تابع را از نوع اختیاری کنیم. کافی است که از کلمه Optional قبل از نام ورودی تابع استفاده کنیم.

نکته: به ورودی اختیاری اصطلاحا Optional می‌گویند.

مثال ۶ توابع - فصل های سال

برنامه‌ای بنویسید که یک عدد را به عنوان ورودی بگیرد و سپس نام فصل سال را به انگلیسی به ما بدهد و اگر مایل بودیم بتوانیم نام فصل سال را به صورت پینگلیش (فارسی) هم داشته باشیم.

```
Function SeasonName(num As Long, Optional farsi As Long) As String

If farsi = 0 Then
    Select Case num
        Case 1: SeasonName = "Sprig"
        Case 2: SeasonName = "Summer"
        Case 3: SeasonName = "autumn"
        Case 4: SeasonName = "Winter"
    End Select
ElseIf farsi = 1 Then
    Select Case num
        Case 1: SeasonName = "Bahar"
        Case 2: SeasonName = "Tabestan"
        Case 3: SeasonName = "Paeez"
        Case 4: SeasonName = "Zemestan"
    End Select
End If

End Function
```

شرح برنامه: این برنامه دو ورودی دارد و ورودی دوم اختیاری است. هر دو ورودی عدد هستند.

- نام ورودی اول Num است و شماره فصل سال خواهد بود.
- نام ورودی دوم farsi است و اگر عدد 1 باشد یعنی باید نام فصل فارسی باشد و چون می‌خواستیم اختیاری باشد از کلمه Optional قبل از نام آن استفاده کرده ایم.

می دانیم که پیش فرض متغیرهای عددی،صفر است و اگر کاربر ورودی دوم را ندهد، یعنی صفر است و بر همین مبنا IF را نوشتیم.

```
IF farsi = 0 Then
      …
ElseIF farsi = 1 Then
      …
End If
```

سوال ۱: این تابع را با ورودی های اشتباه امتحان کنید. چه خروجی خواهید داشت؟

سوال ۲: اگر ورودی اشتباه به یک تابع استاندارد اکسل بدهید، خروجی شما چه خواهد بود؟

تابع با تعداد ورودی زیاد

تابع Sum اکسل را در نظر بگیرید، چند ورودی می‌تواند قبول کند؟ ۱۰ ورودی یا ۱۰۰ ورودی؟ من به شما می‌گویم، ۲۵۵ ورودی!

توابعی هستند که تعداد زیادی ورودی را قبول می کنند و در اینجا می خواهم فقط یک نمونه را به شما معرفی کنم و چون هنوز با Array ها (آرایه یا ماتریس ها) آشنا نشده ایم ، نمی توانیم این بحث را به شکل مفصلی باز کنیم و فقط جهت دیدن یک نمونه برای شما مثالی را انجام خواهم داد.

مثال ۶ توابع - ادغام متن سلول‌ها

آیا تابع CONCATENATE اکسل را می شناسید. این تابع چند ورودی از ما می گیرد و سپس آنها را به هم می چسباند. به عنوان مثال فرمول زیر را در نظر بگیرید :

تصویر تابع CONCATENATE

	A	B	C	D	E
1	Hi	My	Name	is	Farshid
2	=CONCATENATE(A1,B1,C1,D1,E1)				

که خروجی آن می‌شود

```
HiMyNameisFarshid
```

و این تابع دو محدودیت دارد. اول اینکه باید تک به تک سلول ها را به صورت ورودی به آن بدهیم یعنی اگر ۱۰۰ سلول را بخواهید با هم ترکیب کنید، باید آدرس همه ۱۰۰ سلول را به صورت ورودی به این تابع بدهید و نمی توانید مانند تابع SUM اینگونه بنویسید :

```
CONCATENATE( A1:A100)
```

دوم آنکه نمی توانیم یک جداکننده تعریف کنیم. یعنی مثلا بگوییم که بین همه متن ها یک علامت کاما یا Space اضافه کن.

بنابراین می‌خواهیم تابعی بنویسیم که :

۱- تعداد ورودی های آن محدود نباشد.
۲- ورودی ها را به هم بچسباند.
۳- بین متن ها بتوانیم جداکننده دلخواه اضافه کنیم.

```
Function StringBuilder(Seprator As String, ParamArray args() As Variant) As String

Dim txt As Variant

For Each txt In args

If TypeName(txt) = "Range" Then

  For Each x In txt
    StringBuilder = StringBuilder & x & Seprator
  Next x

Else
    StringBuilder = StringBuilder & txt & Seprator
End If

Next txt

StringBuilder = Left(StringBuilder, Len(StringBuilder) - Len(Seprator))

End Function
```

شرح برنامه: همانطور که گفتم چون با آرایه‌ها آشنا نشده ایم نمی توانیم تکنیک های بکار رفته در این برنامه را شرح دهیم. اما کلیات این برنامه اینگونه است:

۱- ورودی اول برنامه جداکننده ای دلخواه و از نوع متنی است.
۲- این تابع یک ورودی به نام args دارد و با عبارت ParamArray مشخص کردیم که این ورودی نامنتهای است و همچنین می‌تواند از «نوع‌های» مختلفی هم باشد.
۳- این تابع در نهایت همه ورودی ها را به هم می چسباند و بین آنها علامت جداکننده را خواهد گذاشت.

در فرمول زیر جداگاننده Space است و سپس به این تابع ۶ عدد ورودی از «نوع‌های» مختلف را پاس کردیم:

```
=StringBuilder(" ",A1:E1,A2:E2,"This","IS","A","String Builder")
```

مثال‌های کاربردی از توابع

از آنجایی که ساخت و استفاده از توابع در VBA بسیار مهم است، در این قسمت چندین مثال کاربردی برای شما خواهم آورد.

اگر چه ممکن است برخی از دستورات و تکنیک‌های بکار رفته در این مثال ها را هنوز نگفته باشیم، اما مطرح کردن آنها می تواند ایده های خوبی را به شما بدهد و برای جستجو و یادگیری آن دستورات قطعا می توانید از گوگل استفاده کنید.

مثال ۷ - تابعی برای تجزیه یک متن به حروف

در بسیاری از موارد باید یک متن را به حروفش تجزیه کنید، حال می خواهیم تابعی بنویسیم که یک متن را بگیرد و سپس حرف به حرف آن متن را چاپ کند:

```
Function SplitText(txt As String) As String
Dim i As Long

For i = 1 To Len(txt)
    Debug.Print Mid(txt, i, 1)
Next i

End Function
```

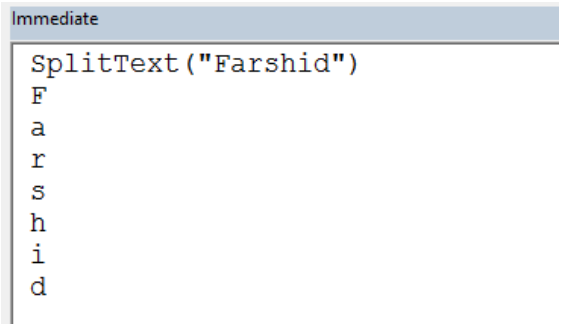
شرح برنامه: چون به حرف به حرف آن متن نیاز داریم پس باید یک حلقه به تعداد حروف آن متن بسازیم و تعداد حروف متن را با تابع Len محاسبه می‌کنیم:

```
For i = 1 To Len(txt)
```

حال باید تک به تک حروف این متن را جدا کنیم. برای اینکار از تابع Mid استفاده می‌کنیم و در هر بار، از حرف i ام متن ، یکی را جدا می‌کنیم:

```
Mid(txt, i, 1)
```

تصویر اجرای تابع در Immediate



نکته: این تابع بر روی هر ورودی کار می کند و بهتر است به جای کلمه آنکه بگوییم این تابع «حروف» را چاپ می کند، بهتر است بگوییم که این تابع «کاراکترهای یک متن» را چاپ می کند.

مثال ۸ - تابعی برای استخراج یک عدد از متن

می خواهیم یک عدد را از یک متن استخراج کنیم. این تابع بسیار کاربردی است مثلا در شرح سندهای حسابداری که یک عدد وجود دارد، لازم دارید که آن عدد را استخراج کنید.

از تکنیک تابع قبلی استفاده می‌کنیم ، یعنی کاراکتر به کاراکتر را تجزیه می‌کنیم و اگر کاراکتری از نوع عددی بود، آنها را در یک متغیر ذخیره می‌کنیم:

```
Function ExtractNumber(txt As String) As Long
 Dim i As Long
 Dim temp As String

    For i = 1 To Len(txt)

        If IsNumeric(Mid(txt, i, 1)) Then
                temp = temp & Mid(txt, i, 1)
        End If

    Next i

    ExtractNumber = temp

End Function
```

شرح برنامه: با تابع IsNumeric هر کاراکتر را بررسی می‌کنیم که آیا عدد است و اگر عدد بود، آن کاراکتر را به متغیر temp اضافه می‌کنیم.

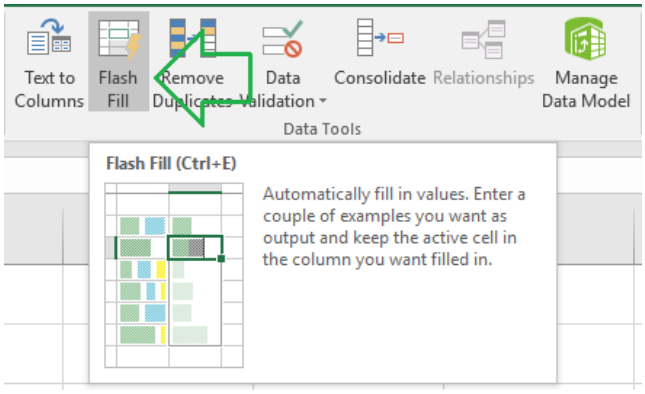
در نهایت متغیر temp شامل همه اعداد متن خواهد بود.

سوال ۱ : این تابع خیلی کامل نیست مثلا نمی تواند علامت منفی، علامت ممیز را تشخیص دهد. این تابع را کاملتر کنید تا بتواند علامت منفی اعداد را هم تشخیص دهد.

سوال ۲: آیا می توانید این تابع را طوری تغییر دهید که فقط اولین عدد از یک متن را فقط استخراج کند.

نکته ۱: از Excel 2013 به بعد ما ابزاری به نام Flash Fill اضافه شده است که می تواند اینکار یعنی جداسازی عدد از متن را انجام دهد. (این ابزار در Excel 2013 دارای خطایی بود که نمی توانست با متن های فارسی کار کند که این خطا در Excel 2016 مرتفع شده است):

تصویر ابزار Flash Fill در Data (tab)



مثال 9- حذف حروف صدا دار

تابعی بنویسید که حروف صدا دار انگلیسی (A,E,I,O,U) را از یک متن حذف کند.

```
Function RemoveVowels(Txt) As String
Dim i As Long
RemoveVowels = ""
For i = 1 To Len(Txt)
    If Not UCase(Mid(Txt, i, 1)) Like "[AEIOU]" Then
        RemoveVowels = RemoveVowels & Mid(Txt, i, 1)
    End If
Next i
End Function
```

شرح برنامه: ابتدا باید حرف به حرف متن را داشته باشیم و به همین منظور از تکنیک که در مثال تجزیه یک متن به حروف آموختیم، استفاده می‌کنیم.

سپس می‌خواهیم بدانیم که آن هر یک از آن حروف برابر یکی از مقادیر A یا E یا I یا O و یا U است. ما یک عملگر (Operator) خیلی جالب به نام Like در VBA داریم که توسط این عملگر دو متن را با هم مقایسه می‌کنیم.

اجازه بدهید در همین جا عملگر Like را بررسی کنیم و سپس به ادامه حل این مثال بپردازیم.

معرفی عملگر Like

گفتیم که ما عملگرهای متفاوتی را داریم مانند علامت مساوی، علامت ضرب و ... . Like یک عملگر است که برای مقایسه «شباهت» دو متن بکار می رود. فرض کنید دو متغیر متنی به نام Text1 و Text2 داریم و می توانیم از این عملگر اینگونه استفاده کنیم:

```
Text1 Like Text2
```

Like مانند سایر عملگرهای مقایسه ای ، پاسخ True یا False خواهد داشت و از نتیجه آن می‌توانیم اینگونه استفاده کنیم:

```
Debug.Print Text1 Like Text2
```

و یا در IF

```
IF Text1 Like Text2 Then Debug.Print "ok"
```

یک مثال ساده:

```
Sub Chapter7_Ex5()
Text1 = "IRAN"
Text2 = "Iran"
Debug.Print Text1 Like Text2
End Sub
```

و پاسخ این مقایسه البته که False است .

البته عملکرد عملگر Like بیش از اینهاست، بگذارید یک مثال بزنم اگر کسی بگوید می‌روم «شیراز» و فردا او به شما زنگ بزند و به شما بگوید که در «شهر شیراز» است ، آیا دروغ گفته است؟

قطعا خیر. زیرا از نظر شما واژه «شیراز» همان «شهر شیراز» است و بازهم اگر آن دوست شما بگوید که در «شهر شیراز بسیار زیبا » است باز هم حقیقت را گفته است.

زیرا همه این عبارات در کلمه «شیراز» با هم مشابهت دارند و اگر بخواهیم این عبارات در در VBA باهم مقایسه کنیم، از عملگر Like به شکل زیر استفاده می‌کنیم و پاسخ همه آنها True خواهد بود:

```
Sub Chapter7_Ex6()

Debug.Print "Shahr Shiraz" Like "*Shiraz"
Debug.Print "Shahr Shiraz ziba" Like "*Shiraz*"

End Sub
```

عملگر Like الگوها را متوجه می‌شود و کاراکتر «*» یعنی هر چند تعداد کاراکتر. بنابراین اگر بخواهیم بگوییم هر جلمه ای که در انتهای آن واژه Shiraz است، می نویسیم:

```
*Shiraz
```

و اگر بخواهیم بگوییم که در جلمه ای که کلمه Shiraz در آن وجود داشته باشد و هر چه می‌خواهد در ابتدا و یا انتهای جمله باشد، می نویسیم:

```
*Shiraz*
```

نکته ۱: می توانید کاراکتر «*» را بخوانید «هر چی». و در نتیجه *Shiraz* می شود، «هرچی باشد سپس Shiraz باشد و سپس هر چی باشد».

کاراکتر در الگو	شرح	الگو	نمونه‌های صادق
*	صفر یا هر تعداد کاراکتر	"a*a"	"aBBBa" "aa" "aBBBBBBBa"
?	فقط یک کاراکتر	"a2a"	"ana" "asa" "aba"
#	هر یک عدد	"0##2"	"0912" "0892" "0002"
[charlist]	هر یک کاراکتر در لیست	[a-c]	سه حرف a,b,c "a" "b" "z"
[!charlist]	هر یک کاراکتر که در لیست نیست	[!a-c]	بجز سه حرف a,b,c

مثال:

```
Dim MyCheck
MyCheck = "aBBBa" Like "a*a"            ' Returns True.
MyCheck = "F" Like "[A-Z]"              ' Returns True.
MyCheck = "F" Like "[!A-Z]"             ' Returns False.
MyCheck = "a2a" Like "a#a"              ' Returns True.
MyCheck = "aM5b" Like "a[L-P]#[!c-e]"   ' Returns True.
MyCheck = "BAT123khg" Like "B?T*"       ' Returns True.
MyCheck = "CAT123khg" Like "B?T*"       ' Returns False.
```

ادامه مثال ۹ - حذف حروف صدا دار

امیدوارم که حالا بتوانید بهتر منظور عبارت زیر را درک کنید:

```
UCase(Mid(Txt, i, 1)) Like "[AEIOU]"
```

ابتدا حروف را بزرگ می‌کند و سپس با عملگر Like بررسی می کند که آیا آن حرف، یکی از کاراکترهای لیست [AEIOU] نباشد و سپس آن حرف را در یک متغیر به سایر حروف می چسبانیم:

```
If Not UCase(Mid(Txt, i, 1)) Like "[AEIOU]" Then
        RemoveVowels = RemoveVowels & Mid(Txt, i, 1)
```

حال فرض کنید که کاربر یک ورودی عددی را به این تابع بدهد. چون ورودی این تابع باید یک متن باشد و اساسا اعداد حروف صدادار ندارد، بنابراین حرفه‌ای است که مانند خروجی تابع ما یک خطا شود.شاید فکر کنید که می‌توانید خروجی خطای تابع را اینگونه تعریف نمایید تا خطا را مشخص نمایید:

```
RemoveVowels = "#N/A"
```

اما اگر چه این خروجی از نظر بیننده ممکن است خطا تلقی شود، اما واقعیت آن است که این یک متن است و در Excel خطاها دارای ویژگی های خاصی هستند. مثلا یک خطای یک تابع باعث می شود که خروجی سایر توابع که از آن تابع استفاده می کنند، خطا شود و یا اینکه در اکسل می توانیم با تابع IFERROR خطاها را مدیریت کنیم و یا حتی در Conditional Formatting، خطاها را به صورت یک مقدار خاص، مدیریت کنیم.

بنابراین خروجی خطا، باید به گونه ای باشد که اکسل آنرا به عنوان **«خطای واقعی»** شناسایی کند و برای این که خطای استاندارد #N/A را تولید کنیم، از تابع CVErr به شکل زیر استفاده می‌کنیم:

```
RemoveVowels = CVErr(xlErrNA)
```

این تابع انواع خطاهای استاندارد اکسل را می تواند تولید کنید:

- xlErrDiv0 (for #DIV/0!)
- xlErrNA (for #N/A)
- xlErrName (for #NAME?)
- xlErrNull (for #NULL!)
- xlErrNum (for #NUM!)
- xlErrRef (for #REF!)
- xlErrValue (for #VALUE!)

و در نهایت این تابع به شکل زیر خواهد شد:

```
Function RemoveVowels(Txt) As Variant
Dim i As Long
RemoveVowels = ""

If WorksheetFunction.IsText(Txt) Then
    For i = 1 To Len(Txt)
        If Not UCase(Mid(Txt, i, 1)) Like "[AEIOU]" Then
            RemoveVowels = RemoveVowels & Mid(Txt, i, 1)
        End If
    Next i
Else
    RemoveVowels = CVErr(xlErrNA)
End If
End Function
```

نکته ۱: خروجی این تابع را دقت کنید که از نوع Variant شده است زیرا ممکن است این تابع خروجی متنی داشته باشد و یا اینکه خروجی آن چیزی بجز متن، یعنی یک خطا باشد.

نکته ۲: برای آنکه خروجی اعداد را تشخص دهیم از تابع IsText اکسل استفاده کرده‌ایم که در فصول آینده در مورد استفاده از توابع اکسل توضیح خواهیم داد.

فصل هشتم - Array

در این فصل در مورد Arrayها صحبت می‌کنیم. Arrayها را به فارسی آرایه‌ها می‌گوییم و منظور آرایه یک نوع متغیر است که «مجموعه‌ای» از مقادیر را در خودش نگهداری می‌کند.

تا به حال هر متغیری که تعریف کرده ایم ، فقط می‌توانست یک مقدار را در خودش نگهداری کند، اما یک Array می‌تواند ده ها مقدار را در خودش نگهداری کند. دقیقا مانند «یک» اتوبوس که می‌توانید ۳۰ نفر را در خودش جای دهد.

کار با آرایه‌ها در ابتدا ممکن است کمی سخت به نظر برسد و به همین دلیل این فصل را با حوصله بیشتری بخوانید و تمرین کنید. احتمال می‌دهم که حتی برخی از نکات آنرا دقیقا متوجه نشوید اما اشکالی ندارد، این بحث از مواردی است که انتظار می‌رود کم کم در ذهن شما شفاف شود و نباید در ابتدا از یادگیری آن ناامید شوید..

در ضمن باید بدانید که برخی از کارها فقط با آرایه‌ها قابل انجام است و همچنین در بسیاری از موارد توصیه می‌شود که برای سرعت بخشیدن به اجرای برنامه از آرایه‌ها استفاده گردد.

نکته ۱: ما در این فصل از واژه «آرایه» و «ماتریس» برای ترجمه Array استفاده خواهیم کرد.

نکته ۲: هر یک از اعضای یک آرایه دارای یک شماره منحصر بفرد هستند که به آن شماره Index یا «اندیس» می‌گوییم. (دقیقا مانند افرادی که در یک اتوبوس هستند و هر یک از آنها شماره صندلی خودش را دارد.)

تعریف متغیری از نوع آرایه

آرایه‌ها یک متغیر هستند که قرار است چندین مقدار را نگهداری کنند و مانند همه متغیرها می‌توانیم آنرا با Dim تعریف کنیم. تعریف یک آرایه به شکل زیر است و در داخل پرانتز طول آرایه را می‌نویسیم:

```
Dim myArray (2)
```

نکته ۱: این آرایه دارای سه عضو خواهد بود ، زیرا شمارش اندیس‌های یک آرایه در VBA از عدد 0 شروع می‌شود.

	0	1	2
myArray			

برای آنکه اعضای این آرایه را مقدار دهیم کنیم باید به شکل زیر عمل نمایید:

```
myArray(0) = 10
myArray(1) = 20
myArray(2) = 30
```

	0	1	2
myArray	**10**	**20**	**30**

و پس از مقدار دهی اگر بخواهیم که سه عضو اول یک آرایه را باهم جمع بزنیم، به شکل زیر عمل خواهیم کرد:

```
s= myArray(0)+ myArray(1)+ myArray(2)
```

نکته ۲: شاید در مواقعی بخواهید که شماره گذاری اعضای آرایه از ۰ شروع نشود و اولین عضو شماره ۱۰۰ و آخرین عضو شماره ۲۰۰ باشد که می‌توانیم از شکل زیر اینکار را انجام دهیم:

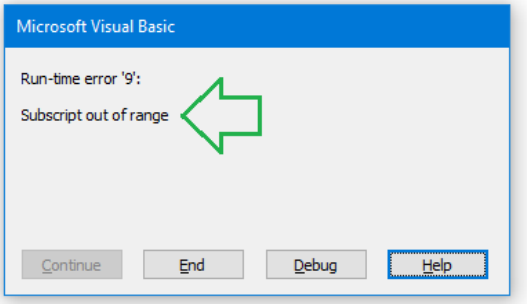
```
Dim BigArray (100 to 200)
```

حال آرایه BigArray دارای ۱۰۰ عضو است و اولین عضو آن bigArray(100) است و آخرین عضو آن bigArray (200) خواهد بود.

و اگر بخواهیم از اندیسی که وجود ندارد (مثلا اندیس ۱۷ در آرایه bigArray) استفاده کنیم، خطای زیر را خواهیم دید:

تصویر خطای مقدار دهی اشتباه یک آرایه

```
Sub Chapter7_1()

Dim bigArray(100 To 200)
bigArray(17) = 3

End Sub
```



نکته ۳: اگر بخواهیم که «همواره»شماره گذاری اعضای آرایه از عدد 1 شروع شود ، باید دستور زیر را در ابتدا ماژول بنویسیم:

```
Option Base 1
```

از این پس هر آرایه‌ای که در این ماژول تعریف شود، اندیس (شماره) اولین عضو آن عدد 1 خواهد بود.

نکته ۴: همانطور که دیدیم در هنگام تعریف آرایه، تعداد اعضای آن آرایه را در داخل پرانتز نوشتیم، به این نوع آرایه‌ها که از همان ابتدا تعداد اعضایشان مشخص است، Static Array می‌گویند.

کار با اعضای یک آرایه، تکنیک For - Next

دیدیم یک آرایه می‌تواند ده‌ها عضو داشته باشد و اگر بخواهیم با همه اعضای آرایه کار کنیم باید از یک For - Next استفاده کنیم. فرض کنید که یک آرایه با 5 عضو داریم و همچنین فرض می‌کنیم که اندیس اولین عضو آن عدد 1 است و می‌خواهیم که مقادیر هر یک از اعضای آن آرایه را چاپ کنیم، بنابراین خواهیم نوشت:

```
For i = 1 To 5
      Debug.Print myArray(i)
Next i
```

در عمل بهتر است که متغیر i را طوری تعریف کنیم که حتی اگر myArray ، دارای 10 عضو هم بود دیگر لازم نباشد که کد را تغییر دهیم. برای آنکه بدانیم که بزرگترین اندیس یک آرایه چه عددی است از تابع UBound می‌توانیم استفاده کنیم و کد قبلی را به شکل زیر تغییر می‌دهیم:

```
For i = 1 To UBound(myArray)
      Debug.Print myArray(i)
Next i
```

البته بازهم این کد خیلی کامل نیست زیرا فرض کرده‌ایم که اندیس اول آرایه از عدد1 شروع می‌شود و اگر این آرایه از عدد 0 شروع شود، این کد با خطا مواجه می‌شود زیرا myArray(5) را نداریم و در ضمن اینکه در این کد myArray(0) هم چاپ نخواهد شد، برای حل این مشکل از تابع LBound که برای یافتن اولین اندیس یک آرایه است به شکل زیر می‌توانیم استفاده کنیم:

```
For i = LBound(myArray) To UBound(myArray)
      Debug.Print myArray(i)
Next i
```

و این حلقه بر روی تک به تک اعضای ماتریس myArray حرکت خواهد کرد و هر یک از آنها را چاپ می‌کند.

کار با اعضای یک آرایه، تکنیک For - Each

تکنیک دیگری که می‌توانیم برای کار با اعضای یک آرایه (یک مجموعه) بکار ببریم استفاده از حلقه For -Each به شکل زیر است:

```
For Each i In myArray
    Debug.Print i
Next i
```

توجه ۱: دستور For - Each برای کار با مجموعه‌ها (Collections) بکار می‌رود و چون می‌توان یک آرایه را به صورت مجموعه تصور کرد بنابراین می‌توانیم برای آرایه‌ها هم آنرا بکار ببریم.

توجه ۲: در دستور For - Each دیگر لازم نیست که اولین و آخرین اندیس یک آرایه را بدانیم.

مثال ۱ آرایه‌ها - بزرگترین عدد

برنامه ای بنویسید که ۱۰ عدد صحیح را از کاربر بگیرد و سپس بزرگترین آنها را محاسبه و چاپ کند.

توجه داشته باشید که می‌توانید این برنامه را با تکنیک‌های فصل‌های قبلی بنویسید اما با کمی سختی و شلوغ کردن کد. زیرا باید ۱۰ تا متغیر با نام‌های مختلفی تعریف کنید و سپس باید کلی if بنویسید تا مشخص کنید که کدام عدد بزرگتر است.

```
Sub Chapter7_1()

Dim x(1 To 10) As Integer
Dim i As Integer
Dim big  As Integer
big = 0

For i = 1 To 10
    x(i) = InputBox("Enter Element " & i & ": ")
Next i

For i = 1 To 10
    If x(i) > big Then big = x(i)
Next i

Debug.Print "The biggest value is " & big

End Sub
```

شرح برنامه: برای آنکه ۱۰ عدد را بگیریم و آنها را در حافظه ذخیره کنیم، بهترین تکنیک استفاده از یک آرایه به طول ۱۰ است. شاید برای ما راحت تر باشد که اندیس این آرایه از عدد ۱ شروع شود به همین دلیل اینگونه می‌نویسیم:

```
Dim x(1 To 10)
```

و چون باید اعداد صحیح باشند نه اعشاری پس نوع دیتا تایپ آرایه را Integer تعریف می‌کنیم:

```
Dim x(1 To 10) As Integer
```

حالا به راحتی ۱۰ عدد را از کاربر می‌گیریم و آنها را در این آرایه ذخیره می‌کنیم:

```
For i = 1 To 10
    x(i) = InputBox("Enter Element " & i & ": ")
Next i
```

حالا ما یک آرایه داریم که ۱۰ عضو دارد و هر عضوی مقدار خود را . باید بزرگترین عضو را بیابیم. برای اینکار ما یک متغیر به نام big در نظر می‌گیریم و در ابتدا مقدار big را عدد صفر می‌دهیم. حال تک به تک اعضای آرایه با مقداری که متغیر big دارد را مقایسه می‌کنیم، اگر مقدار آن عضو بیشتر از big بود، آنگاه مقدار big باید مقدار آن عضو شود :

```
For i = 1 To 10
    If x(i) > big Then big = x(i)
Next i
```

(در توضیح بیشتر باید بگویم که ما یک متغیر به نام big داریم و هرگاه عضوی یافتیم که مقدارش بیش از مقدار متغیر big بود، آنگاه مقدار متغیر big برای با مقدار آن عضو خواهد شد و پس از پایان حلقه، قطعا بزرگترین مقدار در متغیر big قرار خواهد گرفت.)

آرایه‌های داینامیک- متغیری از نوع Variant

اگر خاطرتان باشد گفته بودیم که می‌توانیم متغیری از جنس Variant داشته باشیم و اگر متغیری Variant باشند، می‌تواند هر چیزی را در داخل خودش ذخیره کند، حتی یک آرایه !

بنابراین می‌توانیم بنویسیم:

```
Dim x as Variant
```

و حالا به یک آرایه نیاز داریم که در x ذخیره شود. این آرایه را می‌توانید با تابع Array به شکل زیر بسازید:

```
x = Array(2, 3, 4, 10)
```

و حتی می‌توانید مقادیر را از سلول‌های اکسل بخوانید (این تکنیک را در آینده شاید بگوییم) و در x ذخیره کنید و هر تعدادی سلولی که باشند مهم نیست:

```
x = Range("B5:C17")
```

و حالا متغیر x ما در خودش یک آرایه ذخیره کرده است و البته می‌توانیم اعضای آنرا چاپ کنیم.

یادآوری: ما به عنوان برنامه نویس باید تعداد اعضای x را بدانید تا یک حلقه بسازد و اعضای آنرا چاپ کند. قبلا اشاره کردم که برای دانستن اندیس (شماره) اولین عضو از تابع Lbound و برای اندیس آخرین عضو از تابع Ubound استفاده می‌کنیم:

```
Sub Chapter7_2()

Dim x As Variant
x = Array(2, 3, 4, 10)

For i = LBound(x) To UBound(x)
    Debug.Print x(i)
Next i

End Sub
```

توجه ۱: این نوع از آرایه‌ها را که تعداد اعضای آن مشخص نیست را Dynamic Array (آرایه‌های داینامیک) می‌گویند.

توجه ۲: گفتیم که اگر طول آرایه‌ای را از ابتدا مشخص کنیم، به آن Static Array می‌گویند.

مثال ۲ آرایه‌ها - تجزیه یک متن

می‌خواهیم برنامه‌ای بنویسیم که یک متن را که داده‌های آن با علامت "," از هم جدا شده است را بگیرد و سپس آنرا تجزیه کند و هر داده را نمایش دهد.

در VBA ما یک تابع به نام Split داریم که یک متن و یک جداکننده را از ما می‌گیرد و سپس با توجه به جداکننده ، متن را تجزیه می‌کند و در نهایت به ما یک آرایه خواهد داد که اعضای آن آرایه، حاصل تجزیه متن اصلی بر حسب جدا کننده است.

```
Sub Chapter7_3()

Dim x As Variant
Dim s As String

s = "Farshid,Babak,Farid,Farzad,Abtin,Baran"
x = Split(s, ",")
For i = LBound(x) To UBound(x)
    Debug.Print x(i)
Next i

End Sub
```

شرح برنامه : چون مشخص نیست که آرایه ما چند عضو دارد، بنابراین یک متغیر به نام x از نوع Variant تعریف کرده‌ایم و سپس متن را با تابع Split جدا می‌کنیم و خروجی تابع که یک آرایه است را در متغیر x قرار دادیم.

مثال ۳ آرایه‌ها - استخراج کد ملی از شرح

فرض کنید که ما شرح سندهای را در سلولهای اکسل داریم و در هر شرح سند، نام فرد، تاریخ و شماره حساب و کد ملی فرد درج شده است. حال لازم داریم که کد ملی افراد را از این شرح استخراج کنیم.

قبل از حل مساله، باید بدانیم که کد ملی بایدمتنی به طول ۱۰ باشد و البته باید یک عدد هم باشد و اطلاعات شرح سند با Space از هم جدا شده‌اند.

```
Function CodeMeli(txt As String) As String

Dim arr As Variant
arr = Split(txt, " ")

For i = LBound(arr) To UBound(arr)
    If Len(arr(i)) = 10 And IsNumeric(arr(i)) Then
        CodeMeli = arr(i)
        Exit Function
    End If
Next i

End Function
```

شرح برنامه: چون نمی‌دانیم که یک متن از چند واژه تشکیل شده است بنابراین یک متغیر به نام arr از نوع Variant تعریف می‌کنیم و سپس متن را تجزیه و حاصل تجزیه را در این متغیر می‌گذاریم.

```
Dim arr As Variant
arr = Split(txt, " ")
```

حال تک تک اعضای آرایه را بررسی می‌کنیم که آیا ۱۰ رقم هستند و همچنین آیا یک عدد است و اگر این دو شرط برقرار بود،آنگاه آن عضو از آرایه، همان کد ملی خواهد بود:

```
If Len(arr(i)) = 10 And IsNumeric(arr(i)) Then
        CodeMeli = arr(i)
```

در ضمن آنکه یک تابع برای اینکار ساختیم ، تا بتوانیم در سلول‌های اکسل از آن استفاده کنیم:

تصویر تابع استخراج کد ملی از یک متن

	A	B	C
1	شرح سند		
2	من به حساب 23432 فرید با کد ملی 2236587452 در تاریخ 950517 واریزی را انجام دادم.	=CodeMeli(A2)	
3	کاوه هم به حساب 23434 برای من که کد ملیم 8333223345 است ، در دیروز مورخ 960902 وار	8333223345	
4	کد ملی من 5566332258 است و شماره حسابم 232342342 می باشد و تاریخ امروز هم 12 آبان ا	5566332258	
5	اگر این کتاب را خرید لطفا به شماره حساب من که در سایت است مبلغش را واریز کنید. ممنونم		
6			

یادآوری: می‌توانید اینکار را با ابزار Flash Fill در Excel 2016 بدون نیاز به کد نویسی انجام دهید.

آرایه‌های داینامیک

تا اینجا دیدیم که اگر طول یک آرایه مشخص نباشد، می‌توانیم متغیری از جنس Variant تعریف کنیم و سپس می‌توانیم در این متغیر آن آرایه را ذخیره کنیم.

در اینجا یاد می‌گیریم که تکنیک دیگری هم برای تعریف آرایه‌های داینامیک هم وجود دارد و می‌توانیم آرایه داینامیک را بدون نیاز به متغیری از جنس variant هم تعریف کنیم .

برای اینکار کافی است که هنگام تعریف آرایه، طول آنرا داخل پرانتز «ننویسیم»:

```
Dim myArray()
```

حال متغیر myArray یک آرایه است و می‌تواند هر طولی داشته باشد. در ضمن بهتر است که نوع این آرایه هم مشخص کنیم:

```
Dim myArray() As Integer
```

اگر بخواهید که اولین عضو این آرایه را مقدار دهی کنید، با خطا مواجه می‌شوید، زیرا باید VBA بداند که این آرایه چند عضو دارد . تعجب کردید؟ بگذارید توضیح دهم.

در این حالت که یک Dynamic Array ایجاد کرده ایم، به معنای آن نیست که آرایه با طول بی‌نهایت می‌توانیم داشته باشیم، بلکه به معنای آن است که می‌توانید طول یک آرایه را در حین اجرای برنامه تعیین کنید و لازم نیست که از ابتدا طول آنرا بدانید.

بنابراین طول برنامه را اینگونه تعیین می‌کنیم:

```
Dim myArray() As Integer
ReDim myArray(10)
```

و در این دستور VBA متوجه می‌شود که باید بزرگترین اندیس برابر 10 داشته باشد و چون اولین عضو از اندیس 0 شروع می‌شود، کلا آرایه ما 11 عضو خواهد داشت.

و اگر بخواهیم می‌توانیم مقدار دهی اعضا را هم انجام دهیم:

```
Sub Chapter7_4()

    Dim myArray() As Integer
    ReDim myArray(10)
    myArray(0) = 100
    myArray(1) = 200

End Sub
```

اما کار با آرایه‌های داینامیک به همین جا ختم نمی‌شود، در واقع این امکان را دارید که «مجدد» تعداد اعضای یک آرایه را تعیین کنید:

```
Sub Chapter7_5()

    Dim myArray() As Integer
    ReDim myArray(10)
    myArray(0) = 100

    ReDim myArray(20)

    Debug.Print myArray(0)

End Sub
```

نکته ۱: اگر با دستور ReDim تعداد اعضای یک آرایه را «مجدد» تعریف کنید، مقادیر قبلی آن آرایه پاک خواهند شد. به همین دلیل در پروسیجر Chapter7_5 ، نتیجه دستور Debug.Print myArray(0) چاپ عدد 100 نخواهد بود و مقدار 0 چاپ خواهد شد.

نکته ۲: اگر بخواهیم در تعیین مجدد تعداد اعضای یک آرایه مقادیر قبلی پاک نشوند، باید از واژه Preserve استفاده نماییم:

```
Sub Chapter7_6()

    Dim myArray() As Integer
    ReDim myArray(10)
    myArray(0) = 100
    ReDim Preserve myArray(20)

    Debug.Print myArray(0)

End Sub
```

و در این حالت عدد 100 را خواهیم داشت.

نکته ۳: همانطور که تا اینجا دیدیم می‌توانیم یک آرایه داینامیک را با یکی از دو روش زیر تعریف کنیم:

- روش اول: استفاده از متغیری از نوع Variant
- روش دوم: تعریف آرایه بدون مشخص کردن تعداد اعضا در هنگام تعریف اولیه

نتکته: در خصوص استفاده بهینه از حافظه و سایر ویژگی‌ها، من دلایل و یا منابعی را نیافتم که بگوید کدامیک از این دو روش نسبت به هم برتری دارند.

آرایه‌های دو بعدی (چند بعدی)

تا اینجا تمام آرایه‌هایی که برای شما مثال زدیم فقط یک بعد داشتند و می‌توانید آن را مانند یک ماتریس سطری تصور کنید:

تصویری آرایه یک بعدی

	0	1	2	3
myArray				

و اگر بخواهیم که آرایه‌ای دو بعدی داشته باشیم، کافی است که آنرا به شکل زیر تعریف کنیم:

```
Dim myArray(2, 5)
```

می‌توانید آنرا به شکل ماتریسی که سه سطر و شش ستون دارد تصور کنید.

یادآوری: اندیس آرایه‌ها از عدد صفر شروع می‌شود.

تصویر یک آرایه دو بعدی

		0	1	2	3	4	5
myArray	0						
	1						
	2						

و اگر بخواهیم که اعضای این آرایه را مقدار دهی کنیم، باید اینکار را به شکل زیر انجام دهیم:

```
myArray(2, 2)= 100
```

نکته ۱: اگر بخواهیم که شماره گذاری اندیس‌ها از عدد ۱ شروع شوند، می‌توانیم تعریف آرایه را به شکل زیر انجام دهیم:

```
Dim myArray(1 To 2, 1 To 5)
```

نکته ۲: ما می‌توانیم آرایه های بیش از ۲ بعدی را در VBA تعریف کنیم. (اینکه حداکثر بعد یک آرایه چه عددی می‌تواند باشد را جایی ندیدم که ذکر کرده باشند و فقط در برخی از منابع گفته شده است که در تئوری یک آرایه می‌تواند به حجم حافظه محدود شود و یا 2GB و ... .)

نکته ۳: گفته بودم که با تابع LBound و UBound می‌توانیم بزرگترین و کوچکترین اندیس یک آرایه را بیابیم و اگر بخواهیم که در یک آرایه دو بعدی اینکار را انجام دهیم، باید مشخص کنیم که کدام بعد منظور ما است:

```
 Sub Chapter7_7()
Dim myArray(1 To 2, 1 To 5)
Debug.Print UBound(myArray, 1)
Debug.Print UBound(myArray, 2)

End Sub
```

نکته ۴: برای حرکت بر روی اعضای یک آرایه چند بعدی باید از حلقه‌های تو در تو استفاده کنیم.

آرایه‌ها و سلول‌های اکسل

ما می‌توانیم به یکباره تمامی مقادیر سلول‌های اکسل را در یک آرایه ذخیره کنیم و یا اینکه مقادیر یک آرایه را به یکباره در سلول‌ها کپی نماییم. تاکید من بر روی واژه «به یکباره» است. یعنی لازم نیست که تک به تک سلولها را بخواهیم و سپس هر عضو از آرایه را مقدار دهی کنیم :

```
Sub Chapter7_8()

Dim myArray()
myArray = Sheet4.Range("A1:B5")

End Sub
```

توجه: البته ما هنوز کار با سلول‌های اکسل را به شما درس نداده‌ایم اما با این وجود می‌توانید درک کنید که مقدار دهی آرایه فقط در یک سطر انجام شده است.

نکته ۱: در این حالت یک آرایه ۲ بعدی خواهیم داشت.

نکته ۲: در این حالت نباید DataType آرایه را تعریف کنیم . در غیر اینصورت پیغام خطای Type_mismatch نمایش داده می‌شود.

مثال ۴ آرایه‌ها- خواندن و نوشتن در سلول‌ها

برنامه‌ای بنویسید که اعداد مثبت سلولهای A1:A5 را در ۱۰۰ ضرب کند.

می‌خواهم تاکید بسیار خاصی بر روی این مثال انجام دهم و توجه شما را به روشی که در این مثال بکار می‌بریم جلب کنم. در این مثال ما همه داده‌ها را به یکبار در داخل یک آرایه ذخیره می‌کنیم و سپس محاسبات را روی تک تک اعضای آرایه انجام می‌دهیم و در نهایت به یکباره همه اعضای آن آرایه را در سلولهای اکسل کپی می‌کنیم . این روش از سایر روش‌ها بسیار سریعتر خواهد بود.

اگر بپرسید که سایر روش‌ها چیست، پاسخ خواهم داد ساده ترین روش آن است که اصلا از آرایه‌ها استفاده نکنید و با تک تک سلولها مستقما کار کنید و در نتیجه سرعت اجرای این روش بسیار کند تر از آرایه ها خواهد بود.

```
Sub Chapter7_9()

Dim myArray()
myArray = Sheet4.Range("A1:B5")

For i = LBound(myArray, 1) To UBound(myArray, 1)
    For j = LBound(myArray, 2) To UBound(myArray, 2)
        If myArray(i, j) >= 0 Then myArray(i, j) = myArray(i, j) * 100
    Next j
Next i

Sheet4.Range("A1:B5") = myArray

End Sub
```

شرح برنامه:

برای آنکه مقادیر سلول‌ها را در یک آرایه ذخیره کنیم از روش زیر می‌توان استفاده کرد:

```
myArray = Sheet4.Range("A1:B5")
```

آرایه myArray دو بعدی است یعنی از سطرها و ستون‌هایی تشکیل شده است. در این مثال دقیقا می‌دانیم که این آرایه 5 سطر و 2 ستون دارد، بنابراین برای آنکه بتوانیم تک به تک اعضای این آرایه را بخوانیم از ۲ حلقه باید استفاده کنیم:

```
 For i = 0 To 4
    For j = 0 To 1
            …
    Next j
Next i
```

معمولا ما با تابع LBound , UBound ، اندیس آرایه‌ها را محاسبه می‌کنیم، (در این مثال منطقه A1:B5 است اما در عمل ممکن است که از کاربر بخواهیم که منطقه ای از سلول ها را انتخاب کند و به همین دلیل شماره آخرین عضو را نخواهیم دانست)، بنابراین نوشتن کد به روش زیر بهتر است:

```
For i = LBound(myArray, 1) To UBound(myArray, 1)
    For j = LBound(myArray, 2) To UBound(myArray, 2)
      …
    Next j
Next i
```

و حال کافی است که روی تک به تک اعضای آرایه شرط زیر را بررسی و سپس محاسبه لازم را انجام دهیم:

```
If myArray(i, j) >= 0 Then myArray(i, j) = myArray(i, j) * 100
```

و در نهایت برای کپی کردن کل یک آرایه در یک منطقه از اکسل از روش زیر استفاده می‌شود:

```
Sheet4.Range("A1:B5") = myArray
```

آرایه‌ها به عنوان ورودی یک تابع

در فصل توابع یاد گرفتیم که توابع می‌توانند چیزهایی را به عنوان ورودی بگیرند و سپس روی آن ورودی‌ها محاسباتی را انجام دهند و در آخر هم به ما خروجی بدهند. در اینجا خواهیم دید که ورودی و یا خروجی یک تابع می‌تواند از نوع آرایه باشد. (مجددا یادآوری می‌کنم که شاید برنامه نویسان تازه کار، این تکنیک‌ها کمی گیج کننده به نظر آید اما نگران نباشید و بدانید با کمی تمرین، تمرکز و گذشت زمان آنها را به تدریج فرا خواهید گرفت.)

فرض کنیم که می‌خواهیم تابعی داشته باشیم که یک آرایه یک بعدی از جنس Integer را به عنوان ورودی بگیرد و سپس تک تک اعضای آن آرایه را برای ما چاپ کنید. این تابع به شکل زیر خواهد بود:

```
Function PrintArrayElements(arr() As Integer)

For i = LBound(arr) To UBound(arr)
    Debug.Print arr(i)
Next i

End Function
```

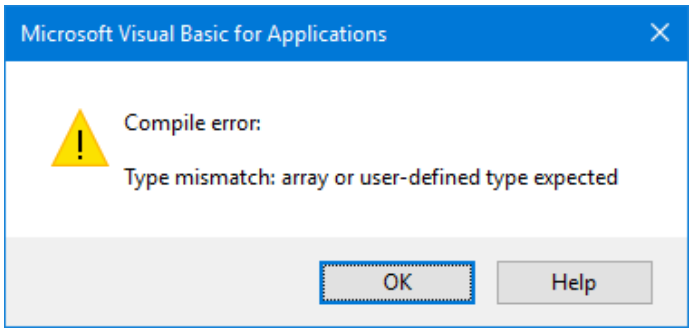
ورودی این تابع arr است و با علامت پرانتزی که جلوی آن است VBA متوجه می‌شود که این ورودی یک آرایه خواهد بود.

حال برای استفاده از این تابع کافی است که یک آرایه را به عنوان ورودی به آن بدهیم:

```
Sub Chapter7_10()

Dim myArray(2) As Integer
myArray(1) = 3: myArray(2) = 33: myArray(0) = 44

PrintArrayElements myArray

End Sub
```

نکته : از آنجایی که در تابع PrintArrayElements ما Data Type آرایه از نوع Integer گذاشته‌ایم، حتما باید ورودی آن نیز آرایه از نوع Integer باشد، در غیر اینصورت خطا زیر را خواهیم دید:

تصویر خطا به علت متفاوت بودن DataType



ورودی تابع از نوع Variant

می‌دانیم که DataType یک متغیر اگر Variant باشد، یعنی می‌توانیم در داخل آن متغیر هر چیزی را ذخیره کنیم. حتی یک آرایه را.

حال اگر ورودی یک تابع را از نوع Variant بگذاریم، می‌توانیم به آن تابع یک آرایه را به عنوان ورودی بدهیم. در واقع می‌توانیم تابع PrintArrayElements را اینگونه هم بنویسیم:

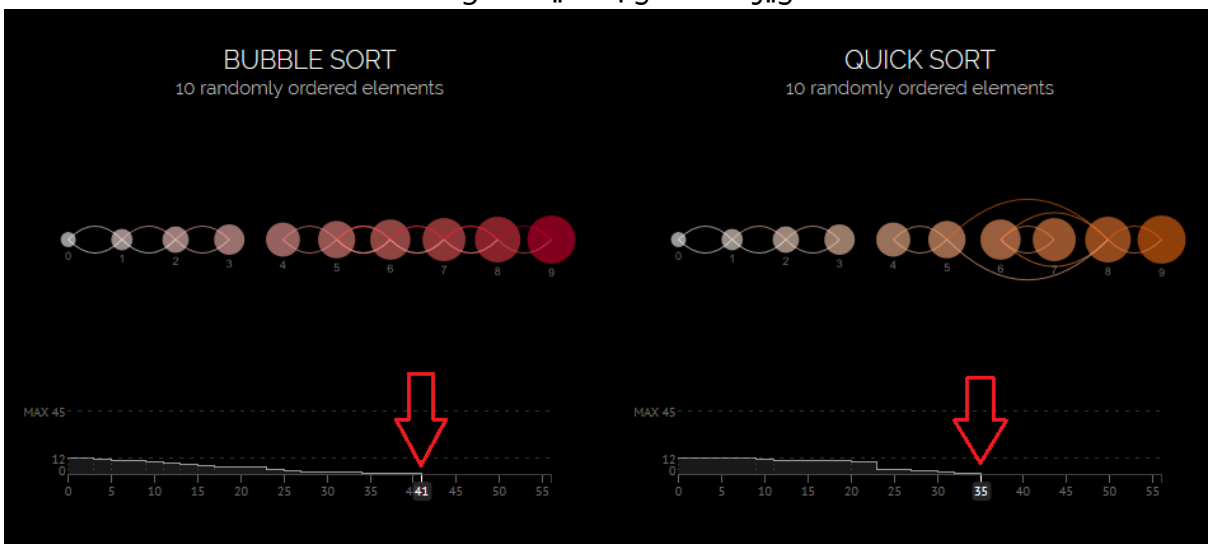
```
Function PrintArrayElements(arr As Variant)

For i = LBound(arr) To UBound(arr)
    Debug.Print arr(i)
Next i

End Function
```

مثال ۴ آرایه‌ها - مرتب سازی

موضوع مرتب سازی (Sort) یا کردن داده‌ها در دنیای کامپیوتر امری جدی است. زیرا روش‌های (الگورتیم‌های) بسیاری برای مرتب سازی وجود دارد و بهترین روش، روشی است که سریعتر بتواند اینکار را انجام دهد.

اگر مایل هستید می‌توانید به وب سایت http://sorting.at نگاهی بیندازید و به صورت انیمیشن سرعت اجرای انواع روش‌ها را ببینید.

تصویر صفحه وب سایت Sorting.at



در این تصویر در محور افقی تعداد عملیات مورد نیاز برای مرتب سازی یک مجموعه از اعداد یکسان را می‌توانید مقایسه کنید. در روش Bubble Sort برای مرتب سازی به ۴۱ عملیات و در روش Quick Sort به ۳۵ عملیات نیاز داریم و متوجه می‌شویم که روش Quick Sort سریعتر است.

البته در اینجا قصد نداریم که تمامی الگوریتم‌ها (روش‌های) مرتب سازی را بررسی و مقایسه کنیم و قصد داریم برنامه‌ای بنویسیم که از روش Bubble Sort که فهم آن برای مبتدیان ساده است، مرتب سازی یک مجموعه را انجام دهیم.

قبل از شروع باید الگوریتم Bubble Sort (مرتب سازی به روش حبابی) را برای شما شرح دهم.

روش کار به این صورت است که دو مقدار از مجموعه با هم مقایسه می‌شوند و اگر ترتیب آنها اشتباه بود، جای آنها را عوض می‌کنیم و اینکار را آنقدر انجام می‌دهیم تا دیگر نیازی نباشد که جای دو مقدار را عوض کنیم.

بگذارید برای شما یک مثال بزنم تا دقیقا متوجه شوید. فرض کنید که یک آرایه از اعداد 5,1,4,2,8 داریم.

اولین مرحله مقایسه

5	**1**	4	2	8	➔	**1**	**5**	4	2	8	دو مقدار اول مقایسه می‌شوند و چون مرتب نیستد، جای آنها عوض می‌شود
1	**5**	**4**	2	8	➔	1	**4**	**5**	2	8	تعویض جا انجام می شود چون 5>4 است.
1	4	**5**	**2**	8	➔	1	4	**2**	**5**	8	تعویض جا انجام می شود چون 5>2 است.
1	4	2	**5**	**8**	➔	1	4	2	**5**	**8**	لازم به تعویض جای مقادیر نیست چون 8>5 است.

دومین مرحله مقایسه

1	**4**	2	5	8		**1**	**4**	2	5	8	
1	**4**	**2**	5	8		1	**2**	**4**	5	8	تعویض جا انجام می شود چون 4>2 است.
1	2	**4**	**5**	8		1	2	**4**	**5**	8	
1	2	4	**5**	**8**		1	2	4	**5**	**8**	

در اینجا مرتب سازی کامل شده است اما الگوریتم ما این را نمیتواند متوجه شود و باید یکبار دیگر کل اعداد را دو به دو مقایسه کند تا متوجه شود که نیازی به تعویض جایی نیست و بنابراین مرتب سازی انجام شده است.

سومین مرحله مقایسه

1	**2**	4	5	8		**1**	**2**	4	5	8
1	**2**	**4**	5	8		1	**2**	**4**	5	8
1	2	**4**	**5**	8		1	2	**4**	**5**	8
1	2	4	**5**	**8**		1	2	4	**5**	**8**

و برنامه به شکل زیر خواهد شد:

```
Sub Chapter7_11()
'-- Bubble Sort --
Dim arr(1 To 5) As Long
Dim lower_b As Long, upper_b As Long, i As Long, tmp As Long
Dim swapped As Boolean

lower_b = LBound(arr)
upper_b = UBound(arr)

arr(1) = 5: arr(2) = 1: arr(3) = 4: arr(4) = 2: arr(5) = 8

Do
swapped = False
    For i = lower_b To (upper_b - 1)
        If arr(i) > arr(i + 1) Then
            tmp = arr(i)
            arr(i) = arr(i + 1)
            arr(i + 1) = tmp
            swapped = True
        End If
    Next i
Loop While swapped = True

' Print Sorted Array
For i = lower_b To upper_b
    Debug.Print arr(i);
Next i

End Sub
```

شرح برنامه:

نکته‌ای که می‌خواهم در اینجا اشاره کنم نحوه تعویض جای دو مقدار در آرایه است. مثلا فرض کنید که arr(1) = 5 و arr(2) = 10 است و می‌خواهیم جای این دو عضو را با هم عوض کنیم یعنی در نهایت داشته باشیم arr(1) = 10 و arr(2) = 5 .

در VBA باید این کار را انجام دهیم. مقدار یکی از این اعضا را در یک متغیر سومی مثلا به نام tmp قرار دهیم:

```
tmp = arr(1)
```

و سپس با دستور زیر به سادگی جای این دو مقدار عوض خواهند شد:

```
arr(1) = arr(2)
arr(2) = tmp
```

نکته ۱: ما می‌توانیم این برنامه را کمی بهینه کنیم. اگر به الگوریتم دقت کنید، حتما در اولین مرحله بررسی، بزرگترین مقدار به آخر آرایه منتقل خواهد شد و در مرحله بعدی لازم نیست که آخرین عضو آرایه را با قبلیش مقایسه کنیم. بنابراین می‌توانیم مراحل اجرای حلقه (مقدار i) را در هر بار مقایسه، به شکل زیر یک واحد کمتر کنیم:

```
Sub Chapter7_12()
'-- Bubble Sort Optimized --
Dim arr(1 To 5) As Long
Dim lower_b As Long, upper_b As Long, i As Long, tmp As Long
Dim swapped As Boolean

lower_b = LBound(arr)
upper_b = UBound(arr)

arr(1) = 5: arr(2) = 1: arr(3) = 4: arr(4) = 2: arr(5) = 8

Do
swapped = False
    For i = lower_b To (upper_b - 1)
        If arr(i) > arr(i + 1) Then
            tmp = arr(i)
            arr(i) = arr(i + 1)
            arr(i + 1) = tmp
            swapped = True
        End If
    Next i
i = i - 1
Loop While swapped = True

' Print Sorted Array
For i = lower_b To upper_b
    Debug.Print arr(i);
Next i

End Sub
```

از شما برای خرید و حمایت معنوی و از آن مهمتر خواندن این کتاب تشکر می‌کنم. امیدوارم که بخش اول را مفید یافته باشید و اگر نقد و یا نظراتی دارید ، لطفا مرا آگاه نمایید.

منتظر بخش دوم کتاب که در آن می خواهیم کار با اکسل را شروع نماییم باشد، از طریق ایمیل حتما شما را از آماده شدن آن آگاه خواهیم کرد.

پیروز باشید

فرشید میدانی / موسسه فرساران تفکر

www.farsaran.com

f.meidani@farsaran.com