# توابع Visual Basic.NET

به وسیله: بهروز راد

**كتاب الكترونيكي آموزشي - كاربردي فارسي توابع VB.NET**

## درباره کتاب...

| | |
|---|---|
| نام کتاب | آموزش کاربردي توابع Visual Basic.NET |
| نویسنده | بهروز راد |
| تحصیلات | دانشجوي کامپیوتر |
| تاریخ تالیف | زمستان ۱۳۸۲ |
| تعداد صفحات | ٦٥ صفحه |

## مقدمه:

سپس خداي را كه در كنار نعمات بيكران خود، توفيق ارائه اين كتاب آموزشي را به دوستداران زبان برنامه نويسي Visual Basic.NET عنايت فرمود.
در برابر انبوه كتاب هاي الكترونيكي آموزشي انگليسي زبان در مورد زبان هاي برنامه نويسي، كمبود كتاب هاي فارسي در كشور ما به شدت احساس مي شود.
از آن مهم تر، كتابي كه بر روي يك قسمت از يك زبان برنامه نويسي تمركز داشته و به نحوي شايسته و به گونه اي ساده به توضيح مبحث بپردازد، به ندرت ديده شده است.
از اين رو بر آن شدم تا با ارائه اين كتاب الكترونيكي آموزشي - كاربردي، مهمترين توابع موجود در زبان VB.NET را كه اكثرا با زبان VB.6.0 مشترك هستند، به صورت دسته بندي شده و به طور كامل و با جزئيات كامل تر، با زباني ساده و ارائه مثال هاي متعدد جهت يادگيري و فهم بهتر، به دوستداران اين زبان قدرتمند ارائه كنم.
از آنجا كه هيچ نوشته اي به طور كامل خالي از نقص و ايراد نيست، لذا از تمامي خوانندگان گرامي خواهشمندم اشتباهات كتاب را به طور مستقيم با بنده از طريق ايميل زير در ميان بگذارند.
اميد است كه با اين كار كوچك، گامي در جهت افزايش دانايي و سطح علمي دوستداران اين زبان برداشته شود.

به امید موفقیت روز افزون.

بهروز راد
زمستان ۱۳۸۲

behrouz.rad@gmail.com

## نكات مهم:

١) اكثـر توابـع VB.NET، شـبيه بـه توابـع موجـود در VB.6.0 هـستند و تنهـا در برخـي مـوارد، تغييراتي در نحوه ي فراخواني و احيانا سينتكس تابع وجود دارد.
بـه عنـوان مثـال تـابع ()Len در VB.6.0 طـول يـك رشـته را برگـشت مـي دهـد امـا در VB.NET شما قادر به انجام اين كار را با استفاده از متد Length متغير رشته اي هستيد.
فرضا اگر متغير StrVar به عنوان يك متغير رشته اي تعريف شده باشد، شما مي توانيـد طـول آن را با استفاده از متد Length اين متغير به دست آوريد:

```
Dim strVar As String =“a short string ”
Console.WriteLine(“The string contains“ & strVar.Length & “characters”)
```

يا شما مي توانيد طول اين متغير رشته اي را با قرار دادن آن در آرگومان تابع ()Len بـه دسـت آوريد:

```
Dim strVar As String =“a short string”
Console.WriteLine(“The string contains “ & Len(strVar) & “characters”)
```

٢) در VB.NET توابعي هستند كه كار تقريبا يكساني انجام مي دهند و اين نكته بـسيار مهـم است كه شما بنا به نياز خود تشخيص دهيد كه بايد از كداميك از اين توابع استفاده كنيد.

٣) اگـر شـما يكـي از آرگومـان هـاي اختيـاري را حـذف كنيـد، بايـد بـه جـاي آن از كـاراكتر كامـا استفاده كنيد.

٤) آرگومان هاي اختياري داخل كروشه قرار دارند.
به عنوان مثال شكل كلي تـابع ()Mid كـه تعـدادي از كاراكترهـاي يـك رشـته را از آن اسـتخراج مي كند به صورت زير است:

```
newString =Mid(string [,start ][,length ]
```

محـل شـروع كاراكترهـايي كـه بايـد اسـتخراج شـوند بـه وسـيله آرگومـان Start و تعـداد كاراكترهايي كه بايد استخراج شوند را آرگومان length تعيين مي كند.
همانگونه كه ملاحظه مي شود آرگومان start يـك آرگومـان اختيـاري بـوده و چنانچـه اسـتفاده نشود به طور پيش فرض محل شروع استخراج كاراكترها از ابتدا يعنـي اولـين كـاراكتر رشـته در نظر گرفته مي شود.
اگر آرگومان length استفاده نشود، از مكان مشخص شده تا به انتهاي رشـته اسـتخراج مـي شوند.
تنها آرگومان اجباري در اين تابع، اولين آرگومان يعني String مي باشد كه رشـته اي را تعيـين مي كند كه كاراكترها بايد از آن استخراج شوند.

در زير، تمامي توابع توضيح داده شده در اين كتاب به صورت دسته بندي شده ليست شده است:

**توجه:** توابعي كه در انتهاي آنها پرانتز وجود ندارد، توابع بدون آرگومان هستند.

**توابع ورودي/خروجي (Input/output):**
InputBox(),MsgBox()

**توابعي كه با فايل ها و پوشه ها سر و كار دارند (File and Folder Manipulation):**
ChDir(),ChDrive(),CurDir(),Dir(),FileCopy(),FileDateTime(),FileLen,GetAttr(),Kill,MkDir(),Rename(),RmDir()
SetAttr()

**توابعي كه براي شناسايي نوع داده ها به كار مي روند (Data Type Identification):**
IsArray(),IsDate(),IsDBNull(),IsNothing(),IsNumeric(),IsReference,TypeName(),VarType()

**توابع تبديل نوع متغير (Variable Type Conversion):**
CBool(),CByte(),CChar(),CDate(),CDbl(),CDec(),CInt(),CLng(),CObj(),CShort(),CSng(),CStr(),CType()

**توابعي كه با رشته ها سر و كار دارند (String Manipulation):**
Asc(),AscW(),Chr(),ChrW(),Filter(),InStr(),InStrRev(),Join(),LCase(),Left(),Len(),LTrim(),Mid(),Mid,Replace(),
Right(),RTrim(),Space(),Split(),StrComp(),StrConv(),StrDup(),StrReverse(),Trim(),UCase()

**توابعي كه براي قالب دهي به داده ها به كار مي روند (Data Formatting):**
Format(),FormatCurrency(),FormatDateTime(),FormatNumber(),FormatPercent(),LSet(),RSet(),Str(),Val()

**توابع رياضي (Math):**
Abs(),Atan(),Cos(),Exp(),Fix(),Hex(),Int(),Log(),Oct(),Pow(),Round(),Sin(),Sqrt(),Tan()

**توابعي كه با تاريخ و زمان سر و كار دارند (Date and Time):**
DateAdd(),DateDiff(),DatePart(),DateSerial(),DateValue(),Day(),
Hour(),Minute(),Month(),MonthName(),Now(),Second(),
TimeSerial(),TimeValue(),Weekday(),WeekdayName(),Year()

**توابعي كه جهت انجام عمليات ورودي/خروجي (I/O) بر روي فايل ها مورد استفاده قرار مي گيرند (File I/O):**
EOF(),FileAttr(),FileClose(),FileOpen(),FileGet(),FilePut(),FreeFile(),
Input(),LineInput(),Loc(),Lock(),LOF(),Print(),PrintLine(),Reset(),
Seek(),Unlock(),Width(),Write(),WriteLine()

**توابعي كه براي درست كردن اعداد تصادفي به كار مي روند (Random Numbers):**
Rnd(),Randomize

**توابع گرافيكي (Graphics):**
QBColor(),RGB()

**توابعي كه با رجيستري سر و كار دارند (Registry):**
DeleteSetting(),GetAllSettings(),GetSetting(),SaveSetting()

**توابع كاربردي (Application Collaboration):**
AppActivate(),Shell()

**توابع متفرقه (Miscellaneous):**
Beep,CallByName(),Choose(),Environ(),IIf(),Option,Switch()

در ادامه، تمامي توابع ذكر شده به ترتيب دسته بندي و تك تك، همراه با مثال توضيح داده خواهند شد.

توجه داشته باشید که توابع موجود در هر دسته به ترتیب حروف الفبا توضیح داده نشده بلکه از آسانترین شروع شده است. بنابراین فرصت این را خواهم داشت تا در مثال هایی که در توابع دیگر خواهم زد از توابع دیگر نیز با همدیگر استفاده کنم.

## توابع ورودي/خروجي (Input/output):

ويژوال بيسيك دو تابع اساسي را جهت نمايش يا درخواست اطلاعات به يا از كاربر فراهم كرده است.
اين دو تابع عبارتند از: **MsgBox() , InputBox()**
برنامه هاي كاربردي بايد به نحوي مطلوب با استفاده از فرم هاي جذاب و كاربرپسند با كاربر خود ارتباط برقرار كنند.
يكي از اين راه هاي ارتباط، استفاده از اين دو تابع كاملا كاربردي مي باشد.

### InputBox(prompt [,title ][,default ][,xpos ][,ypos ])

تابع InputBox جهت نمايش يك جعبه گفتگو (DialogBox) همراه با يك پيغام و يك جعبه متن به كار مي رود.
اين كادر محاوره تا زمان ورود اطلاعات از طرف كاربر و كليك دكمه OK يا Cancel بر روي صفحه باقي مي ماند.
آرگومان هاي اين تابع به شرح زير مي باشند:
**prompt:** پيغامي كه در هنگام ظاهر شدن كادر محاوره در آن ظاهر مي شود.
نكته: اگر نياز داريد كه متن Prompt شما در چندين خط چاپ شود آن را با استفاده از كاراكترهاي خطي VbCr يا VbLf به كار ببريد.
**Title:** عنواني كه در نوار عنوان كادر محاوره ظاهر مي شود.
توجه: اگر شما از ذكر اين آرگومان اختياري خودداري كنيد، نام برنامه شما در اين قسمت نمايش داده مي شود.
**Default:** مقدار ورودي پيش فرض كادر محاوره كه در جعبه متن قرار خواهد گرفت.
توصيه: اگر قصد تسريع در تصميم گيري كاربر در وارد كردن مقدار ورودي را داريد، مقداري را حتما در اين قسمت ذكر كنيد.
**Xpos,ypos:** مختصات گوشه ي سمت چپ و بالاي اين كادر محاوره از صفحه نمايش را با اين مقادير تعيين كنيد.
نكته: اين واحدها بر حسب Twips مي باشند.
توجه داشته باشيد كه هر Twips برابر با يك – چهارهزارو چهارصدو چهل هزارم اينچ و يك – پانصدو شصت و هفت صدم سانتي متر و يك بيستم نقطه (Point) است.
ساده ترين شكل كاربرد تابع InputBox() به شكل زير است:

```
SSN =InputBox(“Please enter your name and family ”)
```

متني را كه كاربر در اين كادر محاوره وارد مي كند در متغير SSN قرار خواهد گرفت.
مقدار برگشتي اين تابع هميشه مقداري از نوع String يا رشته اي است، حتي اگر كاربر مقداري عددي را وارد كند.
نكته: هميشه مقداري را كه توسط اين تابع برگشت داده مي شود چك كنيد. به عنوان مثال ممكن است كاربر مقداري را در كادر محاوره وارد نكند.
در صورتي كه فكر مي كنيد كاربر مقداري عددي را در كادر محاوره وارد مي كند با استفاده از تابع IsNumeric() مقدار ورودي را چك كنيد تا حتما از اعداد تشكيل شده باشد.
در صورتي كه فكر مي كنيد كاربر مقداري از نوع تاريخ را وارد مي كند با استفاده از تابع IsDate() مقدار ورودي را چك كنيد و به همين ترتيب، قالب هاي ديگر.

مثال زير چك مي كند كه مقدار ورودي كاربر حتما تاريخ باشد:

```
BDay =InputBox(“Please enter your birth date ”)
If IsDate(BDay) Then
  MsgBox(“Yes. This is a valid birth date”)
Else
  MsgBox(“Please try again with a valid birth date”)
End If
```

## MsgBox(prompt [,buttons ][,title ])

تابع ()MsgBox، يك كادر محاوره را همراه با يك پيغام نمايش داده و تا زمان بسته شدن كه با كليك كاربر بر روي هر يك از دكمه هاي آن انجام مي پذيرد، بر روي صفحه باقي مي ماند.
پيغامي كه در اين كادر محاوره نمايش داده خواهد شد به عنوان اولين آرگومان و با نام **Prompt** تعيين مي شود.
ساده ترين شكل استفاده از اين تابع به شكل زير است:

```
MsgBox("Your Computer is running out of memory!")
```

اين دستور يك پيغام را در يك كادر محاوره همراه با يك دكمه OK نمايش خواهد داد.
تابع ()MsgBox همچنين قادر به نمايش دكمه ها و آيكون هايي نيز در كادر محاوره خود مي باشد.
اين تابع مقداري عددي را برگشت مي دهد كه معرف مقدار دكمه اي است كه كاربر بر روي آن كليك كرده.
مقادير زير كليه مقاديري هستند كه آرگومان **buttons** قادر به دريافت آنهاست:

| مقادير دكمه ها در پارامتر buttons از تابع ()MsgBox | |
|---|---|
| **OKOnly** يا **0** | تنها دكمه OK ظاهر خواهد شد.(مقدار پيش فرض) |
| **OKCancel** يا **1** | دكمه هاي OK و Cancel ظاهر خواهد شد. |
| **AbortRetryIgnore** يا **2** | دكمه هاي Abort و Retry و Ignore نمايش داده خواهند شد. |
| **YesNoCancel** يا **3** | دكمه هاي Yes و No و Cancel نمايش داده خواهند شد. |
| **YesNo** يا **4** | دكمه هاي Yes و No را نمايش مي دهد. |
| **RetryCancel** | دكمه هاي Retry و Cancel را نمايش مي دهد. |
| **مقادير آيكون ها در پارامتر buttons از تابع ()MsgBox** | |
| **Critical** يا **16** | آيكون پيغام حساس را نمايش مي دهد. |
| **Question**يا **32** | آيكون علامت سوال را نمايش مي دهد. |
| **Exclamation** يا **48** | آيكون پيغام هشدار را نمايش مي دهد. |
| **Information**يا **64** | آيكون پيغام اطلاعاتي را نمايش مي دهد. |
| **مقادير دكمه هاي پيش فرض در پارامتر buttons از تابع ()MsgBox** | |
| **DefaultButton1** يا **0** | اولين دكمه را به عنوان دكمه پيش فرض انتخاب مي كند. |
| **DefaultButton2** يا **256** | دومين دكمه را به عنوان دكمه پيش فرض انتخاب مي كند. |
| **DefaultButton3** يا **512** | سومين دكمه را به عنوان دكمه پيش فرض انتخاب مي كند. |
| **مقادير شرطي در پارامتر buttons از تابع ()MsgBox** | |
| **ApplicationModal** يا **0** | تا زماني كه كاربر به كادر محاوره جوابي ندهد نمي تواند بر روي هيچكدام از فرم هاي برنامه جاري سويچ كند. |
| **SystemModal** يا **4096** | تا زماني كه كاربر به كادر محاوره جوابي ندهد نمي تواند به هيچ كدام از برنامه هاي در حال اجرا سويچ كند. |

مقادير دكمه ها به شما اين امكان را مي دهد كه تعيين كنيد كداميك از دكمه ها بر روي كادر محاوره ظاهر شود.
توجه داشته باشيد كه شما نمي توانيد يك مقدار منحصر به فرد براي اين آرگومان تعيين كنيد.
زماني كه رويدادي استثنايي يا غير عادي در ويندوز رخ دهد، به وسيله آيكون ها نوع اين رويداد تعيين مي شود. اين رويداد ممكن است حساس، پرسشي، هشدار و يا اطلاعاتي باشد.
مقادير آيكون ها نوع آيكون ظاهر شونده را براي ما تعيين مي كنند.
مقادير پيش فرض دكمه ها تعيين مي كنند كه كدام دكمه در هنگام ظاهر شدن كادر محاوره به عنوان دكمه پيش فرض انتخاب شده باشد. در صورتي كه تعيين نشود اولين دكمه از سمت چپ به عنوان دكمه پيش فرض انتخاب مي شود.
مقادير شرطي نيز نوع رفتار كادر محاوره را به عكس العمل كاربر مشخص مي كنند.
تابع ()MsgBox، مقداري از نوع Integer را برگشت مي دهد كه معين مي كند كدام دكمه توسط كاربر فشرده شده است. اين مقادير در جدول زير ليست شده است:

| مقاديري كه تابع ()MsgBox برگشت مي دهد | |
|---|---|
| **مقدار** | **توضيحات** |
| **OK** يا **1** | دكمه OK فشرده شده است. |
| **Cancel** يا **2** | دكمه Cancel فشرده شده است. |
| **Abort** يا **3** | دكمه Abort فشرده شده است. |
| **Retry** يا **4** | دكمه Retry فشرده شده است. |
| **Ignore** يا **5** | دكمه Ignore فشرده شده است. |
| **Yes** يا **6** | دكمه Yes فشرده شده است. |
| **No** يا **7** | دكمه No فشرده شده است. |

به عنوان مثال براي نمايش كادر محاوره اي با دكمه هاي OK و Cancel و آيكون پيغام هشدار به صورت زير عمل مي كنيم:

```
Res =MsgBox(“This operation may take several minutes”,_
MsgBoxStyle.Exclamation +MsgBoxStyle.OKCancel)
```

مقداري كه توسط اين كادر محاوره برگشت داده خواهد شد، يكي از مقادير ذكر شده در جدول بالاست كه بسته به انتخاب هر يك از دكمه هاي OK يا Cancel، يكي از مقادير 1 يا 2 خواهد بود.
در مثال بعدي قصد ظاهر كردن كادر محاوره اي با دكمه هاي Yes و No همراه با آيكون پيغام حساس را داريم:

```
Res =MsgBox(“Incomplete data.Would you like to retry?”,_
MsgBoxStyle.YesNo +MsgBoxStyle.Critical)
If Res =MsgBoxResult.Yes Then ‘user clicked Yes
  {prompt again}
Else ‘user clicked No
  {exit procedure}
Endif
```

## توابعي كه با فايل ها و پوشه ها سر و كار دارند (File and Folder Manipulation):

توابع زير برخي از كارهاي مديريتي را در رابطه با فايل ها و پوشه ها همانند انتقال و تغيير نام فايل ها، ايجاد پوشه جديد، حذف فايل يا پوشه و ... انجام مي دهند.
فرمان هاي زير با محتويات فايل ها كاري نداشته و اكثرا همانند فرمان هاي مديريتي فايل ها و پوشه هاي DOS هستند.

### GetAttr(pathname):

اين تابع مقداري را از نوع Integer برگشت مي دهد كه نمايانگر صفت فايل، دايركتوري يا پوشه اي است كه نام آن در آرگومان pathname آمده است.
مقاديري را كه اين تابع مي تواند برگشت دهد در زير ليست شده است:

| مقاديري كه تابع GetAttr() برگشت مي دهد | |
|---|---|
| **مقدار** | **توضيحات** |
| **Normal** يا **0** | نرمال (هيچ صفتي ندارد) |
| **ReadOnly** يا **1** | فقط خواندني |
| **Hidden** يا **2** | مخفي |
| **System** يا **4** | سيستمي |
| **Volume** يا **8** | برچسب درايو |
| **Directory** يا **16** | دايركتوري يا پوشه |
| **Archive** يا **32** | آرشيو (فايل در آخرين دسترسي تغيير پيدا كرده است) |

نكات مهم در استفاده از اين تابع: براي تشخيص اينكه عبارت ما داراي چه صفت يا صفاتي است از عملگر AND استفاده كنيد.( ممكن است عبارت داراي چندين صفت باشد)
اگر عبارت ما هيچگونه صفتي نداشته باشد، مقدار برگشتي، صفر خواهد بود.
به عنوان مثال، جهت تشخيص اينكه عبارت ما داراي صفت فقط خواندني است يا خير به صورت زير عمل مي كنيم:

```
Result =GetAttr(FName) And FileAttribute.ReadOnly
```

اگر عبارت Fname داراي صفت فقط خواندني باشد، مقدار برگشتي كه در متغير Result ذخيره مي شود، 1 خواهد بود در غير اينصورت با فرض اينكه عبارت ما هيچ صفت ديگري ندارد، مقدار Result، صفر خواهد بود.
در مثال زير تعيين مي كنيم كه آيا عبارت ما داراي صفت Archive است يا خير:

```
Result =GetAttr(FName)And FileAttribute.Archive
```

اگر عبارت، داراي مقدار Archive باشد، مقدار Result، 32 خواهد بود.
اگر عبارت ما هر دو مقدار فقط خواندني و آرشيو را دارا باشد، مقدار ReadOnly و Archive با يكديگر جمع مي شوند، در نتيجه Result، 33خواهد شد.
از عملگر OR نيز مي توانيد استفاده كنيد.

### SetAttr(pathname,attributes)

اين تابع، صفتي را بر روي يك فايل قرار مي دهد.
آرگومان pathname، مسير فايلي است كه قرار است صفت بر روي آن ايجاد شود و آرگومان attributes يك مقدار عددي است كه براي تعيين نوع صفت يا صفت هايي است كه قرار است بر روي فايل ايجاد شود.
مقاديري را كه آرگومان attributes مي تواند بپذيرد در جدول تابع GetAttr() ذكر شد.

نكات مهم در استفاده از اين تابع: از آنجا كه اين تابع مقدار برگشتي ندارد، آن را در متغير قرار ندهيد.
در اين تابع مي توان يك يا چند صفت را همزمان ايجاد يا حذف كرد.
مثال زير را در نظر بگيريد:

```
SetAttr "C:\Windows\MyFile.txt", vbHidden And -vbReadOnly And vbArchive
```

اين دستور صفت هاي مخفي و آرشيو را بر روي فايل MyFile.txt ايجاد و صفت فقط خواندني آن را بر مي دارد.
توجه داشته باشيد در صورتي كه يك صفت را براي يك فايل تعيين مي كنيد صفت يا صفات قبلي فايل حذف مي شوند.
به مثال زير دقت كنيد:

```
If GetAttr(file_name) And FileAttribute.Hidden Then
  SetAttr(file_name, GetAttr(file_name)– FileAttribute..Hidden)
End If
```

در اين مثال، ابتدا با استفاده از تابع GetAttr() چك مي شود كه فايل مورد نظر ما داراي صفت مخفي است يا خير، اگر داراي صفت مخفي باشد صفت مخفي آن با استفاده از تابع SetAttr() برداشته مي شود.
همچنين مي توان با استفاده از تابع MsgBox() در مورد حذف صفت فايل تاييد گرفت:

```
If GetAttr(file_name) And FileAttribute.ReadOnly Then
  reply=MsgBox(“This is a read-only file.Delete it anyway?”, _
  MsgBoxStyle.YesNo)
  If reply=MsgBoxResult.Yes Then
    SetAttr(file_name, GetAttr(file_name) - FileAttribute.ReadOnly)
    Kill(file_name)
  End If
Else
  Kill(file_name)
End If
```

در اين مثال، فايل در مورد داشتن صفت فقط خواندني چك مي شود و در صورت داشتن آن با پرسشي از كاربر در مورد حذف آن تاييد گرفته مي شود.

## Kill(pathname)

اين تابع، فايل يا فايل هايي را كه مسير و نام آنها در آرگومان pathname تعيين مي شود پاك مي كند.
توجه داشته باشيد كه فايل هاي پاك شده به سطل زباله ويندوز منتقل نمي شوند و براي هميشه پاك خواهند شد.
نكته مهم: در صورتي كه با استفاده از تابع FileCopy() فايلي را به سطل زباله ويندوز انتقال دهيد، فايل منتقل شده در پنجره سطل زباله ظاهر خواهد شد اما شما قادر به بازيابي آن نخواهيد بود.
همچنين شما در اين تابع قادر به استفاده از كاراكترهاي جانشيني هستيد. (* معرف همه) و (؟ معرف يك كاراكتر)
اگر فايلي كه نام آن در آرگومان pathname ذكر مي شود وجود نداشته باشد، يك خطاي RunTime (زمان اجرا ) اتفاق مي افتد.
سعي كنيد از اين تابع به شكل زير استفاده كنيد:

```
Try
Kill(“C:\Log.txt”)
Catch exc As Exception
End Try
```

در اين مثال با استفاده از تعريف ساختار Error در برنامه به وسيله Try-Catch، از رخ دادن خطاي زمان اجرا كه ممكن است به دليل وجود نداشتن فايل Log.txt در مسير ذكر شده

ايجاد شود يا هر دليل ديگري كه ممكن است از حذف فايل جلوگيري كند، جلوگيري شده و در صورت رخ دادن خطا، كنترل برنامه به خط بعد منتقل مي شود.

### FileDateTime(pathname)

اين تابع تاريخ و زمان ايجاد يك فايل يا تاريخ و زمان آخرين زماني كه فايل تغير پيدا كرده و ويرايش شده است را برمي گرداند.
مثال زير را در نظر بگيريد:

```
Console.WriteLine(FileDateTime(“myDocument.txt”))
```

تاريخ و زمان برگشتي فرضا به شكل زير خواهد بود:

“21/11/01 14:13:02 PM”

### FileLen(pathname)

مقدار برگشتي اين تابع، عددي از نوع Long Integer خواهد بود كه ميزان حجم فايلي را مشخص مي كند كه نام آن در آرگومان pathname آمده است.
مثال زير را در نظر بگيريد:

```
MsgBox(“The file contains” & FileLen(“.\docs \myDocument.txt”) & “bytes”)
```

ميزان حجم فايل تعيين شده در كادر محاوره ظاهر خواهد شد.
نكته مهم: تابع FileLen() با تابع LOF() (كه حجم فايلي را كه هم اكنون باز است برمي گرداند) تفاوت دارد. جهت اطلاعات بيشتر، توضيحات تابع LOF() را كه در گروه توابع ورودي/خروجي فايل ها (File I/O) دسته بندي شده است ببينيد.

### MkDir(path)

اين تابع يك پوشه (دايركتوري) جديد را ايجاد مي كند.
آرگومان path، نام پوشه و مسيري است كه قصد ايجاد آن را داريم.
به مثال زير توجه كنيد:

```
MkDir(“C:\Users \New User”)
```

اين دستور، پوشه اي جديد به نام New Users در پوشه Users ايجاد مي كند. (البته اگر پوشه Users وجود داشته باشد.)
همچنين شما مي توانيد به يك مسير يا پوشه دلخواه سويچ كرده و پوشه هاي خود را در آنجا ايجاد كنيد.
مثال زير را در نظر بگيريد:

```
ChDrive(“C:\”)
ChDir(“C:\”)
MkDir(“Users”)
ChDir(“Users”)
MkDir(“New User”)
```

نكته مهم: حتما از ساختار Try-Catch در ساخت پوشه ها استفاده كنيد چون در صورتي كه پوشه اي كه قصد ايجاد آن را داريد در مسير ذكر شده وجود داشته باشد، خطايي رخ خواهد داد. (البته اگر پوشه وجود داشت، مي توانيد آن را حذف و اقدام به ايجاد پوشه ي جديد كنيد)

### RmDir(path)

اين تابع براي حذف پوشه يا دايركتوري كه مسير و نام آن در آرگومان Path آمده است به كار مي رود.
نكات مهم در استفاده از اين تابع: آرگومان Path نمي تواند داراي كاراكترهاي جانشيني (* و ؟) باشد. به عبارت ديگر شما نمي توانيد چندين پوشه را با يك بار فراخواني اين تابع حذف كنيد.

براي حذف پوشه، بايد دو شرط زير برقرار باشد:
۱) پوشه خالي باشد. ۲) پوشه جاري نباشد.
در صورت رعايت نكردن هر يك از موارد فوق، يك خطاي زمان اجرا اتفاق مي افتد.
براي حذف پوشه اي كه داراي تعدادي فايل است، ابتدا با تابع Kill() فايل هاي درون آن را حذف و سپس تابع RmDir را فراخواني كنيد. علاوه بر آن اگر پوشه ي شما داراي تعدادي زيرپوشه نيز باشد، بايد آنها را هم قبل از حذف پوشه ي اصلي حذف كنيد.
فرمان زير را در نظر بگيريد:

```
RmDir("C:\Users")
```

در صورتي كه پوشه ي Users داراي زيرپوشه اي به نام New User باشد، خطايي اتفاق مي افتد.
پس فرمان فوق را به صورت زير اصلاح مي كنيم:

```
RmDir("C:\Users \New User")
RmDir("C:\Users")
```

## ChDir(path)

اين تابع، پوشه (دايركتوري) جاري را تغيير مي دهد.
به عنوان مثال براي سويچ به پوشه C:\Windows به شكل زير عمل مي كنيم:

```
ChDir("C:\Windows")
```

نكات مهم: اگر در آرگومان path اين تابع از آوردن **نام درايو** خودداري شود، تابع ChDir() تلاش مي كند تا در **درايو جاري** به پوشه اي كه نام آن در در آرگومان path مشخص شده سويچ كند و اگر پوشه وجود نداشته باشد، يك خطا رخ مي دهد و مسير جاري تغيير پيدا نمي كند.
توجه: تابع ChDir()، پوشه جاري را تغيير مي دهد نه درايو جاري را.
به عنوان مثال اگر درايو جاري (C:) باشد، دستور زير پوشه جاري را به پوشه ديگري در درايو (D:) تغيير مي دهد اما درايو (C:) به عنوان درايو جاري باقي مي ماند:

```
ChDir "D:\TMP"
```

براي تغيير درايو جاري از تابع ChDrive() كه به عنوان تابع بعدي توضيح داده شده است استفاده كنيد.
شما همچنين مي توانيد **نام هاي نسبي** را به عنوان نام پوشه به كار ببريد.
به عنوان مثال دستور ChDir("..") پوشه جاري را يك مرحله به بالا در پوشه اصلي انتقال مي دهد در صورتي كه دستور ChDir("..\MyFiles")، پوشه MyFiles را در پوشه اصلي به عنوان پوشه جاري در نظر مي گيرد.
توجه: هم پوشه ي جاري و هم پوشه ي MyFiles، دو زيرپوشه در يك پوشه اصلي هستند.

## ChDrive(drive)

اين تابع درايو جاري را تغيير مي دهد. همانند: ChDrive("D:")
اين دستور، درايو جاري را به درايو D تغيير مي دهد.
نكات مهم: درايوي كه نام آن در آرگومان drive ذكر مي شود بايد وجود داشته باشد.
اگر در آرگومان drive نام چندين درايو ذكر شود، نام اولين درايو جهت تغيير در نظر گرفته مي شود.

## CurDir([drive ])

هنگامي كه اين تابع، بدون آرگومان اختياري drive فراخواني شود، نام پوشه جاري را در درايو جاري برمي گرداند.
براي اينكه بفهميد كه كدام پوشه در درايو ديگري پوشه جاري است، نام درايو را در آرگومان drive ذكر كنيد.

به عنوان مثال اگر شما در يك پوشه در درايو (C:) باشيد، دستور CDir =CurDir() مسير پوشه جاري را در درايو جاري برمي گرداند.
براي اينكه بفهميم كه پوشه جاري در يك درايو ديگر مثلا (D:) چيست، تابع CurDir() را به صورت زير فراخواني مي كنيم:

```
DDir =CurDir("D")
```

## Dir([pathname [,attributes ]])

اين تابع، دو آرگومان اختياري را دريافت كرده و يك مقدار از نوع رشته اي را كه نام فايل يا پوشه اي است كه با آرگومان pathname (مسير فايل يا پوشه) يا attributes (صفت يا صفات فايل) مطابقت دارد، برگشت مي دهد.
نكات مهم در استفاده از اين تابع:
در اين تابع مجاز به استفاده از علامت هاي جايگزين هستيد.(* و ؟)
مثال زير را در نظر بگيريد:

```
Dim sFile As String
sFile = Dir("C:\Windows\")
Do Until sFile = vbNullString
   Console.WriteLn sFile
   sFile = Dir()
Loop
```

در اين مثال، كليه فايل هايي كه در پوشه Windows وجود دارند(در پوشه اصلي Windows نه در زير پوشه هاي آن) چاپ خواهند شد.
اگر هيچ فايلي با آرگومان هاي ذكر شده در تابع Dir() صدق نكند، مقدار برگشتي، يك مقدار خالي خواهد بود.
دومين آرگومان، يك مقدار عددي يا ثابت است كه در توضيحات تابع SetAttr() آمدند و مي تواند يك مقدار يا بيشتر باشد.
در صورتي كه از ذكر اين آرگومان اختياري خودداري شود، تنها فايل هاي Normal (فايل هاي بدون صفت) برگشت داده خواهند شد.
يكي از كاربردهاي اين تابع براي تعيين وجود يك فايل يا پوشه در مسير مشخص شده است.
به فرمان زير توجه كنيد:

```
OCXFile =Dir("C:\WINDOWS \SYSTEM \MSCOMCTL.OCX")
```

مقدار برگشتي اين فرمان در صورت وجود فايل MSCOMCTL.OCX در مسير ذكر شده، همان نام فايل است در غير اينصورت يك مقدار تهي برگشت داده مي شود.
در اين مثال مي خواهيم بدانيم كه چند فايل با پسوند DLL در مسيري كه مشخص مي كنيم وجود دارد:

```
Dim DLLFile As String
Dim DLLFiles As Integer
DLLFile =Dir("C:\WINNT \SYSTEM \*.DLL")
If DLLFile <>"" Then DLLFiles =DLLFiles +1
While DLLFile <>""
      DLLFile =Dir()
      DLLFiles =DLLFiles +1
End While
MsgBox(DLLFiles)
```

در اين دستورات، تابع Dir() ابتدا با يكي از آرگومان هايش فراخواني مي شود.
اگر يك فايل DLL در مسير مشخص شده وجود داشت، نام آن برگشت داده مي شود و تابع مجددا فراخواني مي شود اما اين بار بدون آرگومان.
هر بار كه تابع، نام يك فايل DLL را برگشت مي دهد يك واحد به متغير DLLFiles اضافه مي شود. اين كار براي تمامي فايل هاي DLL در مسير مشخص شده ادامه مي يابد و در آخر، مقدار متغير DLLFiles، تعداد فايل هاي مشخص شده در مسير تعيين شده را دارا مي باشد.
خوب مي خواهيم تابع را گسترش دهيم و فقط فايل هاي DLL اي را جستجو كنيم كه داراي صفت Hidden (مخفي) هستند.
پس تابع به شكل زير مي شود:

```
DLLFile =Dir("C:\WINNT \SYSTEM \*.DLL",FileAttribute.Hidden)
```

شما همچنين قادر به تركيب چندين صفت با هم هستيد. مثلا فايل هاي DLL اي كه داراي صفات مخفي و فقط خواندني و سيستمي هستند.
براي تعيين نام زيرپوشه هايي كه در يك پوشه قرار دارند بايد از صفت FileAttribute.Directory استفاده كنيد كه نام پوشه ايي را بر گشت مي دهد كه نرمال يا اصطلاحا بدون صفت باشد. (در VB.6.0 با نام vbDirectory شناخته مي شود)
براي تعيين اينكه مقدار ورودي، فايل است يا پوشه بايد خواص مقدار ورودي را با استفاده از تابع GetAttr() بررسي كنيد.
مثال زير تعداد زيرپوشه هاي موجود در پوشه System را به دست مي آورد:

```
Dim path,FName As String
Dim totFolders As Integer
path ="C:\Windows\System\"
FName =Dir(path,FileAttribute.Directory)
While FName <>""
      If (GetAttr(path &FName)And FileAttribute.Directory)=_
        FileAttribute.Directory Then
        Console.WriteLine(FName)
        totFolders =totFolders + 1
      End If
      FName =Dir()
End While
Console.WriteLine("Found" & totFolders & "folders")
```

## FileCopy(source_file,dest_file)

اين تابع، يك فايل را به يك مكان جديد بر روي هارددیسك كپي مي كند.
آرگومان source_file، نام فايلي است كه قرار است كپي شود.
اگر فايل در پوشه جاري باشد فقط ذكر نام فايل كفايت مي كند در غير اينصورت بايد مسير كامل فايل را ذكر كنيد.
آرگومان dest_file نام فايل و مسيري است كه فايل قرار است به آنجا كپي شود.
سه نكته مهم در رابطه با استفاده از اين تابع:

۱) مي توانيد براي فايلي كه قرار است به مكان ديگري كپي شود نام جديدي در نظر بگيريد.
۲) در صورتي كه در موقع كپي فايل، برنامه ي ديگري در حال استفاده از فايل باشد، فايل كپي نخواهد شد.
۳) در اين تابع مجاز به استفاده از كاراكترهاي جايگزين نيستيد.(* و ؟) به عبارت ديگر نمي توانيد تنها با يك بار فراخواني اين تابع، چندين فايل را همزمان كپي كنيد.

مثال زير را در نظر بگيريد:

```
source ="C:\VB\Examples\Files.txt"
destination ="D:\MasteringVB\NewName.txt"
FileCopy(source,destination)
```

اين دستورات، فايل Files.txt را به مكان جديدي كپي كرده و نام فايل كپي شده را به NewName.txt تغيير مي دهد.
همچنين مي توانيد همان نام قبلي فايل را بنويسيد.

## Rename(oldpath,newpath)

با كمك اين تابع مي توانيد نام يك فايل يا پوشه را تغيير دهيد.
آرگومان oldpath، نام و مسير فايل يا پوشه اي را تعيين مي كند كه قرار است تغيير نام پيدا كند و نام جديد با استفاده از آرگومان newpath تعيين مي شود.
مثال زير را در نظر بگيريد:

```
Rename("C:\Users","C:\All Users")
```

اين دستور، پوشه Users را در درايو C به All Users تغيير نام مي دهد.

نكات مهم در استفاده از اين تابع:
تغيير نام پوشه، تاثيري در شامل بودن زيرپوشه ها و فايل ها در آن ندارد.
اگر شما قصد داشته باشيد كه دو پوشه تو در تو را همزمان تغيير نام دهيد يك خطا رخ خواهد داد.
به عنوان مثال:

```
Rename(“C:\Users \New User”, “C:\All Users \User1”)
```

با اجراي اين دستور، يك خطا اتفاق خواهد افتاد.
براي تغيير نام دو يا چند پوشه ي تو در تو بايد آنها را يكي يكي تغيير نام دهيد.
تابع Rename همچنين در صورت نياز، در هنگام تغيير نام يك فايل قادر به **انتقال** آن (Cut) به يك مكان جديد بر روي هارددیسك است.
توجه داشته باشيد كه اين تابع قادر به انتقال پوشه ها نيست.
به عنوام مثال فرمان زير فايل Profile1.cps را در مسير مشخص شده در درايو C همراه با تغيير نام آن به UserProfile.cps به مسير مشخص شده در درايو D منتقل مي كند:

```
Rename(“C:\AllUsers\User1\Profile1.cps”, “D:\New User\UserProfile.cps”)
```

اين تابع قادر به ايجاد يك پوشه ي جديد نيست.
تابع Reaname بر روي فايل هاي باز يا فايل هايي كه توسط برنامه هاي ديگر، همزمان در حال استفاده هستند كاربرد ندارد.
در اين تابع مجاز به استفاده از كاراكترهاي جايگزين(* و ؟) نيستيد.

## توابعي كه براي شناسايي نوع داده ها به كار مي روند (Data Type Identification):

توابع موجود در اين بخش، تنها نوع داده را تعيين مي كنند.

### IsArray(variable)

اين تابع تعيين مي كند كه يك متغير از نوع آرايه است يا خير. چنانچه از نوع آرايه باشد، تابع مقدار True و گرنه False را بر مي گرداند.
آرگومان variable نام متغيري است كه بايد بررسي شود.
مثال هاي زير را در نظر بگيريد:

```
Dim names(100)
IsArray(names)
```

خروجي اين تابع True است چون كه names به عنوان يك آرايه ۱۰۰ تايي تعريف شده است.

```
Dim A As Integer
IsArray(A)
```

خروجي تابع در اين حالت، False است زيرا A متغيري از نوع صحيح تعريف شده است.

### IsDate(expression)

اين تابع تعيين مي كند كه يك مقدار از نوع تاريخ است يا خير. چنانچه مقدار از نوع تاريخ باشد، تابع True و گرنه False را بر مي گرداند.
آرگومان expression عبارتي است كه بايد چك شود.
مثال زير را در نظر بگيريد:

```
BDate =InputBox("Please enter your birth date")
If IsDate(BDate)Then
  MsgBox "Date accepted"
End If
```

### IsDBNull(expression)

اين تابع در صورتي كه يك مقدار، DBNull باشد، مقدار True را برگشت مي دهد.
DBValue مقداري است كه وجود ندارد و با مقدار Nothing تفاوت دارد.
DBValue نمايانگر آن است كه از ذكر يك مقدار خودداري شده و يا نوع آن تعريف نشده.

### IsNothing(expression)

اين تابع تعيين مي كند كه آيا مقداري (متغيري) كه از نوع Object (شي) تعريف شده داراي مقدار است يا خير. اگر مقداراي به آن نسبت داده شده باشد، خروجي تابع True و گرنه False است.
مثال زير را در نظر بگيريد:

```
Dim obj As Object
```

بعد از اين تعريف، خروجي عبارت IsNothing(obj) برابر True است چون هنوز مقداري به متغير ما نسبت داده نشده است.
حالا ما يك مقدار از نوع Object (شي) را به اين متغير نسبت مي دهيم:

```
obj =New Rectangle(10,100,12,12)
```

بعد از اجراي اين دستور، متغير ما داراي مقدار است و خروجي عبارت IsNothing(obj) برابر False مي باشد.
در اين مثال مي توان با نوشتن فرمان Set obj=Nothing، فضاي اشغال شده حافظه را كه براي ايجاد اين شي در اختيار برنامه قرار گرفته است به حافظه اصلي برگرداند.

به دو شكل مي توان اين تابع را به كار برد:
١) If IsNothing(obj) Then
٢) If obj Is Nothing Then
در شكل دوم، از كلمه ي كليدي (Is) استفاده شده است.

## IsNumeric(expression)

اين تابع يك رشته را تست مي كند و چنانچه كليه كاراكترهاي رشته، ارقام بين 0 تا 9 باشند، مقدار True و گرنه مقدار False را برمي گرداند.
مثال زير را در نظر بگيريد:

```
age =InputBox(“Please enter your age”)
If Not IsNumeric(age)Then
  MsgBox(“Please try again,this time with a valid number”)
End If
```

در اين مثال با استفاده از تابع ()InputBox، سن كاربر درخواست مي شود و سپس مقدار ورودي كاربر با تابع ()IsNumeric چك مي شود. چنانچه كاربر كاراكتري غير عددي را وارد كرده بود، به او اطلاع داده مي شود.

## IsReference(expression)

اين تابع تعيين مي كند كه يك متغير از نوع Object يا Reference اضافه شده به برنامه است يا خير و خروجي آن يا True و يا False است.

## TypeName(variable_name)

اين تابع، نوع يك متغير را به صورت رشته اي برمي گرداند.
ممكن است متغيري كه شما مي خواهيد نوع آن را با اين تابع به دست آوريد، تعريف شده يا نشده باشد.
فرض مي كنيم كه متغيرهاي زير تعريف شده اند:

```
Dim name As Integer
Dim a
```

دستورات زير نتايج زير را توليد مي كنند:

```
Console.WriteLine(TypeName(name))
Integer
Console.WriteLine(TypeName(a))
Nothing
a =“I ’m a string”
Console.WriteLine(TypeName(a))
String
```

## VarType(variable)

اين تابع، نوع يك متغير را برمي گرداند.
اين تابع همانند تابع ()TypeName عمل مي كند با اين تفاوت كه اين تابع، يك مقدار عددي را برگشت مي دهد ولي تابع TypeName يك مقدار رشته اي را برگشت خواهد داد.
آرگومان Variable، متغيري است كه نوع آن بايد مشخص شود.
مقاديري كه اين تابع برگشت مي دهد در زير ليست شده است:

| مقاديري كه تابع VarType() برگشت مي دهد |
| --- |
| مقدار |
| Array |
| Boolean |
| Byte |
| Char |
| Currency |
| DataObject |
| Date |
| Decimal |
| Double |
| Empty |
| Error |
| Integer |
| Long |
| Null |
| Object |
| Short |
| Single |
| String |
| UserDefinedType |
| Variant |

مقدار ثابت عددي اين مقادير توسط تابع VarType() برگشت داده خواهد شد.

## توابع تبديل نوع متغير (Variable Type Conversion):

متغيرها قابليت تبديل شدن به يكديگر را دارند.
تمامي توابع اين بخش آرگومان هاي عددي مي پذيرند و آنها را تبديل به نوع ديگري از متغيرها مي كنند.
اين توابع را در جدول زير مشاهده مي كنيد:

| توابع تبديل نوع متغير | |
|---|---|
| **نام تابع** | **عملكرد** |
| **CBool(*expression* )** | تبديل به Boolean |
| **CByte(*expression* )** | تبديل به Byte |
| **CChar(*expression* )** | تبديل به Char |
| **CDate(*expression* )** | تبديل به Date |
| **CDec(*expression* )** | تبديل به Decimal |
| **CDbl(*expression* )** | تبديل به Double |
| **CInt(*expression* )** | تبديل به Integer |
| **CLng(*expression* )** | تبديل به Long |
| **CObj(*expression* )** | تبديل به Object |
| **Cshort(*expression* )** | تبديل به Short |
| **CSng(*expression* )** | تبديل به Single |
| **CStr(*expression* )** | تبديل به String |
| **CType(*expression,type* )** | نوع سفارشي |

### CType(varName,typeName)

اين تابع، يك مقدار يا عبارت را كه مقدار آن در آرگومان varName تعيين شده به نوعي كه در آرگومان typeName مشخص شده تبديل مي كند.
دستورات زير را در نظر بگيريد:

```
CType(1000, System.Double)
CType("1000", System.Double)
```

اولين دستور، مقدار ۱۰۰۰ را كه از نوع عدد صحيح (Integer) است و دستور دوم مقدار ۱۰۰۰ را كه از نوع رشته اي (String) است به نوع عدد اعشاري با دقت مضاعف (Double) تبديل مي كند.

## توابعي كه با رشته ها سر و كار دارند (String Manipulation):

در VB.NET توابع بسيار مهم، مفيد و كارآمدي جهت كار با رشته ها وجود دارند.
ميانگيني كه در رابطه با عملكرد برنامه هاي كاربردي گرفته شده نشان مي دهد كه توابع رشته اي بيش از توابع عددي كاربرد دارند.
در اين بخش دو تابع همنام به نام هاي ()Mid و Mid وجود دارد كه عملكردي شبيه به هم داشته و در جاي خود بحث خواهند شد.
تمامي توابع رشته اي VB.NET داراي متد و خواص مشابه و برابر ي در كلاس System.String همانند كلاس StringBuilder هستند.

### Asc(character) , AscW(string)

تابع ()Asc، كد اسكي يك كاراكتر را كه نام در آرگومان character ذكر شده، برگشت مي دهد.
اين تابع بر روي همه سيستم ها كار مي كند حتي اگر آن سيستم از Unicode پشتيباني نكند.
اگر در آرگومان character به جاي كاراكتر، يك كلمه يا رشته آورده شود، اولين كاراكتر رشته در نظر گرفته مي شود.
تابع ()AscW عملكردي شبيه به تابع ()Asc دارد با اين تفاوت كه كد اسكي كاراكتر را به صورت Unicode برگشت مي دهد در صورتي كه تابع ()Asc كد اسكي را به صورت بايت برگشت مي دهد.
از اين توابع مي توانيد جهت تبديل كاراكتر كليد فشرده شده توسط كاربر به كد اسكي معادل آن استفاده كنيد.
به عنوان مثال نتيجه خروجي (”Ali“)ASC، عدد 65 است و نتيجه خروجي (”N“)ASC، عدد 78 است.

### Chr(number) , ChrW(number)

اين دو تابع دقيقا عكس توابع ()Asc و ()AscW عمل مي كنند.
تابع ()Chr، كاراكتر معادل كد اسكي يك عدد از نوع بايت را برگشت مي دهد.
از اين تابع مي توانيد جهت چاپ حروف يا كاراكترهايي كه بر روي صفحه كليد وجود ندارند همانند شكلك ها يا حروفي كه كد اسكي آنها از 127 بيشتر است استفاده كنيد.
تابع ()ChrW نيز يك كاراكتر از نوع Unicode يك مقدار عددي را برگشت مي دهد.
از اين توابع مي توانيد جهت تبديل كد اسكي كليد فشرده شده توسط كاربر به كاراكتر معادل آن استفاده كنيد.

### LCase(string) , UCase(string)

تابع ()Lcase، در يك رشته، حروف بزرگ رشته را به حروف كوچك تبديل مي كند.
تابع ()Ucase، در يك رشته، حروف كوچك رشته را به حروف بزرگ تبديل مي كند.
مثال زير را در نظر بگيريد:

```
Title =“Visual Basic.Net”
LTitle =LCase(Title)
UTitle =UCase(Title)
```

بعد از اجراي اين دستورات متغيرهاي Ltitle و Utitle، داراي مقادير زير خواهند شد.

```
Ltitle=”visual basic.net"
Utitle=”VISUAL BASIC.NET”
```

## InStr([startPos,] string1,string2 [,compare ])

اين تابع، زيررشته اي را در رشته ديگر جستجو كرده و محل وجود زيررشته در رشته را برمي گرداند.
اولين آرگومان، اختياري است و مكاني از رشته است كه جستجو بايد از آنجا آغاز شود. اگر ذكر نشود 1 در نظر گرفته مي شود و جستجو از ابتداي رشته آغاز خواهد شد.
اين تابع، آرگومان string2 را در آرگومان string1 جستجو مي كند.
مثال زير را در نظر بگيريد:

```
str1 =“The quick brown fox jumped over the lazy dog”
str2 =“the”
Pos =Instr(str1, str2)
```

بعد از اجراي دستورات فوق، متغير Pos داراي مقدار 33 خواهد شد.

نكات مهم:
۱) اگر رشته اي كه جستجو براي آن آغاز مي شود بيش از يك بار در رشته جستجو شونده تكرار شده باشد **و كلمه باشد**، محل آخرين وجود آن برگشت داده مي شود همان طور كه در اين مثال كلمه the، دوبار تكرار شده است اما اگر **قسمتي از يك كلمه باشد**، حتي اگر چندين بار هم تكرار شده باشد، محل اولين وقوع آن برگشت داده خواهد شد.
۲)جستجو به طور پيش فرض با توجه به كوچك يا بزرگ بودن حروف انجام خواهد شد يعني فرضا كلمات the، The و THE با هم تفاوت دارند.
و اين تنظيم به وسيله تغيير آخرين آرگومان يعني compare كه به طور پيش فرض مقدار CompareMethod.Binary را دارد مي تواند تغيير كند.
براي اينكه بزرگ يا كوچك بودن حروف، در جستجو ناديده گرفته شود، مقدار آرگومان Compare را به CompareMethod.Text تغيير دهيد.
مثال زير را در نظر بگيريد:

```
str1 =“The quick brown fox jumped over the lazy dog”
str2 =“the”
Pos =InStr(1, str1, str2, CompareMethod.Text)
```

پس از اجراي اين دستورات مقدار متغير pos، 1 خواهد شد. در صورتي كه آرگومان compare را به CompareMethod.Binary تغيير دهيم مقدار متغير pos، 33 خواهد شد.

## InStrRev(string1,string2 [,start ][,compare ])

اين تابع همانند تابع InStr() عمل مي كند با اين تفاوت كه در تابع InStr() جستجو از ابتداي رشته (چپ به راست) ولي در تابع InStrRev جستجو از انتهاي رشته (راست به چپ) انجام مي شود.
توجه داشته باشيد كه جاي آرگومان start در اين تابع عوض شده.

## StrComp(string1,string2 [,compare ])

اين تابع، دو رشته را با يكديگر مقايسه كرده و حاصل آن را برمي گرداند.
آرگومان هاي string1 و string2 رشته هايي هستند كه بايد با يكديگر مقايسه شوند و آرگومان compare، روش مقايسه را تعيين مي كند.
اگر دو رشته با هم برابر باشند، تابع مقدار **0** را برمي گرداند.
اگر رشته اول بزرگتر از رشته دوم باشد، تابع مقدار **1** را برمي گرداند.
اگر رشته دوم بزرگتر از رشته اول باشد، تابع مقدار **-1** را برمي گرداند.
عمل مقايسه به طور پيش فرض با در نظر گرفتن بزرگ يا كوچك بودن حروف انجام مي شود.
براي اينكه بزرگ يا كوچك بودن حروف، در مقايسه ناديده گرفته شود، مقدار آرگومان Compare را به CompareMethod.Text تغيير دهيد.
مثال زير را در نظر بگيريد:

```
Result=StrComp(“Sybex”, “SYBEX”)
```
پس از اجراي اين دستور، مقدار Result، 1 خواهد شد زيرا عبارت Sybex از SYBEX بزرگتر است و اين بدان علت است كه در ترتيب حروف اسكي (ASCII)، حرف y كوچك بعد از حرف Y بزرگ قرار دارد. به عبارت ديگر حروف كوچك بر حروف بزرگ ارجعيت دارند.
مثال ديگر:
```
Result=StrComp(“Sybex”, “SYBEX”, CompareMethod.Text)
```

اين همان مثال قبلي است با اين تفاوت كه در اين مثال، بزرگ و كوچك بودن حروف تاثيري در نتيجه مقايسه نخواهد داشت پس دو رشته در اين حال با يكديگر برابر بوده و مقدار Result، 0 خواهد شد.

## Left(string,number)

اين تابع، از سمت چپ يك رشته، رشته اي با طول معين را برگشت مي دهد.
آرگومان string، رشته اي است كه بايد رشته اي به طول تعيين شده در آرگومان number از سمت چپ آن برگشت داده شود.
مثال زير را در نظر بگيريد:
```
StrX=”Visual Basic.NET”
StrGet=Left(StrX, 6)
```

پس از اجراي دستورات فوق، 6 كاراكتر اول از سمت چپ متغير StrX در متغير StrGet قرار خواهد گرفت و مقدار متغير StrGet، برابر با ”Visual“ خواهد شد.

## Right(string,number)

اين تابع، از سمت راست يك رشته، رشته اي با طول معين را برگشت مي دهد.
آرگومان string، رشته اي است كه بايد رشته اي به طول تعيين شده در آرگومان number از سمت راست آن برگشت داده شود.
مثال زير را در نظر بگيريد:
```
StrX=”Visual Basic.NET”
StrGet=Right(StrX, 3)
```

پس از اجراي دستورات فوق، 3 كاراكتر اول از سمت راست متغير StrX در متغير StrGet قرار خواهد گرفت و مقدار متغير StrGet، برابر با ”NET“ خواهد شد.

## Mid(string,start,[length ])

اين تابع، بخشي از يك رشته را برگشت مي دهد.
آرگومان string رشته اي است كه بايد بخشي از آن برگشت داده شود.
آرگومان start مكاني از رشته را مشخص مي كند كه استخراج رشته بايد از آن مكان شروع شود و آرگومان length طول رشته برگشتي را مشخص مي كند.
نكات مهم:
۱) استخراج رشته از سمت چپ به راست انجام مي شود.
۲) اگر از ذكر آرگومان اختياري length خودداري شود، تمامي كاراكترها از مكاني كه توسط آرگومان start مشخص شده تا به آخر رشته استخراج مي شوند.
۳) اگر مقدار آرگومان length بيش از تعداد كاراكترهايي باشد كه بعد از آرگومان start در رشته وجود دارند، تمامي كاراكترهاي مشخص شده در رشته، از مكان Start تا به انتها برگشت داده مي شوند.

## Mid(string,start [,length ])=new_string

در حالت پيشرفته تابع Mid() اين تابع وجود دارد كه تا حدي شبيه به تابع Mid() عمل مي كند.

اين تابع رشته اي را در يك رشته ديگر جايگزين مي كند.
آرگومان string رشته اي است كه مقدار new_String بايد در آن جايگزين شود و آرگومان Start موقعيتي از آرگومان (رشته) string را مشخص مي كند كه عمل جايگزيني بايد از آنجا شروع شود و آرگومان length، طول رشته اي را كه بايد جايگزين شود تعيين مي كند.
نكات مهم:
۱) اگر از ذكر مقدار new_string خودداري شود، تمامي كاراكترها در رشته از نقطه تعيين شده در آن تا به انتها جايگزين مي شوند.
۲) اگر برنامه اي مي نويسيد كه با رشته هاي طولاني و متن هاي زياد سر و كار دارد حتما از كلاس StringBuilder استفاده كنيد.
متغيرهاي كلاس StringBuilder از نوع متني(Text) هستند و كامپايلر .NET بسيار بهتر از متغيرهاي رشته اي (String) قادر به كار با آنهاست. (كلاس برنامه را هميشه حفظ كنيد!)

## Len(string)

اين تابع، طول يك رشته را برمي گرداند.
آرگومان string، رشته اي است كه طول آن بايد برگشت داده شود.
مثال:

```
Name =InputBox("Enter your first name")
NameLen =Len(Name)
```

بعد از اجراي اين دستورات، مقدار متغير NameLen، طول رشته اي است كه كاربر در كادر محاوره InputBox() وارد كرده.
از اين تابع مي توانيد براي چك كردن اينكه آيا كاربر مقداري را در مكاني وارده كرده يا نه، استفاده كنيد.
به عنوان مثال:

```
If Len(Text1.Text) = 0 Then
  MsgBox ("Name field can 't be empty")
Else
  MsgBox ("Thank you for registering")
EndIf
```

در اين دستورات، مقدار TextBox چك مي شود كه آيا كاربر چيزي در آن وارد كرده يا نه و سپس پيغام مناسب ظاهر خواهد شد.
تابع Len() قادر به دريافت هر نوع مقدار غير رشته اي نيز در آرگومان string خود مي باشد.
در اين حالت مقدار برگشتي اين تابع، **تعداد بايت هاي** مورد نياز جهت نگهداري مقدار وارد شده در حافظه مي باشد **نه طول آن**.
به عنوان مثال:

```
Len(12.01)
```

در VB.NET، مقادير اعشاري به طور پيش فرض از نوع Double (اعشاري با دقت مضاعف) در نظر گرفته مي شود پس خروجي تابع در اين حالت، عدد 8 يعني تعداد بايت هاي لازم جهت نگهداري يك مقدار از Double در حافظه خواهد بود.
و به عنوان آخرين مثال:

```
Len(12)
```

در اين حالت مقدار برگشتي، 4 خواهد بود. زيرا ميزان فضاي لازم جهت نگهداري يك عدد از نوع Integer، 4 بايت است.

## LTrim(string),RTrim(string),Trim(string)

اين توابع، فضاهاي خالي يك رشته را هر كدام به نحوي حذف مي كنند.
تابع LTrim()، فضاي خالي سمت چپ يك رشته را حذف مي كند.
تابع RTrim()، فضاي خالي سمت راست يك رشته را حذف مي كند.
تابع Trim()، فضاي خالي سمت چپ و راست يك رشته را حذف مي كند.

آرگومان string در هر سه تابع، رشته اي است كه فضاي خالي آن بايد حذف شود.
مقدار برگشتي هر سه ي اين توابع، مقداري رشته اي و از نوع String مي باشد و عبارتي است كه فضاهاي خالي آن بنا به نوع تابع، حذف شده است.
كاربرد اين تابع معمولا در چك كردن مقادير ورودي توسط كاربر براي تشخيص خالي نبودن مقدار ورودي است.
مثلا دستورات زير چك مي كند كه آيا مقدار متغير E_Mail، خالي است يا خير و در صورت خالي بودن آن پيغامي ظاهر مي گردد.

```
If E_Mail = “” Then
  MsgBox (“Applications without an e-mail address won ’t be processed”)
End If
```

حال در نظر بگيريد كه كاربر به جاي تايپ كاراكتري در مكان مورد نظر،يك يا چندين بار از كليد Space استفاده كند. از آنجا كه Space نيز نوعي كاراكتر است پس دستورات بالا كاربرد ندارد.
پس در اين حالت براي دقت چك كردن بايد از توابع بالا استفاده كرد.
معمولا تابع ()Trim از دو تابع ديگر كاربرد بيشتري دارد.
پس دستورات بالا را به شكل زير اصلاح مي كنيم:

```
If Trim(E_Mail)=“” Then
  MsgBox (“Invalid Entry!”)
End If
```

## Space(number)

اين تابع، يك رشته كاراكتري از فضاي خالي (Blanck) ايجاد مي كند.
آرگومان number، تعداد كاراكتر فضاي خالي است كه بايد ايجاد شود.
استفاده از اين تابع براي قالب دهي به خروجي هاي برنامه و ايجاد بافر براي برخي از توابع API مفيد است.

## StrDup(number,character)

اين تابع، يك كاراكتر را در يك رشته به تعداد معين بار تكرار مي كند.
آرگومان number عددي است كه تعداد تكرار كاراكتر را كه به وسيله آرگومان character تعيين مي شود، مشخص مي كند.
آرگومان character مي تواند كاراكتر يا كد اسكي باشد.
اين تابع جايگزين تابع ()String در VB.6.0 شده است.
مثال زير را در نظر بگيريد:

```
Result=StrDup(12, “*”)
```

با اجراي اين دستور، كاراكتر (*) به تعداد 12 بار تكرار خواهد شد و نتيجه در متغير Result قرار خواهد گرفت. يعني متغير Result داراي مقدار زير خواهد شد:

```
Result=”************”
```

## StrConv(string,conversion)

اين تابع، يك مقدار رشته اي را كه به شكل ديگري تبديل شده است برگشت مي دهد.
آرگومان string، رشته اي است كه بايد تبديل شود و آرگومان conversion نوع تبديل را مشخص مي كند كه مي تواند يكي از موارد زير باشد:

| مقادير پارامتر convesion در تابع StrConv() | |
|---|---|
| **مقدار** | **عملكرد** |
| **UpperCase** | تمامي كاراكترهاي موجود در رشته به حروف بزرگ تبديل مي شوند. |
| **LowerCase** | تمامي كاراكترهاي موجد در رشته به حروف |

| | |
|---|---|
| | كوچك تبديل مي شوند. |
| **ProperCase** | اولين كاراكتر هر كلمه در رشته به حرف بزرگ تبديل مي شود. |
| **Wide Narrow** | كاراكترهاي تك بايتي در رشته به كاراكترهاي دوبايتي تبديل مي شوند |
| **Narrow** | كاراكترهاي دوبايتي در رشته به كاراكترهاي تك بايتي تبديل مي شوند. |
| **Katakana** | كاراكترهاي Hiragana به كاراكترهاي Katakana تبديل مي شوند. |
| **Hiragana** | كاراكترهاي Katakana به كاراكترهاي Hiragana تبديل مي شوند. |
| **Traditional** | كاراكترهاي ساده چيني به كاراكترهاي سنتي چيني تبديل مي شوند. |
| **Simplified** | كاراكترهاي سنتي چيني به كاراكترهاي ساده چيني تبديل مي شوند |

براي تبديل هاي چندتايي مي توانيد اين مقادير را با علامت + با هم AND كنيد.
به مثال زير توجه كنيد:

```
newString =StrConv(MyString, vbStrConv.LowerCase + vbStrConv.Unicode)
```

در اين دستور، عبارت موجود در متغير MyString، به حروف كوچك و قالب Unicode تبديل خواهد شد.

### StrReverse(string)

اين تابع، يك مقدار رشته اي را معكوس مي كند.
مثال:

```
Result=StrReverse(“VisualBasic”)
```

پس از اجراي اين دستور، مقدار متغير Result برابر با "lausiVcisaB" خواهد شد.

### Filter(inputStrings,value [,include ][,compare ])

اين تابع به بيان ساده يك رشته يا زيررشته را در يك آرايه رشته اي جستجو كرده و در صورت پيدا كردن آن، يك آرايه را به عنوان خروجي مي دهد كه خانه يا خانه هاي آن تا انديس مقداري كه در آرايه اصلي پيدا شده و با آن مقدار پر مي شود.
در حقيقت مقدار برگشتي كه در آرايه قرار مي گيرد، شامل يكي از مقادير خانه هاي آرايه جستجو شونده است.
آرگومان inputstring، يك آرايه يك بعدي و رشته اي است كه عمل جستجو بايد در آن انجام شود. دقت داشته باشيد كه اين آرايه حتما بايد به صورت يك بعدي تعريف شود.
آرگومان value مقداري رشته اي است كه جستجو بايد براي آن انجام شود.
دو آرگومان آخر، اختياري هستند.
آرگومان include مشخص مي كنه كه آيا كل مقدار value بايد در آرايه باشد يا قسمتي از آن هم مي تواند به عنوان شرط جستجو به كار رود و يكي از مقادير True يا False را مي پذيرد.
آرگومان compare، نوع عمل مقايسه را تعيين مي كند و مشخص مي كند كه آيا بزرگ يا كوچك بودن حروف تاثيري در مقايسه و جستجو داشته باشد يا خير و يكي از مقادير CompareMethod.Binary (پيش فرض يعني در نظر گرفته مي شود) و يا CompareMethod.Text (يعني در نظر گرفته نمي شود) را مي پذيرد.
تعداد خانه هاي آرايه ي برگشت داده شده بستگي به انديس محل وجود رشته اي است كه در آرايه پيدا شده. فرضا اگر مقدار مورد نظر ما در خانه ي 4 آرايه پيدا شد، تعداد خانه هاي آرايه برگشتي 4 خانه خواهد بود.

پس، از آنجا كه خانه اي كه مقدار مورد نظر در آن پيدا مي شود از قبل قابل پيش بيني نيست نبايد براي خانه هاي آرايه برگشتي، تعداد تعريف كنيم و يا به عبارت ديگر بايد يك آرايه بدون بعد تعريف كنيم.
مثال زير را در نظر بگيريد:

```
Dim selNames()As String
Dim Names()As String ={“Ali”, “Mehdi”, “Mehdi”, “Behrouz”, “Sina”}
```

در اين مثال، آرايه اي به نام Names تعريف شده كه پنج خانه ي اول آن با 5 مقدار رشته اي پر شده است. اين آرايه اي است كه ما در آن قصد جستجوي مقدار خود را داريم.
حال مي خواهيم مقدار ذخيره شده در متغيري رشته اي به نام myName را در اين آرايه جستجو كنيم. پس تابع ()Filter را به شكل زير فراخواني مي كنيم:

```
selNames =Filter(Names,myName)
```

پس از اجراي اين دستور اگر مقدار ذخيره شده در متغير myName در آرايه نباشد، در آن صورت آرايه selNames كه آن را به صورت آرايه اي بدون بعد تعريف كرديم آرايه اي بدون خانه خواهد بود و يا به عبارت ديگر خاصيت Length آن برابر با 0 خواهد بود.
اگر مقدار ذخيره شده در متغير myName، ”Ali“ باشد، حد بالاي آرايه selNames صفر خواهد بود و مقدار اولين خانه آرايه برابر با ”Ali“ مي شود يعني: selNames(0)=”Ali“
اگر مقدار ذخيره شده در متغير myName، ”Mehdi“ باشد، حد بالاي آرايه selNames، يك خواهد بود و مقادير خانه هاي اول و دوم آرايه برابر با ”Mehdi“ مي شود يعني:

```
selNames(0)=”Mehdi”
selNames(1)=”Mehdi”
```

## Replace(expression,find,replacewith [,start ][,count ][,compare ])

اين تابع، يك رشته را در رشته ديگر جستجو كرده و رشته سوم را جايگزين رشته جستجو شده مي كند.
آرگومان expression، رشته اي است كه براي جايگزيني جستجو مي گردد.
آرگومان find، رشته اي است كه جستجو مي گردد.
آرگومان replacewith، رشته اي است كه جايگزين رشته پيدا شده مي شود.
آرگومان اختياري start، عددي است كه مكان شروع جستجو در رشته را تعيين مي كند و اگر در نظر گرفته نشود، جستجو از ابتداي رشته شروع مي شود.
آرگومان count، مقداري عددي است كه تعداد دفعاتي را كه جايگزيني بايد انجام گيرد، تعيين مي كند. اگر ذكر نشود، كليه جايگزيني ها انجام خواهد شد.
و در نهايت آرگومان compare، نوع مقايسه را تعيين مي كند كه در توابع قبلي در مورد اين آرگومان توضيح داده شد.
مثال زير را در نظر بگيريد:

```
S=”Visual Basic.NET as powerfull as C#.NET”
K=Replace (S, “NET”, “bes”)
Console.WriteLn(K)
```

خروجي اين قطعه برنامه پس از اجرا به شكل زير خواهد بود:

```
”Visual Basic.bes  as powerfull as C#.bes”
```

## Join(list [,delimiter ])

اين تابع، مقادير موجود در يك آرايه يك بعدي – رشته اي را به صورت يك رشته و پشت سر هم برگشت مي دهد.
آرگومان list، آرايه اي يك بعدي و شامل مقاديري رشته اي است كه قصد بازگشت آنها به صورت يك رشته را داريد و آرگومان اختياري delimiter، يك كاراكتر رشته اي است كه براي جدا كردن رشته ها از هم در نتيجه خروجي به كار مي رود و در صورتي كه ذكر نشود، يك فاصله خالي در نظر گرفته مي شود و اگر مقدار آن تهي يا ”“ تعيين شود، مقادير خانه هاي آرايه پشت سر هم و بدون فاصله در خروجي ظاهر خواهند شد.
مثال زير را در نظر بگيريد:

```
Dim MyArr(1 To 5) As String
Dim Result As String
MyArr(1) = "Behrouz”
```

```
MyArr(2) = "Rad"
Result = Join(MyArr, "|")
MsgBox Result
```

بعد از اجراي اين دستورات، خروجي به شكل زير ظاهر خواهد شد:

```
"Behrouz|Rad|||"
```

دقت داشته باشيد كه اگر تمام خانه هاي آرايه پر نشده باشند، به جاي آنها كاراكتري كه در آرگومان delimiter تعيين شده قرار خواهد گرفت.
تعداد كاراكترهاي delimiter كه ظاهر خواهد شد، تعداد خانه هاي آرايه منهاي يك است.

## Split(expression [,delimiter ][,count ][,compare ])

اين تابع، يك رشته را به تعدادي زيررشته ي كوچكتر تقسيم كرده و هر كدام را در يكي از خانه هاي آرايه قرار مي دهد و يا به عبارت بهتر هر كدام از كلمات رشته را در يك خانه ي آرايه قرار مي دهد.
خروجي اين تابع، يك آرايه يك بعدي است كه هر كدام از خانه هاي آن داراي يكي از مقادير زيررشته هاست.
آرگومان expression، رشته اي است كه بايد به زيررشته هاي كوچكتر تقسيم شود.
آرگومان اختياري delimiter، رشته اي است كه تقسيم بندي بايد بر اساس آن انجام شود.
اگر ذكر نشود فضاي خالي در نظر گرفته مي شود ولي اگر تهي يا "" ذكر شود، كل رشته در اولين عنصر آرايه قرار مي گيرد.
آرگومان اختياري count نيز تعداد زيررشته هايي را كه بايد برگشت داده شوند مشخص مي كند.
اگر مقدار آن **-1** باشد، تمامي زيررشته ها برگشت داده خواهند شد.
آرگومان compare كه نوع مقايسه را تعيين مي كند كه در توابع قبلي توضيح داده شد.
مثال زير را در نظر بگيريد:
فرض مي كنيم متغيري رشته اي به نام path تعريف كرده ايم كه مقدار زير در آن قرار گرفته است:

```
path ="c:\win\desktop\DotNet\Examples\Controls"
```

تابع ()Split در اين حالت مي تواند هر يك اجزاي اين مقدار را كه با كاراكتر (\) از هم جدا شده اند با دستور زير جدا كرده و در خانه هاي يك آرايه به نام parts قرار دهد:

```
parts =Split("c:\win \desktop \DotNet \Examples \Controls ","\")
```

حال با دستورات زير مي توانيم نتيجه كار را مشاهده كنيم:

```
For i =0 To parts.GetUpperBound(0)
    Console.WriteLine(parts(i))
Next
```

## توابعي كه براي قالب دهي به داده ها به كار مي روند (Data Formatting):

VB.NET تمامي توابعي كه در VB.6.0 براي قالب دهي به داده ها به كار مي روند را پشتيباني مي كند.

### Format(expression [,format [,firstdayofweek [,firstweekofyear ]]])

اين تابع به دليل اهميت و گسترگي زيادي كه دارد، تابعي است كه در اين كتاب، بيشترين توضيحات در مورد آن داده شده است.
اين تابع يك مقدار را با فرمت خاصي برمي گرداند.
به بيان ساده، اين تابع شيوه نمايش اعداد، رشته ها و مقادير تاريخ و زمان را مشخص مي كند.
آرگومان expression، مي تواند يك عدد، رشته يا مقداري از نوع تاريخ باشد و آرگومان Format، يك مقدار رشته اي است كه مشخص مي كند تابع ما مقدار خروجي را با چه قالب يا فرمتي برگشت و نمايش دهد.
به عنوان مثال اگر مقدار ورودي ما يعني مقدار آرگومان expression از نوع زمان (Time) باشد و قالب آرگومان format را به شكل "hh:mm:ss" انتخاب كرده باشيم، خروجي تابع، زمان حال را نمايش مي دهد. مثلا: "18:22:37"
به عنوان مثالي ديگر، فرمان زير ، مقدار pi را به شكل عددي اعشاري با دقت مضاعف (Double) محاسبه مي كند.

```
Console.WriteLine(Math.Atan(1)*4)
```

نتيجه خروجي، عدد "3.14159265358979" خواهد بود.
اين نتيجه ي خوبي است اما اگر قرار باشد اين عدد در يك TextBox قرار بگيرد كه براي آن محدوديتي از لحاظ تعداد كاراكترهاي ورودي قائل شده اید و اين مقدار كمتر از تعداد ارقام اين عدد باشد، تمامي ارقام در TextBox قرار نمي گيرند پس بايد براي اين عدد قالب و فرمتي تعيين كرد. پس از تابع Format() به شكل زير استفاده مي كنيم:

```
Console.WriteLine(Format(Math.Atan(1)*4, "##.####"))
```

اين دستور نتيجه خروجي را به شكل 3.1416 نمايش خواهد داد.
مثال ديگر: فرض كنيم كه شما به عنوان يك حسابدار در يك بانك مشغول به كار هستيد و احتياج داريد تا در نتيجه حساب، يك علامت خاص مثلا دلار ($) يا ريال ظاهر شود.
فرض كنيد كه عدد 13,454.332345201 را در نتيجه ي يك حساب به دست مي آوريد.
مي خواهيم با استفاده از تابع Format()، علامت ($) در ابتداي نتيجه ظاهر و همچنين ميزان خرده ي پول به دست آمده فقط تا دو رقم حساب شود. پس به شكل زير عمل مي كنيم:

```
amount =13454.332345201
Console.WriteLine(Format(amount,"$###,###.##"))
```

نتيجه خروجي به صورت زير ظاهر خواهد شد:

$13,454.33

آرگومان هاي firstdayofweek و firstweekofyear، تنها در قالب دهي به مقاديري از نوع تاريخ به كار مي روند.
آرگومان firstdayofweek، نام اولين روز هفته را مشخص مي كند و مي تواند يكي از مقادير زير را بپذيرد:

| مقادیر پارامتر firstdayofweek در تابع Format() | |
|---|---|
| **مقدار** | **توضیحات** |
| **0** یا **System** | مطابق با تنظیمات سیستم |
| **1** یا **Sunday** | یکشنبه (مقدار پیش فرض) |
| **2** یا **Monday** | دوشنبه |
| **3** یا **Tuesday** | سه شنبه |
| **4** یا **Wednesday** | چهارشنبه |
| **5** یا **Thursday** | پنج شنبه |
| **6** یا **Friday** | جمعه |
| **7** یا **Saturday** | شنبه |

آرگومان firstweekofyear، اولين هفته از سال را مشخص مي كند و يكي از مقادير زير را مي پذيرد:

| مقادیر پارامتر firstweekofyear در تابع Format() | |
|---|---|
| **مقدار** | **توضیحات** |
| **0** یا **System** | مطابق با تنظیمات سیستم. |
| **1** یا **FirstJan1** | سال با هفته اول ماه ژانویه آغاز مي شود. |
| **2** یا **FirstFourDays** | سال با هفته اي آغاز مي شود كه حداقل ٤ روز آن در سال جديد باشد. |
| **3** یا **FirstFullWeek** | سال با هفته اي آغاز مي شود كه كاملا در سال جديد باشد. |

قالب هاي مختلفي براي نمايش مقادير عددي، رشته اي، تاريخ و زمان وجود دارد كه در ليست زير تمامي آنها را مي بينيد:

**كاراكترهايي كه مي توان براي قالب دهي به تاريخ و زمان به كار برد:**

**/** : اين كاراكتر، جهت جدا كردن سال، ماه و روز از يك مقدار از نوع تاريخ به كار مي رود. البته در بعضي مكان ها ممكن است از كاراكترهاي ديگري جهت اين كار استفاده شود.
**:** : اين كاراكتر، جهت جداكردن ساعت، دقيقه و ثانيه از يك مقدار از نوع زمان به كار مي رود. البته در بعضي از مكان ها ممكن است از كاراكترهاي ديگري جهت اين كار استفاده شود.
**d** : اين كاراكتر، روز يك ماه را به وسيله يك عدد مشخص مي كند. اين عدد مقداري بين 1 تا 31 است.
**dd** :اين كاراكتر، همانند كاراكتر d رفتار مي كند با اين تفاوت كه اين تابع، قبل از ارقام يك تا نه، صفر (0) مي گذارد. 01،02،03 و... اما كاراكتر d، اعداد يك تا نه را به صورت معمول نشان مي دهد يعني 1،2، 3 و...
**ddd** : اين كاراكتر، نام اختصاري يك روز را نشان مي دهد. مثلا: (Sun-Sat)
**dddd** : اين كاراكتر، نام كامل روز را نشان مي دهد. مثلا: (Sunday-Saturday)
**w** : اين كاراكتر، عددي را كه نمايانگر روز هفته است نشان مي دهد. مثلا عدد 1 براي Sunday و 7 براي Saturday
**ww** : اين كاراكتر، عددي را كه نمايانگر هفته سال است نمايش مي دهد. اين عدد بين 1 تا 54 است.
**M** : اين كاراكتر، عددي را كه نمايانگر ماه سال است نشان مي دهد. اگر با كاراكترهاي h يا hh بيايد، به جاي نمايش عدد ماه، دقيقه را نشان خواهد داد.
**MM** : اين كاراكتر همانند كاراكتر M رفتار مي كند با اين تفاوت كه اين تابع قبل از ارقام 0 تا 9، صفر (0) مي گذارد اما كاراكتر M، اعداد را به صورت معمول نمايش مي دهد.
**MMM** : اين كاراكتر، نام اختصاري يك ماه را نشان مي دهد. مثلا: (Jan-Dec)
**MMMM** : اين كاراكتر، نام كامل ماه را نشان مي دهد. مثلا: (January-December)

**q** :اين كاراكتر، يك چهارم يك سال را با اعدادي بين 1 تا 4 نمايش مي دهد. (متناظر فصل)
**y** : اين كاراكتر، عددي را كه نمايانگر روز سال است نمايش مي دهد. (1 تا 366)
**yy** : اين كاراكتر، عددي دو رقمي را نشان مي دهد كه نمايانگر سال است. (00 تا 99)
**yyyy** : اين كاراكتر، عددي 4 رقمي را نمايش مي دهد كه نمايانگر سال است. (0000 تا 9999)
**h** : اين كاراكتر، عددي را كه نمايانگر ساعت است نمايش مي دهد. (0 تا 12)
**hh** : اين كاراكتر همانند كاراكتر h رفتار مي كند با اين تفاوت كه اين تابع قبل از ارقام 0 تا 12، صفر (0) مي گذارد اما كاراكتر h، اعداد را به صورت معمول نمايش مي دهد.
**H** : اين كاراكتر، شكل كاملتر كاراكتر h است و ساعت را به صورت 0 تا 24 نمايش مي دهد. به عنوان مثال اگر ساعت فعلي سيستم 3 باشد، 3صبح را با عدد 3 و 3 بعد از ظهر را با عدد 15 نمايش مي دهد.
**HH** : اين كاراكتر، همانند كاراكتر H رفتار مي كند با اين تفاوت كه قبل از ارقام 0 تا 9، صفر (0) مي گذارد. (00 تا 09) ولي كاراكتر H، ارقام 0 تا 9 را به صورت معمول نمايش مي دهد.
**m**:اين كاراكتر، دقيقه را را نمايش مي دهد، منتها بدون گذاشتن عدد صفر قبل از ارقام 0 تا 9. بازه ي اين كاراكتر، عددي بين 0 تا 59 است.
**mm**: همانند كاراكتر m رفتار كرده با اين تفاوت كه قبل از اعداد 0 تا 9، صفر (0) مي گذارد.
**s**: اين كاراكتر، ثانيه را نمايش مي دهد، منتها بدون گذاشتن عدد صفر قبل از ارقام 0 تا 9. بازه ي اين كاراكتر بين 0 تا 59 است.
**ss**: همانند كاراكتر s رفتار كرده با اين تفاوت كه قبل از اعداد 0 تا 9، صفر (0) مي گذارد.
**AM/PM** : مشخص مي كند كه هم اكنون قبل از ظهر است (AM) يا بعد از ظهر (PM)
**am/pm**: همان كاراكتر AM/PM است، فقط قبل يا بعد بودن ظهر را با حروف كوچك نمايش مي دهد. (am/pm)
**A/P** : قبل يا بعد بودن ظهر را با حروف اختصاري مشخص مي كند. ( A يا P)
**a/p** : همان كاراكتر A/P است، فقط قبل يا بعد بودن ظهر را با حروف كوچك نمايش مي دهد.(a يا p)
**AMPM** : اين كاراكتر نيز قبل يا بعد بودن ظهر را مشخص مي كند اما به جاي اين كاراكتر، كاراكتري كه به وسيله سيستم تعيين شده قرار خواهد گرفت. به عنوان مثال در ويندوز هاي فارسي با كلمات "صبح" و "عصر" نمايش مي دهد.
براي تغيير نحوه نمايش، به سربرگ Regional Options از قسمت Regional and Language Options در Control Panel مراجعه كنيد.

كاراكترهايي كه مي توان براي قالب دهي به مقادير عددي به كار برد، ١٦ كاراكتر هستند كه نحوه ي كار با آنها بر عهده ي خوانندگان عزيز است.

### FormatCurrency(expression [,numDigitsAfterDecimal ] [,includeLeadingDigit ][,useParensForNegativeNumbers ][,groupDigits ])

اين تابع، يك مقدار عددي را به شكل يك مقدار ارزي (Currency) قالب بندي شده همراه با نمادي كه در سربرگ Regional Options از قسمت Regional and Language Options از Control Panel مشخص شده است، برگشت مي دهد.
به عنوان مثال در ويندوزهاي فارسي، نماد مقادير ارزي با كلمه "ريال" و در ويندوزهاي انگليسي با كاراكتر "$"، مشخص مي شود.
براي تغيير نحوه نمايش اين مقادير، به قسمت گفته شده در بالا مراجعه كنيد.
در اين تابع، تمامي آرگومان ها به جز اولين آرگومان، اختياري هستند.
آرگومان expression، مقداري عددي است كه قصد برگشت آن با فرمت Currency را داريم.
آرگومان numDigitsAfterDecimal، مقداري عددي است كه مشخص مي كند چند عدد در سمت راست بعد از اعشار بايد قرار بگيرند. مقدار پيش فرض، 1- است يعني تنظيمات Regional and Language Options را به كار مي برد.
آرگومان includeLeadingDigit، يك مقدار ثابت Tristate است. (ثابت Tristate، مقداري است كه مي تواند True، False يا UseDefault باشد) و مشخص مي كند كه آيا اگر مقداري اعشاري كمتر از 1 بود، در سمت چپ اعشار، عدد صفر قرار گيرد يا خير.
به عنوان مثال عدد هفت دهم كه مي تواند به شكل "0.7" و يا ".7" نمايش داده شود.

آرگومان useParensForNegativeNumbers، نيز يك مقدار ثابت Tristate است كه مشخص مي كند آيا اعداد مثبت را با دو پرانتز در دو طرف آنها برگشت دهد يا خير.
آرگومان groupDigits هم يك مقدار ثابت Tristate است و مشخص مي كند آيا اعداد بيش از 5 رقمي با كاراكتري كه در قسمت Regional and Language Options مشخص شده از يكديگر جدا شوند يا خير. به عنوان مثال در صورت True بودن اين آرگومان، عدد 234,456 به طريقي كه مشاهده مي كنيد با كاراكتر (,) از يكديگر جدا شده و اعداد از سمت راست، سه تا سه تا جدا مي شوند. در صورت False بودن اين آرگومان، اعداد پشت سر هم و بدون كاراكتر جداكننده ظاهر مي شوند.

### FormatDateTime(date [,namedFormat ])

اين تابع، يك مقدار از نوع تاريخ (Date) يا زمان (Time) را قالب بندي مي كند.
آرگومان date، يك مقدار از نوع تاريخ يا زمان است كه بايد قالب بندي شود.
آرگومان اختياري namedFormat، مشخص مي كند كه چه نوع قالب بندي بايد براي تاريخ يا زمان به كار رود.
اين آرگومان مي تواند يكي از مقادير زير را بپذيرد:

**GeneralDate**: اگر در آرگومان date، يك مقدار از نوع تاريخ آمده باشد، تاريخ به شكل كوتاه مثلا "14/02/2004"نمايش داده خواهد شد و اگر مقدار از نوع زمان باشد، زمان به شكل كامل مثلا "عصر 06:47:55" نمايش داده مي شود.
توجه: اگر هر دو مقدار تاريخ و زمان به كار روند، پشت سر هم نمايش داده خواهند شد.
**LongDate**: اين مقدار ثابت، يك مقدار از نوع تاريخ را به صورت طولاني نمايش و برگشت مي دهد. مثلا: "شنبه, ١٤ آبان, ١٣٨٢ "
**ShortDate**: اين مقدار ثابت، يك مقدار از نوع تاريخ را به صورت كوتاه نمايش و برگشت مي دهد. مثلا: "٢٠٠٤/٠٢/١٤"
**LongTime**: اين مقدار ثابت، يك مقدار از نوع زمان را به صورت طولاني نمايش مي دهد. مثلا: "٠٦:٥٨:٣٤ عصر"
**ShortTime**: اين مقدار ثابت، يك مقدار از نوع زمان را به صورت كوتاه نمايش مي دهد. مثلا: "١٩:٢٣"
**نكته مهم**: تمامي مقادير ثابت بالا، بر اساس تنظيماتي كه در قسمت Regional and Language Options از Control Panel صورت گرفته است عمل مي كنند.

### FormatNumber(expression [,numDigitsAfterDecimal ] [,includeLeadingDigit ][,useParensForNegativeNumbers ][,groupDigits ])

اين تابع، يك مقدار عددي را با قالب بندي ديگري به صورت عددي برگشت مي دهد.
تمامي آرگومان هاي اين تابع در توضيحات تابع ()FormatCurrency بيان شدند.

### FormatPercent(expression [,numDigitsAfterDecimal ] [,includeLeadingDigit ][,useParensForNegativeNumbers ][,groupDigits ])

اين تابع، يك عدد را بر حسب درصد برگشت مي دهد يعني عدد را در ١٠٠ ضرب كرده و با نماد درصد (%) و با قالبي كه مشخص شده است برگشت و نمايش مي دهد.
تمامي آرگومان هاي اين تابع در توضيحات تابع ()FormatCurrency بيان شدند.

### LSet(string,len) , RSet(string,len)

اين توابع، به تعدادي كه مشخص مي شود، در سمت چپ (Lset) يا راست (Rset) يك عبارت، فضاي خالي قرار مي دهند.
آرگومان string كه مي تواند از هر نوع مقداري باشد، رشته اي را مشخص مي كند كه بايد در سمت چپ يا راست آن، فضاي خالي اضافه شود.

آرگومان len نيز مقداري عددي است كه تعداد فضاي خالي را كه بايد ايجاد شود مشخص مي كند.
مثال هاي زير را در نظر بگيريد:

```
Console.WriteLine("[" & RSet("Behrouz",20) & "]")
Console.WriteLine("[" & LSet("Rad",20) & "]")
Console.WriteLine(RSet(34.56,10))
Console.WriteLine(RSet(4356.99,10))
Console.WriteLine(RSet(4.01,10))
```

خروجي پس از اجرا به شكل زير خواهد شد:

```
[            Behrouz]
[Rad                 ]
     34.56
     4356.99
     4.01
```

## Val(string) , Str(number)

تابع Val()، يك مقدار رشته اي را به عنوان ورودي دريافت كرده و آن را به صورت مقداري عددي برگشت مي دهد.
تابع Str() دقيقا عكس تابع Val() رفتار كرده و يك مقدار عددي را دريافت كرده و آن را به صورت مقداري عددي برگشت مي دهد.
مكانيزم كار تابع Val() به اين شكل است كه از ابتداي رشته ورودي شروع به خواندن آن مي كند و زماني كه به كاراكتري **غير عددي** رسيد جستجو متوقف و نتيجه برگشت داده مي شود.
مثال زير را در نظر بگيريد:

```
Dim a As String ="18:6.05 "
Console.WriteLine(Val(a))
```

نتيجه ي خروجي، عدد 18 خواهد بود زيرا كاراكتر ":" كاراكتري غير عددي است.
مثالي براي تابع Str():

```
Dim a As Integer = 18
Console.WriteLine(Str(a))
```

نتيجه خروجي، رشته "18" خواهد بود.

## توابع رياضي (Math):

در VB.NET تمامي توابع رياضي VB.6.0 به عنوان متدهايي در كلاس Math قرار داده شده اند. پس در اين بخش آنها را با نام متد صدا مي زنيم.
براي استفاده از متدهاي رياضي در VB.NET بايد كلاس Math را به برنامه ي خود اضافه يا قبل از استفاده از اين متدها، كلمه Math را قبل از نام متد ذكر كنيد مثل: (Math.Abs، (Math.Cos

### Abs(expression)

اين متد، قدر مطلق يك عبارت عددي را محاسبه مي كند.
آرگومان expression، عددي است كه بايد قدر مطلق آن محاسبه گردد. اگر expression بزرگتر يا مساوي صفر باشد، نتيجه ي متد همان عبارت expression خواهد بود وگرنه عبارت expression در عدد 1– ضرب شده و نتيجه برگشت داده مي شود.
مثال زير را در نظر بگيريد:

```
Abs(1.01)
Abs(-1.01)
```

نتيجه هر دو اين دستورات، عدد 1.01 خواهد بود.

### Atan(expression)

اين متد، آرك تانژانت يك مقدار را محاسبه و برگشت مي دهد.
آرگومان expression مقداري است كه آرك تانژانت آن بايد محاسبه گردد. نتيجه اين متد، مقداري بين 1.57079– تا 1.57079 است.
مثال:

```
Atan(2.3)=1.16
```

### Cos(expression)

اين متد، كسينوس يك زاويه را بر حسب راديان محاسبه مي كند.
آرگومان expression، عبارتي است كه بايد كسينوس آن محاسبه شود. حاصل متد ()Cos، مقداري بين 1– تا 1 است.
مثال:

```
Cos(0)=1
Cos(2.3)=-0.67
```

### Exp(expression)

اين متد، تواني از e (عدد نپر) را برمي گرداند.
نكته: تقريبا e=2.7182
آرگومان expression، مقداري عددي(تواني) است كه بايد عدد نپر آن محاسبه گردد و مي تواند يك عدد صحيح، اعشاري، مثبت يا منفي باشد.
مثال:

```
Exp(2)=7.31
```

### Int(expression) , Fix(expression)

هر دو اين متدها، مقداري عددي را دريافت كرده و عددي از نوع Integer (صحيح) را برگشت مي دهند.
آرگومان expression عبارتي عددي است كه به عنوان ورودي اين متدها داده مي شود.

اگر مقدار ورودي، صحيح و مثبت يا منفي باشد، همان مقدار expression برگشت داده مي شود.
اگر مقدار ورودي، اعشاري و مثبت باشد، قسمت اعشاري حذف شده و قسمت صحيح برگشت داده مي شود.
اگر مقدار ورودي، اعشاري و منفي باشد دو حالت دارد:
در حالت اول اگر از متد Int() استفاده شود، اولين عدد صحيح منفي كوچكتر يا مساوي عبارت ورودي برگشت داده مي شود.
در دومين حالت اگر از متد Fix() استفاده شود، اولين عدد صحيح منفي بزرگتر يا مساوي عبارت ورودي برگشت داده مي شود.
مثال هاي زير را در نظر بگيريد:

```
Int(12)=12
Fix(6)=6
Int(14.23)=14
Int(500/Int(2.5)))=250
Fix(15.5)=15
Int(-2.3)=-3
Fix(-2.3)=-2
```

پيشنهاد مي كنم اگر مي خواهيد كه از دست قسمت اعشاري يك عدد خلاص شويد و آن را اصطلاحا گرد كنيد، به شكل زير عمل كنيد:

```
Int(value +0.5)
```

Value، مقداري عددي است كه قصد گرد كردن آن را داريد.
اين دستور، فرمان بسيار كارآمدي براي گردكردن هر نوع مقداري عددي است با اين وجود، متد Round() كه در قسمت بعد توضيح داده شده است به شيوه اي آسانتر، بهتر و كارآمدتر اين كار را انجام مي دهد.

## Round(expression [,numdecimalplaces ])

اين متد، يك عدد اعشاري را گرد مي كند.
آرگومان expression، مقداري است كه بايد گرد شود.
آرگومان اختياري numdecimalplaces مشخص مي كند كه اين مقدار تا چند رقم بعد از اعشار بايد گرد شود. اگر از ذكر اين آرگومان خودداري شود، يك مقدار از نوع صحيح (Integer) برگشت داده مي شود.
مثال هاي زير را در نظر بگيريد:

```
Round(3.49)=3
Round(3.51)=4
Round(3.49,1)=3.5
Round(3.51,1)=3.5
```

## Log(expression)

اين متد، لگاريتم طبيعي (يا لگاريتم پايه e كه تقريبا e=2.7182 است) مقداري را برمي گرداند.
آرگومان expression، مقداري عددي است كه لگاريتم طبيعي آن بايد محاسبه شود و حتما بايد مقداري مثبت باشد.
عبارت Log(Exp(N)) و Exp(Log(N)) هر دو عدد N را برمي گردانند.
لگاريتم طبيعي، لگاريتم پايه عدد e (نپر) است.
مقدار دقيق عدد نپر را مي توانيد با دستور Exp(1) به دست بياوريد.
براي محاسبه لگاريتم ها در پايه هاي ديگر، عدد لگاريتم طبيعي را بر لگاريتم پايه تقسيم كنيد.
به عنوان مثال، دستور زير لگاريتم يك عدد در پايه ي 10 را محاسبه مي كند.

```
Log10 =Log(number)/Log(10)
```

## Hex(expression) , Oct(expression)

اين متدها عددي را به عنوان ورودي دريافت كرده و آن را به مبناي ٨ (Oct) يا مبناي ١٦ (Hex) تبديل مي كنند و به صورت يك مقدار رشته اي (String) برگشت مي دهند.
به عنوان مثال عبارت Hex(47)، مقدار "2F" و عبارت Oct(47) مقدار "57" را برگشت مي دهند.
براي تعيين كردن اعداد در مبناي ١٦، ابتداي آنها كاراكتر "&H" را قرار دهيد.
كاراكتر و نماد معادل براي اعداد در مبناي ٨، كاراكتر "&0" مي باشد.
فرض كنيد مقادير زير تعريف شده باشند:

```
Dvalue =199 : Ovalue =&O77
```

حال دستور Oct(Dvalue)، مقدار رشته اي "307" و دستور Hex(Dvalue) مقدار رشته اي "3F" را برگشت مي دهد.
مي توانيد از كاراكترهاي معادل اين متدها نيز استفاده كنيد. مثال زير را در نظر بگيريد:

```
MsgBox ("The number 3F in decimal is " & &H3F)
```

مقداري كه نشان داده مي شود، عدد 63 خواهد بود.

## Pow(value1,value2)

اين متد، دو عدد را به عنوان ورودي دريافت كرده و عدد اول را به توان عدد دوم مي رساند.
دستور زير را در نظر بگيريد:

```
Console.WriteLine(Math.Pow(2,3))
```

در خروجي، عدد ٨ نمايش داده خواهد شود. ٢ به توان ٣ برابر با ٨ است.

## Sin(expression)

اين متد، سينوس زاويه اي را بر حسب راديان محاسبه و برگشت مي دهد.
آرگومان expression، عبارتي است كه سينوس آن بايد محاسبه شود.
نتيجه متد Sin()، مقداري بين 1 تا 1– است.
مثال:

```
Sin(2.3) = 0.75
```

## Sqrt(expression)

اين متد، مربع (عدد به توان ٣) يك مقدار عددي را محاسبه و برگشت مي دهد.
آرگومان expression، عددي است كه بايد مربع آن محاسبه شود.

## Tan(expression)

اين متد، تانژانت زاويه اي را بر حسب راديان محاسبه مي كند.
مثال:

```
Tan(2.3) = -1.12
```

## توابعي كه با تاريخ و زمان سر و كار دارند (Date and Time):

توابع كاربردي و كارآمدي جهت كار با تاريخ و زمان در VB.NET وجود دارد كه ممكن است در برخورد اول با اين توابع، از تنوع آنها تعجب كنيد!
نكته مهم: در VB، مقاديري كه از نوع تاريخ هستند بايد حتما بين دو علامت (#) قرار گيرند.

### Now()

اين تابع، تاريخ و زمان جاري سيستم را برمي گرداند.
مثال:

```
MsgBox(Now())
```

به عنوان مثال، خروجي دستور فوق مي تواند "02/10/2004 09:23:10 PM" باشد.
توجه داشته باشيد كه تاريخ و زمان با كاراكتر خالي (Blank) از هم جدا شده اند.
نكته مهم: براي استخراج تنها تاريخ يا زمان برگشت داده شده از اين تابع، از خواص Date و TimeOfDay اين تابع به شكل زير استفاده كنيد.

```
Console.WriteLine(Now.Date.ToString)
Console.WriteLine(Now.TimeOfDay.ToString)
```

### Day(date)

اين تابع، يك مقدار از نوع تاريخ را به عنوان ورودي مي پذيرد و روز آن را برمي گرداند.
به عنوان مثال اگر مقدار ورودي، تاريخ 12/15/2004 باشد، مقدار خروجي عدد 15 يعني روز تاريخ خواهد بود.
مثال زير روز جاري سيستم را برگشت مي دهد.

```
Day(Now())
```

### Weekday(date [,firstdayofweek ])

اين تابع، عددي بين 1 تا 7 را برمي گرداند كه مشخص كننده روز هفته است.
مثلا، 1 براي Sunday، 2 براي Monday و به همين ترتيب.
آرگومان date يك مقدار از نوع تاريخ است.
آرگومان اختياري firstdayofweek، اولين روز هفته را مشخص مي كند و مي تواند يكي از مقادير زير را بپذيرد:
مقدار اين آرگومان به طور پيش فرض Sunday يعني يكشنبه است.

مطابق با تنظيمات سيستم >>> 0 يا System
يكشنبه (مقدار پيش فرض) >>> 1 يا Sunday
دوشنبه >>> 2 يا Monday
سه شنبه >>> 3 يا Tuesday
چهارشنبه >>> 4 يا Wednesday
پنج شنبه >>> 5 يا Thursday
جمعه >>> 6 يا Friday
شنبه >>> 7 يا Saturday

### WeekdayName(weekday [,abbreviate [,firstdayofweek ]])

اين تابع همانند تابع Weekday() عمل مي كند با اين تفاوت كه اين تابع، نام روز معادل يك عدد كه بين 1 تا 7 است را برمي گرداند.
آرگومان weekday يك مقدار عددي صحيح است و عددي بين 1 تا 7 را مي پذيرد.
آرگومان اختياري abbreviate، يك مقدار منطقي (Boolean) است كه مشخص مي كند آيا نام روزي كه برگشت داده مي شود به صورت كوتاه و اختصاري باشد يا خير. به طور پيش فرض اين مقدار False است يعني نام به صورت كامل برگشت داده مي شود.

آرگومان اختياري firstdayofweek نيز اولين روز هفته را مشخص مي كند و مي تواند يكي از مقاديري را كه آرگومان firstdayofweek از تابع Weekday مي پذيرد، بپذيرد.
مقدار اين آرگومان به طور پيش فرض Sunday يعني يكشنبه است.

### Month(date)

اين تابع از مقدار تاريخ، ماه آن را برمي گرداند كه عددي بين 1 تا 12 است.
آرگومان date، مقداري از نوع تاريخ است كه ماه آن بايد برگشت داده شود.
به عنوان مثال براي به دست آوردن عدد متناظر ماه جاري سيستم از دستور زير استفاده كنيد:

```
Month(Date())
```

### MonthName(month [,abbreviate ])

اين تابع نيز همانند تابع Month() عمل مي كند با اين تفاوت كه به جاي عدد ماه، نام ماه را برگشت مي دهد.
آرگومان month، مقداري عددي و بين 1 تا 12 است كه بايد ماه متناظر آن برگشت داده شود.
آرگومان اختياري abbreviate، يك مقدار منطقي (Boolean) است كه مشخص مي كند آيا نام ماهي كه برگشت داده مي شود به صورت كوتاه و اختصاري باشد يا خير. به طور پيش فرض اين مقدار False است يعني نام ماه به صورت كامل برگشت داده مي شود.

### Year(date)

اين تابع از مقدار تاريخ، سال آن را برمي گرداند.
مقداري كه اين تابع برمي گرداند، عددي بين 0 تا 9999 مي باشد.
آرگومان date، مقداري از نوع تاريخ است كه سال آن بايد برگشت داده شود.
به عنوان مثال براي به دست آوردن عدد متناظر سال جاري سيستم از دستور زير استفاده كنيد:

```
Year(Now())
```

### Hour(time)

اين تابع از مقدار زمان، بخش ساعت آن را برمي گرداند.
مقداري كه اين تابع برمي گرداند بين 0 تا 24 است.
آرگومان Time، مقداري از نوع زمان (Time) است كه ساعت آن بايد برگشت داده شود.
به عنوان مثال دستور زير ساعت جاري سيستم را نشان مي دهد:

```
Console.WriteLine(Hour(Now()))
```

### Minute(time)

اين تابع از مقدار زمان، دقيقه آن را برمي گرداند.
مقداري كه اين تابع برمي گرداند بين 0 تا 59 است.
آرگومان time، مقداري از نوع زمان است كه دقيقه آن بايد برگشت داده شود.
به عنوان مثال دستور زير دقيقه جاري سيستم را نشان مي دهد:

```
Console.WriteLine(Minute(Now()))
```

### Second(time)

اين تابع از مقدار زمان، ثانيه آن را برمي گرداند.
مقداري كه اين تابع برمي گرداند بين 0 تا 59 است.
آرگومان time، مقداري از نوع زمان است كه ثانيه آن بايد برگشت داده شود.
به عنوان مثال دستور زير ثانيه جاري سيستم را نشان مي دهد:

```
Console.WriteLine(Second(Now()))
```

## DateSerial(year,month,day)

اين تابع، سه عدد صحيح را به عنوان ورودي دريافت كرده و آن را به تاريخ تبديل مي كند.
آرگومان هاي month،year و day به ترتيب سال، ماه و روز هستند.
دستور زير را در نظر بگيريد:

```
Console.WriteLine(DateSerial(2002,10,1))
```

خروجي اين دستور، رشته ي "01/10/2002" خواهد بود.
همچنين از اين تابع مي توانيد جهت برخي محاسبات بر روي مقدار تاريخ استفاده كنيد.
به عنوان مثال فرض كنيد مي خواهيد بدانيد كه نودمين روز سال در چه تاريخي از سال قرار خواهد گرفت. مي توانيد از دستور زير جهت آگاهي از اين موضوع استفاده كنيد:

```
Console.WriteLine(DateSerial(1996,1,90))
```

خروجي اين دستور، عبارت "03/30/1996" خواهد بود.
به عنوان مثالي ديگر مي خواهيم بدانيم كه ۱۰۰۰ روز بعد از تاريخ فعلي سيستم در چه تاريخي قرار خواهد گرفت. پس به شكل زير عمل مي كنيم:

```
Console.WriteLine(DateSerial(Year(Now.Date),Month(Now.Date),Weekday(Now.Date)+1000))
```

شما همچنين قادر به اضافه كردن يا كم كردن (تفريق كردن) يك عدد متناظر ماه ها به آرگومان month و همچنين يك عدد متناظر سال به آرگومان year هستيد.
توجه داشته باشيد كه تعداد روزهاي هر ماه در اين تابع ۳۰ روز است.

## DateValue(date)

اين تابع، يك عبارت رشته اي را به عنوان ورودي دريافت كرده و آن را به يك مقدار از نوع تاريخ تبديل مي كند.
به عنوان مثال اگر اين تابع را به شكل زير به كار ببريم:

```
Console.WriteLine(DateValue("December 25, 2002"))
```

عبارتي كه در خروجي چاپ مي شود، "12/25/2002" خواهد بود.
يكي از كاربردهاي سودمند و مفيد اين تابع در هنگامي است كه با تابع DateDiff() به كار رود.
تابع DateDiff() در ادامه توضيح داده مي شود.
فرض كنيد مي خواهيم تعداد روزهاي بين دو تاريخ را محاسبه كنيم:
دستور زير اين كار را براي ما انجام مي دهد:

```
Console.WriteLine((DateDiff(DateInterval.Day,DateValue("12/25/1993"), _
DateValue("12/25/1996 "))))
```

عددي كه در خروجي ظاهر خواهد شد، 1096 يعني تعداد روزهاي بين اين دو تاريخ خواهد بود.

## TimeSerial(hours,minutes,seconds)

اين تابع، سه عدد صحيح را به زمان مشخصي تبديل مي كند.
آرگومان هاي Hour، Minute و Second به ترتيب اعداد صحيحي هستند كه ساعت، دقيقه و ثانيه را كه بايد به زمان تبديل شوند، تعيين مي كند.
مثال زير را در نظر بگيريد:

```
TimeSerial(4,10,55)
```

اين تابع، مقدار "4:10:55 AM" را برگشت مي دهد.
اين تابع معمولا براي محاسبه ي زمان به كار مي رود.

به عنوان مثال، فراخواني تابع TimeSerial() به شكل زير، ۲ ساعت و ۱۵ دقيقه و ۳۲ ثانيه قبل از ساعت 4:13:40 PM را برمي گرداند.

```
TimeSerial(16 -2,13 -15,40 -32)
```

زمان برگشتي، 1:58:08 PM خواهد بود.

## TimeValue(time)

اين تابع، يك مقدار رشته اي را دريافت كرده و آن را به مقدار زماني تبديل مي كند.
همانند تابع DateValue()، اين تابع نيز مي تواند جهت انجام عملياتي نظير جمع يا تفريق زمان ها به كار رود.

## DateAdd(interval,number,date)

اين تابع همان طور كه از نام آن نيز پيداست، مقداري را به تاريخي مشخص اضافه كرده و تاريخ جديد را برگشت مي دهد.
آرگومان interval، يك واحد زماني همانند روز، ساعت، هفته و ... است كه فاصله دوره اي را مشخص مي كند كه تاريخ جديد بايد بر حسب آن محاسبه و برگشت داده شود و مي تواند يكي از مقادير زير را بپذيرد:

**Year**: سال
**Quarter**: يك چهارم سال يا ۳ ماه يا يك فصل
**Month**: ماه
**DayOfYear**: روز از سال
**Day**: روز
**WeekDay**: روز از هفته
**WeekOfYear**: هفته از سال
**Hour**: ساعت
**Minute**: دقيقه
**Second**: ثانيه

آرگومان number، تعداد هر دوره ي زماني را مشخص مي كند. مثلا ممكن است بخواهيم بدانيم كه ۳ ماه بعد از تاريخ فعلي سيستم چه تاريخي مي شود پس اين پارامتر را برابر با ۳ قرار مي دهيم.
نكته: اگر اين آرگومان عددي مثبت باشد، تاريخي كه برگشت داده خواهد شد مربوط به آينده است و اگر اين آرگومان، منفي باشد، تاريخي كه نشان داده مي شود مربوط به گذشته مي باشد.
آرگومان date نيز تاريخي را مشخص مي كند كه عمليات بايد بر روي آن انجام شود.

به عنوان مثال مي خواهيم بدانيم كه ۱ ماه بعد از ۳۱ دسامبر سال ۲۰۰۲، چه تاريخي مي شود. پس به شكل زير عمل مي كنيم:

```
Console.WriteLine(DateAdd(DateInterval.Month,1,#12/31/2002#))
```

نتيجه اين دستور، عبارت "1/31/2003 12:00:00 AM" خواهد بود.
اين تابع تا حد زيادي شبيه به تابع DateSerial() است كه توضيح داده شد با اين تفاوت كه تعداد روزهاي هر ماه در اين تابع بنا به نام ماه فرق مي كند مثلا ممكن است يك ماه ۳۰ يا ۳۱ روز باشد اما تعداد روزهاي ماه در تابع DateSerial()، ثابت و ۳۰ روز مي باشد.
مثال زير را در نظر بگيريد:

```
day1 =#1/31/2002#
Console.WriteLine(DateSerial(year(day1),month(day1)+1,day(day1)))
```

نتيجه عبارت، "3/2/02" خواهد بود كه يك تاريخ در ماه مارس است نه فوريه.

## DateDiff(interval,date1,date2 [,firstdayofweek [,firstweekofyear ]])

اين تابع، شكل پيشرفته تر تابع DateAdd() است و اختلاف بين دو تاريخ را برمي گرداند.
آرگومان interval يك مقدار ثابت است و فواصل بين دوره اي را مشخص مي كند كه شما براي محاسبه اختلاف بين دو تاريخ به كار مي برد.
مقاديري را كه اين آرگومان مي تواند بپذيرد همان مقاديري هستند كه آرگومان interval تابع DateAdd() مي تواند بپذيرد.
آرگومان هاي date1 و date2 تاريخ هايي هستند كه اختلاف آنها بايد برگشت داده شود.
نكته: هميشه مقدار date2 از date1 كم مي شود.
آرگومان هاي اختياري firstdayofweek و firstweekofyear نيز به ترتيب، اولين روز هفته و اولين هفته سال را تعيين مي كنند.
مقاديري را كه اين آرگومان ها مي توانند بپذيرند، همان مقادير آرگومان firstdayofweek از تابع Format() است كه پيشتر توضيح داده شد. (به توضيحات تابع format() مراجعه كنيد)
مثلا شما مي توانيد از اين تابع براي اطلاع از تعداد روزها، هفته ها، ثانيه ها و ... بين دو تاريخ استفاده كنيد.
مثال زير تعداد روزهاي بين يك تاريخ مشخص شده و تاريخ فعلي سيستم را نمايش مي دهد:

```
Dim century As Date
century =#1/1/2000#
MsgBox(DateDiff(DateInterval.Day,century,Now()))
```

نكته مهم: توابع DateDiff و TimeSerial در پروژه هاي وب و مخصوصا فوروم هايي كه به وسيله ي زبان ASP.NET ايجاد مي شوند، كاربرد فراواني دارند.
يك نكته ي بسيار مهم: نكته اي كه متاسفانه در اكثر پروژه هاي وب در ايران رعايت نمي شود اين است كه ساعت رسمي ايران در روز اول فروردين يك ساعت به جلو كشيده مي شود و در اين حالت، ساعت كشور ما از گرينويچ، ٤:٣٠ ساعت جلوتر است، در حالي كه در روز ٣١ شهريور يك ساعت به عقب كشيده مي شود و به حالت اوليه خود باز مي گردد يعني +3.5 GMT

## DatePart(interval,date [,firstdayofweek [,firstweekofyear ]])

اين تابع، بخشي از يك تاريخ را برمي گرداند.
آرگومان interval بخشي است كه بايد از تاريخ استخراج شود و همان مقاديري را مي پذيرد كه مي توان براي قالب دهي به تاريخ و زمان در تابع Format() به كار برد.
آرگومان date نيز تاريخي است كه قسمتي از آن بايد استخراج شود.
آرگومان هاي اختياري firstdayofweek و firstweekofyear نيز در تابع DateDiff() توضيح داده شدند.
به عنوان مثال فرض مي كنيم تاريخ جاري سيستم، ٢٣ اكتبر سال ٢٠٠١ ميلادي است.
دستورات زير نتايج مختلفي را به ما نشان مي دهند كه نتيجه به صورت پررنگ تر نمايش داده شده است:

```
Dim Day1 As DateTime
day1 =Now()
Console.WriteLine(DatePart("yyyy ",day1))
2001
Console.WriteLine(DatePart("q ",day1))
3
Console.WriteLine(DatePart("m ",day1))
10
Console.WriteLine(DatePart("d ",day1))
23
Console.WriteLine(DatePart("w ",day1))
3
Console.WriteLine(DatePart("ww ",day1))
43
```

```
Console.WriteLine(DatePart(“h ”,day1))
15
Console.WriteLine(DatePart(“n ”,day1))
3
Console.WriteLine(DatePart(“s ”,day1))
30
```

## توابعي كه جهت انجام عمليات ورودي/خروجي (I/O) بر روي فايل ها مورد استفاده قرار مي گيرند (File I/O):

يكي از جنبه هاي مهم برنامه نويسي در هر زبان، قابليت دسترسي و مديريت فايل ها مي باشد.
در ويژوال بيسيك، ٣ نوع فايل وجود دارد:
١) فايل هاي ترتيبي.
٢) فايل هاي با دسترسي تصادفي.
٣) فايل هاي باينري.

فايل هاي ترتيبي اغلب فايل هاي متني (Text) هستند. به عبارت ديگر شما مي توانيد اينگونه فايل ها را با هر ويرايشگر متني همانند Notepad باز كنيد.
در اينگونه فايل ها هر كاراكتر ذخيره شده در فايل، يك بايت را اشغال مي كند.
تمامي اطلاعات ذخيره شده در فايل هاي متني به صورت رشته اي ذخيره خواهند شد.
مثلا مقادير عددي به صورت Integer يا Double ذخيره نمي شوند بلكه به صورت String عمل ذخيره سازي آنها انجام مي شود.
عمل خواندن فايل هاي ترتيبي، به ترتيب يعني از ابتدا تا انتهاي فايل و خط به خط انجام مي شود بنابراين شما نمي توانيد در يك زمان در يك فايل ترتيبي هم اطلاعات را بخوانيد و هم اطلاعات را بنويسيد.
اگر قصد خواندن و نوشتن همزمان در و از يك فايل ترتيبي را داريد، بايد از دو فايل ترتيبي استفاده كنيد. يكي براي خواندن اطلاعات و يكي براي نوشتن اطلاعات درون فايل ترتيبي.
اگر در برنامه اي كه مي نويسيد به طور مكرر با فايل ها سرو كار داريد به گونه اي كه بايد مرتبا محتويات يك فايل خوانده شود و پس از انجام تغييرات مجددا ذخيره شود، بايد از فايل هاي با دسترسي تصادفي استفاده كنيد.
اين فايل ها در مقايسه با فايل هاي ترتيبي از سرعت بيشتري برخوردارند.
فايل هاي تصادفي نيز همانند فايل هاي ترتيبي براي مقاديري از نوع رشته اي به ازاي هر كاراكتر، يك بايت را اختصاص مي دهند اما بر خلاف فايل هاي ترتيبي، در اين نوع فايل ها مقادير عددي با نوع خاص خود ذخيره مي شوند مثلا Integer يا Double.
براي نمايش فايل هاي تصادفي در محيط Dos مي توانيد با نوشتن دستور TYPE در خط فرمان، مقادير رشته اي ذخيره شده در اين فايل ها را مشاهده كنيد اما قادر به ديدن اعداد ذخيره شده نيستيد.
فايل هاي تصادفي براي ذخيره كردن داده هايي كه در بخش هايي با طول مشخص دسته بندي شده اند به كار مي روند.
اين بخش ها را اصطلاحا **ركورد** مي ناميم.
فايل هاي تصادفي به شما اين امكان را مي دهند كه به هر ركورد دلخواه در فايل به سرعت دسترسي داشته باشيد.
از آنجايي كه تمامي ركوردها در فايل تصادفي داراي طول ثابتي هستند، اضافه كردن ركورد جديد در فايل با اختصاص دادن يك شماره شاخص (index) به اين ركورد بسيار آسان است.
فايل هاي با دسترسي تصادفي برخلاف فايل هاي ترتيبي مي توانند در يك زمان، هم فايل را بخوانند و هم در آن بنويسند.
در فايل هاي تصادفي اطلاعات را مي توان در هر جاي فايل نوشت بدون آنكه همجواري ركوردها از بين برود يا لطمه اي بخورد.
**فايل هاي باينري**: اينگونه فايل ها شبيه فايل هاي ترتيبي هستند.
محتويات اين فايل ها مي تواند هر چيزي باشد.
كاراكتر، عدد يا حتي تصاوير مي توانند در فايل هاي باينري ذخيره شوند.
در اين نوع فايل ها، داده ها به صورت باينري ذخيره مي شوند.
مثلا عدد ٤٢٦ عددي از نوع صحيح است و چون اعداد صحيح دو بايت از حافظه را اشغال مي كنند پس ميزان فضاي اختصاص يافته به اين عدد در اين نوع فايل، ٢ بايت است.

انجام عمليات بر روي فايل ها بدون در نظر گرفتن نوع آنها، به طور كلي شامل ٣ مرحله مي باشد:

**بازكردن فايل**: در اين مرحله، سيستم عامل ميزان معيني از حافظه را براي نگهداري داده هاي فايل كنار مي گذارد. اگر فايل وجود نداشته باشد، ابتدا ساخته شده و سپس باز مي شود.
براي بازكردن يا در صورت لزوم، ايجاد يك فايل از تابع ()FileOpen استفاده كنيد.
**پردازش فايل**: يك فايل مي تواند براي خواندن، نوشتن و يا خواندن-نوشتن همزمان داده هايش باز شود.
داده هايي كه خوانده مي شوند، پردازش شده و سپس در همان جاي قبلي يا در فايل ديگري ذخيره مي شوند.
**بستن فايل**: هنگامي كه كار با فايل به اتمام رسيد بايد آن را بست. هنگامي كه فايل بسته شد، سيستم عامل فضاي اختصاص داده شده به فايل را به حافظه اصلي باز مي گرداند.
براي بستن يك فايل باز از تابع ()FileClose استفاده كنيد.

## FreeFile(file_number)

هر فايلي كه باز مي شود بايد شماره اي به آن اختصاص يابد.
اگر برنامه ي شما در طول دوره كاري خود تعداد زيادي فايل را باز يا بسته مي كند، ممكن نيست كه شما هميشه شماره فايل يا فايل هايي را كه هم اكنون باز هستند بدانيد.
در VB، تابع ()FreeFile شماره فايلي بعدي را كه قرار است باز شود تعيين مي كند.
اين تابع هميشه با تابع ()FileOpen جهت بازكردن يك فايل به كار مي رود.
نكته: تابع ()FreeFile را نمي توان به طور مستقيم در تابع ()OpenFile به كار برد.
دستور زير را در نظر بگيريد:

```
FileOpen(FreeFile(), "C:\MyFile.dat",OpenMode.Input)
```

اين دستور از نظر نحوي درست كار مي كند اما شماره فايل را در اختيارتان قرار نمي دهد تا بتوانيد اعمال خواندن يا نوشتن را بر روي فايل انجام دهيد.

دستور زير را در نظر بگيريد:

```
fNum =FreeFile()
FileOpen(fNum, fileName)
```

بعد از اجراي اين دو دستور، هر فرماني كه با فايل سروكار داشته باشد مي تواند با آوردن نام fNum در آرگومان مربوطه ي خود، به اين فايل دسترسي داشته باشد.
همان طور كه گفته شد اين تابع، شماره فايل بعدي را تعيين مي كند مگر اينكه اين شماره به فايلي اختصاص داده شده باشد. در اينصورت در صورت فراخواني مجدد اين تابع، عددي مشابه فراخواني قبلي برگشت داده مي شود.
به عنوان مثال دستورات زير كار نمي كنند:

```
fNum1 =FreeFile()
fNum2 =FreeFile()
FileOpen(fNum1, file1)
FileOpen(fNum2, file2)
```

دستور دوم اشتباه است چون هر وقت كه تابع ()FreeFile براي تعيين شماره ي يك فايل جديد فراخواني مي شود بايد مورد استفاده قرار بگيرد و نمي توان قبل از به كارگيري آن دوباره از همين تابع جهت ايجاد يك شماره جديد استفاده كرد.
پس دستورات فوق را به شكل زير اصلاح مي كنيم:

```
fNum1 =FreeFile()
FileOpen(fNum1, file1)
fNum2 =FreeFile()
FileOpen(fNum, file2)
```

## FileOpen(number,path,mode [,access ][,share ][,recordLen ])

براي استفاده از يك فايل، بايد ابتدا آن را باز يا در صورت نياز آن را ايجاد كنيد.
تابع FileOpen() با دريافت مقاديري اين كار را براي شما انجام مي دهد.
اين تابع، جانشين دستور Open در VB.6.0 شده است. توجه داشته باشيد كه دستور Open در VB.NET قابل استفاده نيست.
آرگومان path مسير و نام فايلي است كه بايد باز شود و آرگومان number شماره اي است كه شما به اين فايل اختصاص مي دهيد (معمولا با تابع FreeFile()).
آرگومان mode، نوع عملياتي را كه قرار است بر روي فايل انجام شود تعيين مي كند و مي تواند يكي از مقادير زير باشد:

| مقادير پارامتر Mode در تابع FileOpen() | |
|---|---|
| **مقدار** | **عملكرد** |
| **Input** | اين مقدار براي بازكردن فايل ترتيبي كه وجود دارد به كار مي رود. اگر فايل را با اين مقدار باز كنيم فقط مي توانيم اطلاعات را از آن بخوانيم. چنانچه فايلي كه قبلا بر روي ديسك وجود نداشته و بخواهيم با مد Input باز كنيم، خطايي اتفاق مي افتد. |
| **Output** | اين مقدار، فايل ترتيبي را طوري باز مي كند كه بتوان اطلاعاتي را در آن نوشت. اگر فايلي كه با مد Output باز مي شود وجود نداشته باشد، آن را ايجاد مي كند ولي اگر وجود داشته باشد، اطلاعات قبلي آن را از بين مي برد و آن را براي دريافت اطلاعات جديد آماده مي كند. به طور كلي اين مقدار فايل را در حالت خروجي باز مي كند يعني فقط مي توان اطلاعات را به فايل اضافه كرد. |
| **Append** | فايل ترتيبي كه قبلا وجود داشته باشد و بخواهيم اطلاعاتي زا به آن اضافه كنيم، بايد از اين مقدار استفاده كنيم. چنانچه فايل قبلا وجود نداشته باشد، آن را ايجاد مي كند. |
| **Binary** | اين مقدار، فايل را به صورت باينري باز مي كند. در حالت باينري مي توان به هر يك از بايت هاي فايل دسترسي داشت و مي توان يك بايت را از فايل خواند يا در فايل نوشت. |
| **Random** | اين مقدار، يك فايل را به طور تصادفي باز مي كند. در اين حالت مي توان در فايل نوشت يا از فايل خواند. |

سه مقدار اول با فايل هاي ترتيبي، مقدار Random با فايل هاي با دسترسي تصادفي و مقدار Binary براي فايل هاي باينري به كار مي روند.
آرگومان اختياري access نوع دسترسي به فايل را تعيين مي كند. يعني فايل براي خواندن يا نوشتن يا هر دو عمل باز شود.
مقاديري كه اين آرگومان مي تواند بپذيرد در جدول زير آمده است:

| مقادير پارامتر access در تابع FileOpen() | |
|---|---|
| **مقدار** | **عملكرد** |
| **Read** | فايل را در حالت ورودي باز مي كند. يعني فقط مي توان فايل را خواند. |
| **Write** | فايل را در خالت خروجي باز مي كند. يعني فقط مي توان در فايل نوشت. |
| **ReadWrite** | فايل را در حالت ورودي-خروجي باز مي كند. يعني هم مي توان اطلاعات را از فايل خواند و هم در فايل نوشت. |

اين آرگومان بيشتر براي بالا بردن امنيت كار با فايل به كار مي رود و ضرورت چنداني در استفاده از آن وجود ندارد. مثلا فايل هاي ترتيبي مي توانند براي فقط خواندن يا فقط نوشتن باز شوند نه هر دو. پس استفاده از آرگومان ReadWrite اين آرگومان در هنگام باز كردن يك فايل ترتيبي بي اثر است.

ولي جهت بالا بردن ضريب امنيت استفاده از فايل، هنگامي كه فايل را در هر حالتي باز مي كنيد، با توجه به نوع فايل استفاده از اين آرگومان نيز ضرري ندارد.
آرگومان اختياري share، نحوه عكس العمل فايلي كه توسط برنامه شما باز شده است را به برنامه هاي ديگر ويندوز، در همان زمان كه شما فايل را باز كرده ايد مشخص مي كند.
به عبارت ديگر شما مي توانيد نحوه دسترسي برنامه هاي ديگر را به فايلي كه شما باز كرده و در حال استفاده از آن هستيد تعيين كنيد.
آرگومان share مي تواند يكي از مقادير زير را بپذيرد:

| مقادير پارامتر share در تابع FileOpen() | |
|---|---|
| **مقدار** | **عملكرد** |
| **Shared** | برنامه هاي ديگر مي توانند از فايل استفاده كنند. |
| **LockRead** | برنامه هاي ديگر فقط در صورتي مجاز به استفاده از فايل هستند كه آن را فقط براي خواندن باز كرده باشند. |
| **LockWrite** | برنامه هاي ديگر فقط در صورتي مجاز به استفاده از فايل هستند كه آن را فقط براي نوشتن باز كرده باشند. |
| **LockReadWrite** | برنامه هاي ديگر مجاز به استفاده از فايل نيستند |

آرگومان share يك پارامتر بسيار مهم براي برنامه هايي است كه تحت شبكه نوشته مي شوند. به عنوان مثال دو كاربر به طور همزمان قصد نوشتن در يك فايل را دارند.
و اما آخرين آرگومان يعني آرگومان اختياري recordlen كه فقط در فايل هاي با دسترسي تصادفي كاربرد دارد.
اگر فايلي كه قصد بازكردن آن را داريد از اين نوع باشد بايد طول هر ركورد فايل را كه بر حسب بايت است به وسيله اين آرگومان مشخص كنيد.
هنگامي كه در ويژوال بيسيك فايلي با دسترسي تصادفي ايجاد مي كنيد، هيچ اطلاعاتي راجع به طول ركورد يا ساختار آن در فايل ذخيره نمي شود.
اين مورد را قبل از بازكردن اين نوع فايل ها بايد بدانيد.
طول ركورد، مجموع بايت هاي تمامي فيلدهاي آن فايل است كه مي توانيد با دستور Len(record) آن را محاسبه كنيد.
آرگومان record در اين دستور نام ساختاري است كه براي بازكردن فايل ترتيبي به كار برده ايد.
دستور زير فايل C:\samples\vb\cust.dat را كه يك فايل ترتيبي است با اختصاص دادن عددي كه با استفاده از تابع FreeFile() به دست آورده، در حالت ورودي-خروجي باز مي كند:

```
Dim Fnum As Integer =FreeFile()
FileOpen(Fnum, “c:\samples \vb \cust.dat ”, OpenMode.Output, OpenAccess.ReadWrite)
```

## FileClose(file_number)

پس از اينكه كار با فايل تمام شد، بايد آن را بست.
تابع FileClose() فايلي را كه شماره ي آن به وسيله ي آرگومان file_number مشخص شده مي بندد.
دستور زير، فايلي را كه شماره ي آن در عبارت fNum1 وجود دارد مي بندد.

```
FileClose(fNum1)
```

## Reset()

اين تابع در حقيقت كار شما را در بستن فايل ها بسيار آسانتر مي كند.
اين تابع تمامي فايل هايي را كه با استفاده از تابع FileOpen() در برنامه شما باز شده اند مي بندد.

## EOF(file_number) , LOF(file_number)

اين دو تابع معمولا در توابعي كه با فايل ها سر و كار دارند بسيار به كار مي روند.
تابع EOF() براي تست شرط پايان فايل به كار مي رود. اين تابع فايلي را كه شماره آن به وسيله آرگومان file_number مشخص شده چك مي كند و چنانچه اشاره گر فايل به انتهاي آن رسيده باشد، اين تابع مقدار True وگرنه مقدار False را برمي گرداند.
تابع LOF()، اندازه يك فايل باز را كه شماره آن به وسيله آرگومان file_number مشخص شده است بر حسب بايت برمي گرداند.

شبه كد زير با استفاده از تابع EOF() و يك حلقه تكرار، پايان يك فايل را چك مي كند:

{رفتن به اولين ركورد فايل}
While Not EOF(fNum)
{پردازش ركورد جاري}
{رفتن به ركورد بعد}
End While

با استفاده از تابع LOF() قادر به محاسبه ي تعداد ركورد هاي يك فايل با دسترسي تصادفي هستيد. دستور زير اين كار را براي ما انجام مي دهد:

```
Rec_Length =LOF(file_number) / Len(record)
```

## Print (file_number,output_list) , PrintLine(file_number,output_list)

براي نوشتن در يك فايل ترتيبي كه به صورت Output يا Append باز شده است از تابع Print() استفاده مي كنيم.
آرگومان file_number، شماره فايلي است كه قبلا با يكي از دو حالت بالا باز شده است و آرگومان output_list نيز يك آرايه ي پارامتري است، يعني مقادير يا متغيرهاي مقداردهي شده اي كه بايد در فايل نوشته شوند.
**نكته:** منظور از آرايه ي پارامتري در VB، آرگومانی است كه مي تواند مقادير نامحدودي بپذيرد. اين مقادير با استفاده از كاراكتر (,) از هم جدا مي شوند.
دستور زير را در نظر بگيريد:

```
Print(fNum, var1, var2, “My VB World”,333.333)
```

اين دستور، ٤ مقدار شامل مقدار ٢ متغير، يك مقدار رشته اي و يك مقدار عددي را در فايلي كه شماره آن در عبارت fNum وجود دارد ذخيره مي كند.
تابع Print() داده ها را در خطوط مجزا ذخيره نمي كند يعني اينطور نيست كه با هر بار فراخواني اين تابع، داده ها در خط بعدي فايل ذخيره شوند بلكه پشت سر هم و بدون فاصله قرار خواهند گرفت.
به عنوان مثال، دستور زير فقط يك خط متني در فايلي كه شماره آن در fNum وجود دارد مي نويسد:

```
Print(fNum, "This is the first half of the line ")
Print(fNum, "and this is the second half of the same line.")
```

تابع PrintLine() همان كاري را انجام مي دهد كه تابع Print() انجام مي دهد با اين تفاوت كه با هر بار فراخواني، داده را در خطوط مجزا قرار مي دهد همانند دستور WriteLn در زبان پاسكال كه اطلاعات را در خطوط جدا، ذخيره مي كند يا نمايش مي دهد.
براي خواندن داده هايي كه با تابع Print() در فايل ذخيره شده اند از توابع LineInput() و Input() استفاده مي كنيم.
تابع Print() به طور پيش فرض بين فيلدهاي مختلف در يك فايل فاصله اي قرار نمي دهد.
براي قالب دهي به فيلدهايي كه در هر خط فايل متني ذخيره مي شوند مي توانيد از كلمه كليدي (Tab) استفاده كنيد تا VB در هنگام ذخيره داده ها در فايل، بين فيلد هاي مختلف فاصله قرار دهد.
همچنين خود مي توانيد براي اين فاصله مقداري تعيين كنيد. مثلا: Tab(12)
مثال زير يك فايل متني را ساخته و مقداري داده را در آن ذخيره مي كند:

```
On Error Resume Next
Kill("c:\test.txt ")
Dim fNum As Integer =FreeFile()
FileOpen(fNum, "c:\test.txt", OpenMode.Output)
PrintLine(fNum, "Behrouz", TAB(12), "Rad", TAB(25), "Manager", TAB(45), 33)
PrintLine(fNum, "sina", TAB(12), "ahmadi", TAB(25), "Programmer", TAB(45), 28)
PrintLine(fNum, "arvin", TAB(12), "sanaei", TAB(25), "Engineer", TAB(45), 41)
PrintLine(fNum, "pedram", TAB(12), "amiri", TAB(25), "Administrator", TAB(45), 25)
PrintLine(fNum, "**************************")
PrintLine(fNum, "Behrouz", TAB, "Rad", TAB, "Manager", TAB, 3)
PrintLine(fNum, "sina", TAB, "ahmadi", TAB, "Programmer", TAB, 28)
PrintLine(fNum, "arvin", TAB, "sanaei", TAB, "Engineer" , TAB, 41)
PrintLine(fNum, "pedram", TAB, "amiri", TAB, "Administrator", TAB, 25)
FileClose(fNum)
```

خروجي دستورات بالا به شكل زير خواهد شد:

```
Behrouz     Rad         Manager          33
sina        ahmadi      Programmer       28
arvin       sanaei      Engineer         41
pedram      amiri       Administrator    25
**************************
Behrouz          Rad          Manager       33
sina             ahmadi       Programmer    28
arvin            sanaei       Engineer      41
pedram           amiri        Administrator 25
```

## Input(file_number,var)

بديهي است كه پس از نوشتن در فايل ترتيبي، بايد بتوان اطلاعات را از آن خواند و پردازش كرد. براي خواندن اطلاعات از فايل ترتيبي، بايد آن را به صورت Input باز كرد.
تابع Input() داده هاي يك فايل ترتيبي را خوانده و آنها را در متغيري كه به عنوان دومين آرگومان اين تابع وجود دارد يعني var مي ريزد.
آرگومان file_number شماره فايلي است كه به صورت Input باز شده است و آرگومان var، يك آرايه پارامتري است كه شامل اسامي فيلدها يا ليستي از متغيرها است كه اطلاعات خوانده شده در آنها قرار مي گيرد.
در اين دستور، اگر هنگام خواندن فايل تعداد فيلدهايي كه شما نام آنها را در آرگومان var مي آوريد از تعداد فيلدهاي فايل كمتر باشد، فيلدهاي بعدي در فراخواني بعدي فرمان Input() نمايش داده مي شوند. ولي چنانچه تعداد فيلدهايي كه نام آنها در آرگومان var مي نويسيد، بيشتر از تعداد فيلدهاي يك ركورد فايل باشد، به تعداد لازم از ركوردهاي بعدي خوانده مي شوند.
به عنوان مثال، دستورات زير يك مقدار عددي و يك مقدار از نوع تاريخ را از فايلي كه باز شده است مي خوانند:

```
Dim numVal As Long,DateVal As Date
Input(1,numVal)
Input(1,DateVal)
```

## LineInput(file_number)

پيشنهاد مي كنم براي خواندن فايل ترتيبي از اين تابع استفاده كنيد.
آرگومان file_number شماره فايل باز شده است و اين تابع يك سطر از فايل ترتيبي را برگشت مي دهد.
اين تابع، در اولين بار فراخواني، اولين خط فايل ترتيبي را برگشت مي دهد و در فراخواني بعدي دومين خط و به همين منوال.
دستورات زير سطرهاي اول و دوم يك فايل ترتيبي را در متغيرهاي Line1 و Line2 قرار مي دهند.

```
Line1 =LineInput(fNum)
Line2 =LineInput(fNum)
```

براي ذخيره سازي متن هاي ساده در يك فايل، يك فايل ترتيبي درست كنيد و داده هاي خود را در آن ذخيره كرده و در موقع لزوم براي خواندن خط به خط داده هاي ذخيره شده با استفاده از تابع LineInput() يا براي خواندن كل فايل از تابع FileGet() استفاده كنيد.

### FilePut(file_number,value [,record_number ]) , FileGet(file_number,value [,record_number ])

اين دو تابع براي خواندن يا نوشتن ركوردها به يا از فايل با دسترسي تصادفي به كار مي روند.
تابع FilePut() براي نوشتن ركورد و تابع FileGet() براي خواندن ركورد استفاده مي شوند.
براي كار با اين توابع نياز به دانستن شماره ركوردي داريد كه مي خواهيد آن را بخوانيد يا در فايل بنويسيد.
آرگومان file_number، شماره فايل است.
آرگومان record_number، شماره ركوردي است كه بايد خوانده يا نوشته شود.
آرگومان record_number يك پارامتر اختياري است و چنانچه از ذكر آن خودداري شود، ركورد در مكان فعلي اشاره گر فايل نوشته مي شود. (در تابع FilePut())
آرگومان value، متغير ركوردي است كه در فايل نوشته شده است يا مي شود.
پس از اينكه يك ركورد به فايل نوشته يا از فايل خوانده شد، ركورد بعد از آن به عنوان ركورد جاري در نظر گرفته مي شود.
به عنوان مثال اگر تابع FilePut()، ۱۰ بار پشت سر هم و بدون تعيين كردن شماره ركورد فراخواني شود، ۱۰ ركورد اول فايل با دسترسي تصادفي، ايجاد يا بازنويسي مي شود.

از آنجايي كه فايل با ساختار دسترسي تصادفي، انعطاف پذيرترين ساختار و بيشترين كاربرد را دارد، با ذكر چند مثال با نحوه عملكرد اين ساختار بيشتر آشنا مي شويم:
در اينجا من قصد ايجاد يك فايل نوع دسترسي تصادفي را جهت نگهداري مشخصات محصول يك شركت دارم. هر مشخصه از اين محصول در يك متغير به نام ProductRecord قرار خواهد گرفت. ساختار ProductRecord را به شكل زير تعريف مي كنم:
توجه: در VB.NET از كلمه كليدي Structure جهت تعريف يك ساختار سفارشي استفاده مي كنيم. اين كلمه جايگزين كلمه كليدي Type در VB.6.0 شده است.

```
Structure ProductRecord
   ProductID As String
   Description As String
   Price As Decimal
End Type
```

اطلاعات كالا قبل از ذخيره شدن در فايل، در هر يك از اين ۳ متغير قرار خواهند گرفت.
حالا براي دسترسي به فيلدهاي اين ساختار بايد متغيري از نوع اين ساختار يعني ProductRecord تعريف كنيم:

```
Dim PRec As ProductRecord
```

حال مي توانيم مقادير خود را به فيلدهاي متغير Prec نسبت دهيم:

```
PRec.ProductID = “TV00180-A ”
PRec.Description = “SONY  TV ”
PRec.Price = 799.99
```

اكنون همه چيز براي ذخيره كردن مقادير وارد شده در فايل ما آماده است اما قبل از انجام اين كار بايد فايل را ايجاد كنيم:

```
fNum =FreeFile()
FileOpen(fNum, “C:\products.dat ”,OpenMode.Random)
```

توجه داشته باشيد كه من در تابع FileOpen() از ذكر متغير آخر كه طول ركورد را تعيين مي كند خودداري كردم. از آنجائيكه فيلدهاي ما داراي مقادير رشته اي هستند و طول آنها از قبل

مشخص نيست (متغير است)، اجازه مي دهيم كه تابع به طور خودكار ركوردها را اداره و طول آنها را مديريت كند. اصول كار به اين شكل است كه هنگام ذخيره متغير رشته اي، طول آن نيز همراه با مقدار آن در فايل ذخيره مي شود پس شما نيازي به دانستن طول هر ركورد نداريد.
و حالا مقادير نسبت داده شده را با دستور زير در فايل ذخيره مي كنيم:

```
FilePut(fNum,Prec)
```

توجه كنيد كه شما مي توانيد از ذكر آخرين آرگومان تابع FilePut() كه شماره ي ركوردي كه بايد نوشته شود را مشخص مي كند، خودداري كنيد.
پس از ذخيره تمامي مقادير در فايل بايد فايل را بست. پس با دستور زير اين كار را مي كنيم:

```
FileClose(fNum)
```

براي خواندن ركوردها، همان تابع FileOpen() كه براي ذخيره سازي ركوردها استفاده كرديم را به كار مي بريم:

```
fNum =FreeFile()
FileOpen(fNum, “C:\products.dat ”, OpenMode.Random)
```

پس از باز كردن فايل جهت خواند محتويان آن، مي توانيم با يك ساختار تكرار (loop) اين كار را انجام دهيم.
مثال زير نشان مي دهد كه چگونه ركوردهايي با طول متفاوت و متغير را با استفاده از تابع FilePut() در يك فايل با دسترسي تصادفي ذخيره و سپس آنها را با استفاده از تابع FileGet() بخوانيم.
ابتدا كد زير را كه ساختار ركورد ما را تعيين مي كند در قسمت تعاريف فرم (declarations) بنويسيد.

```
Structure ProductRecord
   Dim ProductID As String
   Dim Description As String
   Dim Price As Decimal
End Structure
```

حال كدهاي زير را در رويداد Click() يك دكمه ي فرم قرار دهيد.

```
Dim PRec As ProductRecord
PRec.ProductID = “TV00180-A ”
PRec.Description = “SONY TV ”
PRec.Price = 799.99

fNum =FreeFile()
FileOpen(fNum,“C:\products.dat ”, OpenMode.Random)
FilePut(fNum,PRec)

PRec =New ProductRecord()
PRec.ProductID = “TV-Flat”
PRec.Description =“This is a Flat TV ”
PRec.Price = 699.99
FilePut(fNum,PRec)

PRec =New ProductRecord()
PRec.ProductID = “TV-FST”
PRec.Description = “And this is the real FST TV ”
Prec.Price = 399.99
FilePut(fNum,Prec)

FileClose(fNum)

fNum = FreeFile()
FileOpen(fNum, “C:\products.dat ”, OpenMode.Random)
```

```
Prec =New ProductRecord()
FileGet(fNum,Prec,2)
FileClose(fNum)

Console.WriteLine(Prec.ProductID)
Console.WriteLine(Prec.Description)
Console.WriteLine(Prec.Price)
```

همان طور كه مشاهده مي كنيد، فيلدهاي ProductID و Description محصولات مختلف، كه از نوع رشته اي هستند داراي طول هاي مختلف هستند.
سه قسمت اول كد بالا، ۳ ركورد مختلف را در فايل تصادفي ذخيره كرده و سپس فايل را مي بندد.
قسمت آخر كد نيز، دومين ركورد فايل را خوانده و مقادير آن را در خروجي نمايش مي دهد.
نكته: در VB.6.0، هر ركورد در فايل تصادفي بايد داراي طول ثابت مي بود اما در VB.NET، مي توانيد ركوردهايي ايجاد و در فايل تصادفي قرار دهيد كه داراي طول متغير باشند.

## Write(file_number,output_list) , WriteLine(file_number,output_list)

تابع Write() جهت ذخيره داده ها در يك فايل ترتيبي به كار مي رود.
آرگومان file_number شماره فايلي است كه داده ها در آن ذخيره خواهند شد.
آرگومان output_list نيز كه يك آرايه ي پارامتري است، ليستي از داده هايي است كه مي خواهيم در فايل ترتيبي نوشته شوند و مي توانند متغيرهاي مقداردهي شده يا خود مقدار مورد نظر باشند. داده ها در آرگومان output_list، با كاراكتر (,) از يكديگر جدا مي شوند.
داده هايي كه با تابع Write() در فايل نوشته مي شوند معمولا با تابع Input() خوانده مي شوند.
دستورات زير، يك مقدار عددي و يك مقدار از نوع تاريخ را در يك فايل ترتيبي مي نويسند:

```
NumVal = 3300.004
DateVal = #04/09/1999#
Write(1, NumVal, DateVal)
```

تابع WriteLine() نيز همان كار تابع Write() را انجام مي دهد با اين تفاوت كه با هر بار فراخواني اين تابع، داده ها در سطر جديدي از فايل نوشته مي شوند.
**نكته مهم**: تفاوت توابع Print() با Write() و WriteLine() در اين است كه توابع Write() يا WriteLine() باعث مي شود فيلدها در هنگام نوشته شدن در فايل ترتيبي با كاما (,) از يكديگر جدا شوند ولي در تابع Print() چنانچه فيلدها با كاما از يكديگر جدا شوند، در هنگام ذخيره در فايل، بين فيلدها ١٤ فاصله قرار خواهد گرفت اما اگر فيلدها با كاراكتر (;) از يكديگر جدا شوند، فيلدها بدون فاصله از يكديگر در فايل قرار مي گيرند.

مثال زير شبيه مثالي است كه در توضيحات تابع Print() آمد اما اين بار با استفاده از تابع WriteLine() آن را مي نويسيم:

```
Dim fNum As Integer = FreeFile()
FileOpen(fNum, “C:\test.txt”, OpenMode.Output)
WriteLine(fNum, “Behrouz Rad”, 33, “Manager ”)
WriteLine(fNum, “Sina Ahmadi”, 24, “Programmer ”)
WriteLine(fNum, “Arvin Sanaei”, 37, “Engineer ”)
WriteLine(fNum, “Pedram Amiri”, 28, “Administrator ”)
```

خروجي دستورات فوق به شكل زير خواهد شد:

```
“Behrouz Rad”, 33, ”Manager”
“Sina Ahmadi”, 24, ”Programmer”
“Arvin Sanaei”, 37, ”Engineer”
“Pedram Amiri”, 28, ”Administrator”
```

## Seek(file_number [,position ]) , Loc(file_number)

تابع ()Loc، مكان جاري اشاره گر فايل را برگشت مي دهد.
تابع ()Seek نيز در صورتي كه بدون ذكر آرگومان اختياري position به كار رود، عملي مشابه تابع ()Loc را انجام مي دهد.
در فايل هاي با دسترسي تصادفي، مقدار برگشتي اين دو تابع، شماره آخرين ركوردي است كه از فايل خوانده شده يا در فايل نوشته شده.
در فايل هاي ترتيبي، اين مقدار، با تقسيم بايت خوانده شده جاري بر عدد ۱۲۸ به دست مي آيد.
در فايل هاي باينري نيز، شماره آخرين بايت خوانده شده از يا نوشته شده به فايل به عنوان مقدار برگشتي خواهد بود.
شما قادر به تعيين مكان شروع خواندن يا مكان نوشته شدن ركورد در فايل با مقداردهي به آرگومان position از تابع ()Seek هستيد.
مثلا براي آنكه اشاره گر ركورد را در يك فايل با دسترسي تصادفي به سومين ركورد ببريد يا به عبارت ديگر آن را به عنوان ركورد فعال منظور كنيد، مي توانيد از دستور زير استفاده كنيد:

```
Seek(fNum,3)
```

## Lock(file_number [,fromRecord ][,toRecord ]) , Unlock(file_number [,fromRecord ][,toRecord ])

تابع ()Lock اين امكان را براي شما فراهم مي كند كه يك فايل با دسترسي تصادفي يا تعدادي از ركوردهاي آن را قفل كنيد.
آرگومان file_number شماره فايل را مشخص مي كند.
توجه: آرگومان هاي اختياري fromRecord و toRecord را زماني به كار ببريد كه قصد قفل كردن تعدادي از ركوردهاي فايل را داريد نه همه ركوردها را.
با استفاده از آرگومان هاي fromRecord و toRecord قادر به تعريف محدوده اي جهت قفل كردن و به انحصار در آوردن ركوردها هستيد. آرگومان fromRecord تعيين مي كند كه قفل گذاري از كدام ركورد شروع شده و آرگومان toRecord نيز تعيين مي كند كه قفل گذاري تا كدام ركورد ادامه پيدا كند.
مثلا مي توانيد تعيين كنيد كه ركوردهاي ۵ تا ۱۰ در فايل، قفل شوند.

### نكات مهم در استفاده از تابع ()Lock:

۱)در صورتي كه از ذكر دو آرگومان اختياري اين تابع خودداري شود، تمام ركوردهاي فايل قفل خواهند شد.
۲) در صورتي كه فقط براي آرگومان fromRecord مقدار ذكر شود و آرگومان toRecord مقداردهي نشود، ركوردها از مقدار fromRecord تا به انتهاي فايل قفل مي شوند.
۳) ركوردهايي كه با اين تابع قفل مي شوند، ركوردهاي انحصاري هستند كه فقط توسط برنامه شما قابل استفاده بوده و برنامه هاي ديگر در حال اجرا مجاز به استفاده از آنها نيستند.
٤) اگر برنامه اي ديگر در جهت دسترسي به فايل يا ركوردهايي كه توسط شما قفل شده اند اقدام نمايد، خطايي در برنامه اتفاق مي افتد كه بايد توسط تعريف ساختار خطا در برنامه، آن را كنترل كرده و از پايان برنامه جلوگيري كنيد.

تابع ()Unlock نيز دقيقا بالعكس تابع Lock عمل كرده و فايل يا ركوردهاي قفل شده را آزاد مي كند.

### Width(fNum,length)

اين تابع نيز يك بسيار بسيار كارآمد است كه جهت تعيين بيشترين طول خط يعني تعداد كاراكترهاي قابل ذخيره سازي در هر خط كه مي تواند در يك **فايل ترتيبي** نوشته شود، به كار مي رود.
بيشترين طول خط به وسيله ي آرگومان length تعيين مي شود.
در صورتي كه كاراكترهاي شما از تعداد تعيين شده در آرگومان length بيشتر باشد، كاراكترهاي اضافي به طور اتوماتيك به خط بعد منتقل مي شوند.
ابن تابع را مي توانيد با توابع ()Print و ()Write (كه داده ها را در يك خط و پشت سر هم ذخيره مي كنند) جهت تعيين تعداد كاراكترهاي ذخيره شونده در هر خط، به كار بريد.
همچنين مي توانيد تابع ()Width را با توابع ()PrintLine و ()WriteLine نيز (كه داده ها را در خطوط جدا ذخيره مي كنند) به كار ببريد.

### FileAttr(file_number)

اين تابع، يك عدد از نوع Integer را برگشت مي دهد كه مشخص كننده ي حالت باز شدن يك فايل باز است.
مقاديري را كه اين تابع برگشت مي دهد، در جدول زير مشاهده مي كنيد:

| مقاديري كه تابع ()FileAttr برگشت مي دهد | |
|---|---|
| **مقدار برگشتي** | **مد باز شدن** |
| **1** | Input |
| **2** | Output |
| **4** | Random |
| **8** | Append |
| **32** | Binary |

## توابعي كه براي ايجاد اعداد تصادفي به كار مي روند (Random Numbers):

VB.NET دو تابعي كه در VB.6.0 براي درست كردن اعداد تصادفي به كار مي روند (Rnd و Randomize) را پشتيباني مي كند اما از آنجا كه در VB.NET، اكثر توابع، دسته بندي و در كلاس خاص خودشان قرار داده شده اند، اين دو تابع نيز در كلاس System.Random قرار داده شده اند.

### Rnd([seed ])

اين تابع، در صورتي كه بدون آرگومان اختياري خود به كار رود، يك عدد تصادفي بين صفر و يك را برمي گرداند.
آرگومان اختياري seed، يك عبارت عددي است كه عدد تصادفي بايد بر حسب آن توليد شود.
مثلا دستور Rnd()*10، يك عدد تصادفي بين صفر و ده را برمي گرداند.
نكته: هميشه بين تابع Rnd() و عددي كه مي خواهيد عدد تصادفي بر مبناي آن توليد شود، از كاراكتر (*) استفاده كنيد.
توجه: ترتيب توليد اعداد تصادفي در اين تابع هميشه به يك منوال است. مثلا اگر دستور Rnd()*10 را ۳ بار پشت سر هم اجرا كنيم، هميشه سه عدد (7.055475، 5.33424، 5.795186) برگشت داده خواهد شد.
براي جلوگيري از اين كار و تغيير ترتيب توليد اعداد تصادفي، از تابع Randomize() كه در قسمت بعدي توضيح داده شده است، استفاده كنيد.
مثلا:

```
Dim I as Integer
Randomize()
I = Rnd() * 10
```

اگر مقداري در آرگومان seed ذكر شود، عدد تصادفي كه ايجاد خواهد شد، بين صفر تا آن عددي است كه در آرگومان seed ذكر شده. گاهي اوقات نياز داريد تا بين محدوده اي دلخواه مثلا: 5 تا 10، عددي تصادفي ايجاد كنيد. بدين منظور مي توانيد به شكل زير عمل كنيد:

```
randomNumber =Int((upper – lower + 1) * Rnd() + lower)
```

در اين دستور، متغير upper، حد بالاي محدوده ي ما (مثلا 10) و متغير lower، حد پايين محدوده (مثلا 5) است.
نكته: در كد بالا، جمع كردن نتيجه قسمت اول با عدد 1، از توليد عدد تصادفي بين صفر و يك جلوگيري مي كند. پيشنهاد مي كنم هميشه در توليد اعداد تصادفي، مقدار seed را با عدد يك جمع كنيد.
توجه: تابع Next نيز كه در كلاس System.Random قرار دارد قادر به ايجاد يك عدد تصادفي در محدوده اي خاص است.

### Randomize [seed ]

اين تابع، هسته ي توليد اعداد تصادفي را تغيير مي دهد.
اين تابع هميشه با تابع Rnd() به كار مي رود و معمولا قبل از آن نوشته مي شود.
آرگومان اختياري seed، يك مقدار عددي است كه براي توليد هسته ي اعداد تصادفي به كار مي رود. بهتر است از ذكر اين مقدار خودداري كنيد.

## توابع گرافيكي (Graphics):

در اين قسمت، دو تابع در VB.NET كه براي تعريف رنگ ار آنها استفاده مي شود را توضيح مي دهم.
توجه: توابع LoadPicture() و SavePicture() كه در VB.6.0 وجود داشتند، در VB.NET وجود ندارند.

### QBColor(color)

اين تابع، براي توليد تعدادي رنگ از پيش تعريف شده به كار مي رود و مقداري از نوع Integer كه معرف كد رنگ ايجاد شده است را برمي گرداند.
آرگومان Color، كد رنگي است كه بايد توليد شود و عددي بين 0 تا 15 است.
مقاديري را كه اين آرگومان مي تواند بپذيرد، در جدول زير آمده است:

| مقادير پارامتر color در تابع QBColor() | |
|---|---|
| **مقدار** | **رنگ** |
| **0** | سياه |
| **1** | آبي |
| **2** | سبز |
| **3** | لاجوردي |
| **4** | قرمز |
| **5** | ارغواني |
| **6** | زرد |
| **7** | سفيد |
| **8** | خاكستري |
| **9** | آبي روشن |
| **10** | سبز روشن |
| **11** | لاجوردي روشن |
| **12** | قرمز روشن |
| **13** | ارغواني روشن |
| **14** | زرد روشن |
| **15** | سفيد روشن (با شدت بالا) |

كاربرد اين تابع بيشتر در سيستم هايي است كه كارت گرافيك آنها نصب نشده يا تنها قادر به پشتيباني از ١٦ رنگ مي باشند كه در سيستم هاي قديمي كاربرد بيشتري دارد.

### RGB(red,green,blue)

رنگ ها داراي ٤ مدل اصلي هستند.
مدل، روش تعريف يك رنگ را مشخص مي كند.
مدلي كه مانيتور و تلويزيون خانه ي ما جهت نمايش رنگ ها از آن استفاده مي كند مدل RGB نام دارد. در اين مدل، ٣ رنگ اصلي وجود دارد كه بقيه رنگ ها از اين ٣ رنگ مشتق مي شوند.
رنگ هاي اين مدل عبارتند از: قرمز (Red)، سبز (Green)، آبي (Blue)
نام اين مدل از حروف اول اين ٣ رنگ گرفته شده است.
دامنه ي هر كدام از اين ٣ رنگ، عددي بين 0 تا 255 است.
به وسيله اين تابع، قادر به توليد هر يك از رنگ هاي مدل RGB هستيم.
تعداد رنگ هاي موجود در مدل RGB، ١٦٧٧٧٢١٦ رنگ است كه از ضرب ٣ عدد (256*256*256) به دست مي آيد.
مقدار برگشتي اين تابع، عددي از نوع Long است كه كد رنگ توليد شده را نشان مي دهد.

در جدول زير، كد تعدادي از رنگ هاي متداول را در مدل RGB، ملاحظه مي كنيد:

| كد معادل تعدادي از رنگ هاي پركاربرد در مدل RGB | | | |
|---|---|---|---|
| رنگ | كد پارامتر red | كد پارامتر green | كد پارامتر blue |
| سياه | 0 | 0 | 0 |
| آبي | 0 | 0 | 255 |
| سبز | 0 | 255 | 0 |
| لاجوردي | 0 | 255 | 255 |
| قرمز | 255 | 0 | 0 |
| ارغواني | 255 | 0 | 255 |
| زرد | 255 | 255 | 0 |
| سفيد | 255 | 255 | 255 |

به عنوان مثال، دستور زير، يك رنگ قرمز خالص را ايجاد و به پس زمينه ي عنصر TextBox، نسبت مي دهد:

```
Text1.BackColor = RGB(255, 0, 0)
```

فرمول تبديل مقدار RGB به معادل Long:

```
r + g * 256 + b * 256 ^ 2
```

## توابعي كه با رجيستري سر و كار دارند (Registry):

در VB، تعدادي تابع براي خواندن و نوشتن مقادير مختلف در رجيستري ويندوز وجود دارد.

### SaveSetting(appname, section, key, setting)

اين تابع، جهت ذخيره سازي يك مقدار دلخواه در رجيستري يا به هنگام كردن آن مقدار از طريق نوشتن آن بر روي مقدار قبلي، به كار مي رود.
از اين تابع معمولا براي ذخيره تنظيمات برنامه، استفاده مي شود.
مقادير در يك مسير اصلي، يك شاخه، يك زيرشاخه و در نهايت به صورت يك مقدار رشته اي، در زيرشاخه ذخيره مي شوند.
نكته: مسير اصلي توسط VB تعيين مي شود و نيازي به وارد كردن آن نيست.
آرگومان appname، نام شاخه يا پوشه ي اصلي است كه تمام زيرپوشه ها و مقادير آنها در اين شاخه ذخيره مي شوند. مي توانيد هر نام دلخواهي به اين نام بدهيد اما براي راحتي كار بهتر است نام برنامه ي خود را در اين آرگومان ذكر كنيد.
آرگومان section نيز نام زير شاخه اي در شاخه appname است كه مقادير مربوط به هم در اين زيرشاخه ذخيره مي شوند. نام اين آرگومان نيز دلخواه است.
آرگومان key، نام مقدار و آرگومان setting نيز خود مقداري است كه بايد ذخيره شود.
اگر مقدار setting به هر دليلي ذخيره نشود، يك خطاي زمان اجرا در برنامه اتفاق مي افتد.
به عنوان مثال فرض كنيد كه مقدار طول و عرض فرم برنامه ي ما، توسط كاربر در زمان اجراي برنامه تغيير پيدا كند و بخواهيم دفعه ي بعد كه برنامه توسط كاربر اجرا مي شود، فرم با طول و عرض جديد (يعني همان انداز هاي تغيير پيدا كرده توسط كاربر) نمايش داده شود.
دستورات زير، مقدار طول و عرض تغيير يافته ي فرم را در رجيستري ذخيره مي كنند:

```
SaveSetting(“MyApp”, “Startup”, “Height”,Me.Height)
SaveSetting(“MyApp”, “Startup”, “Width”,Me.Width)
```

دقت كنيد كه بنده جهت راحتي كار، مقدار ۳ پارامتر اول اين دستورات را متناسب با مقداري كه قرار است ذخيره شود انتخاب كردم.
نكته: مناسب ترين مكان براي نوشتن اين كدها، زماني است كه برنامه شما پايان مي پذيرد و مناسبترين زمان جهت فراخواني آنها، زماني است كه برنامه آغاز مي شود. (با استفاده از تابع GetSetting كه در ادامه توضيح داده شده است)

### DeleteSetting(appname,section [,key ])

اين تابع دقيقا عكس تابع SaveSetting() عمل كرده و يك مقدار از زيرشاخه يا كل زيرشاخه را از رجيستري خذف مي كند.
آرگومان هاي اين تابع، همان آرگومان هاي تابع SaveSetting() هستند.
اگر از ذكر آرگومان اختياري key خودداري شود، تمامي مقادير موجود در زيرشاخه حذف خواهند شد.
به عنوان مثال مي خواهيم مقاديري را كه در مثال تابع SaveSetting() در رجيستري ذخيره كرديم، حذف كنيم. پس دستورات زير را مي نويسيم:

```
DeleteSetting(“MyApp”, “Startup”, “Height”)
DeleteSetting(“MyApp”, “Startup”, “Width”)
```

مي توانيد اين دستورات را در يك خط به شكل زير خلاصه كنيد:

```
DeleteSetting(“MyApp”, Startup”)
```

دقت كنيد اگر مقادير ديگري نيز در زيرشاخه ي Startup وجود داشته باشند، آنها نيز حذف مي شوند.

## GetSetting(appname,section,key [,default ])

وقتي كه مقادير در رجيستري نوشته شد طبعا بايد آنها را خواند.
اين تابع، مقدار ذخيره شده ي كاربر را برمي گرداند.
۳ آرگومان اول اين تابع همان پارامترهاي توابع SaveSetting() و DeleteSetting() هستند كه مسير ذخيره داده در رجيستري را مشخص مي كنند.
آرگومان اختياري default، مقداري پيش فرض است كه در صورت نبودن مقداري در مسير ذكرشده، توسط تابع برگشت داده مي شود.
به عنوان مثال مي خواهيم مقاديري را كه در مثال تابع SaveSetting() در رجيستري ذخيره كرديم، به دست آورده و طول و عرض جديد فرم را در هنگام اجراي برنامه تعيين كنيم:

```
Me.Height =GetSetting("MyApp", "Startup", "Height", 1000)
Me.Width =GetSetting("MyApp", "Startup", "Width", 1500)
```

در اين دستورات حتما بايد مقدار آرگومان default را ذكر كنيد چون در صورتي كه مقداري در رجيستري وجود نداشته باشد، فرم ممكن است با سايزي بسيار كوچك يا بسيار بزرگ ظاهر شود.

## GetAllSettings(appname,section)

اين تابع، ليستي از تمامي مقادير موجود در زيرشاخه تعيين شده را با قرار دادن آنها در يك آرايه ي دو بعدي برمي گرداند.
آرگومان هاي appname و section قبلا توضيح داده شدند.
خانه ي (0,0) آرايه، نام اولين مقدار زيرشاخه و خانه ي (0,1) آرايه، مقدار آن را در خود نگه مي دارد.
خانه ي (1,0) آرايه، نام دومين مقدار زيرشاخه و خانه ي (1,1) آرايه، مقدار آن را نگهداري مي كند و به همين ترتيب...
نكته: براي آگاهي از تعداد مقادير ذخيره شده در آرايه يا به عبارت بهتر تعداد مقادير موجود در زيرشاخه، از خاصيت Length آرايه استفاده كنيد.
فرض مي كنيم آرايه AllSettings به عنوان يك آرايه دو بعدي معرفي شده است.
دستور زير، تمامي مقادير موجود در زير شاخه ي Startup از شاخه MyApp را در آرايه AllSettings قرار مي دهد.

```
AllSettings = GetAllSettings("MyApp", "Startup")
```

حالا مي توانيم با دستورات زير، تمامي مقادير موجود در زيرشاخه را كه در آرايه ذخيره شده اند را ببينيم:

```
For i =0 To AllSettings.GetUpperBound(0)
    Console.WriteLine(AllSettings(i,0) & " = " & AllSettings(i,1))
Next
```

## توابع كاربردي (Application Collaboration):

### Shell(path_name [,style ][,wait ][,timeout ])

با استفاده از اين تابع، قادر به اجراي يك برنامه ديگر از درون برنامه ي خود هستيد.
در صورتي كه اين تابع كار خود را با موفقيت انجام دهد، هندل برنامه ي اجرا شده را كه توسط سيستم عامل تعيين مي شود برگشت مي دهد در غير اينصورت، مقدار صفر را برگشت مي دهد.
آرگومان path_name، مسير و نام فايلي است كه مي خواهيم اجرا شود. اگر فايل اجرايي نياز به آرگومان هايي همانند command هاي مخصوص خود داشت، مي توان نام آنها را نيز در ادامه مسير فايل ذكر كرد كه اين مورد معمولا ندرتا وجود دارد.
نكته مهم: اين تابع تنها فايل هاي اجرايي را اجرا مي كند اما مي توان با نسبت دادن نام و مسير فايلي ديگر در ادامه نام و مسير فايل اصلي، در صورتي كه اين فايل با برنامه مورد نظر ما قابل اجرا باشد، آن را اجرا كرد مثلا يك فايل با پسوند TXT را مي توان با دستور زير اجرا كرد:

```
NTXT=Shell("notepad.exe C:\MyFile.txt", AppWinStyle.NormalFocus)
```

اين دستور، فايل MyFile.txt را به وسيله برنامه Notepad باز مي كند.
آرگومان اختياري style، وضعيت فرم برنامه اي كه اجرا خواهد شد را معين مي كند و مي تواند يكي از مقادير زير را بپذيرد:

**مقادير پارامتر Style در تابع Shell()**

| مقدار | عملكرد |
|---|---|
| **Hide** | برنامه به صورت مخفي اجرا مي شود اما به عنوان برنامه فعال در نظر گرفته مي شود. |
| **NormalFocus** | برنامه به صورت معمول و با اندازه خودش اجرا مي شود و به عنوان برنامه فعال در نظر گرفته مي شود. |
| **MinimizedFocus** | برنامه به صورت مينيمايز اجرا خواهد شد و به عنوان برنامه فعال در نظر گرفته مي شود. |
| **MaximizedFocus** | برنامه به صورت تمام صفحه اجرا مي شود و به عنوان برنامه فعال در نظر گرفته مي شود. |
| **NormalNoFocus** | برنامه به صورت معمول و با اندازه خودش اجرا مي شود اما به عنوان برنامه فعال در نظر گرفته نمي شود و برنامه اي كه به عنوان برنامه فعال است، فعال باقي مي ماند. |
| **MinimizedNoFocus** | برنامه به صورت مينيمايز اجرا مي شود اما به عنوان برنامه فعال در نظر گرفته نمي شود و برنامه اي كه به عنوان برنامه فعال است، فعال باقي مي ماند. |

آرگومان اختياري wait تعيين مي كند آيا برنامه شما براي اجراي كامل برنامه اي كه فراخواني خواهد شد صبر كند يعني دستورات بعد از تابع Shell()، پس از فراخواني برنامه مورد نظر اجرا شوند يا بلافاصله.
به عبارت ساده تر، تعيين مي كند كه قبل از اجراي تمامي دستورات بعد از تابع Shell()، برنامه اجرا شود يا در همان زمان فراخواني.
اين پارامتر، مقدار True يا False را مي پذيرد كه به طور پيش فرض، False است يعني برنامه اي كه قرار است اجرا شود بايد صبر كند تا تمامي دستورات بعد از تابع shell() اجرا شوند.
آخرين آرگومان اختياري كه timeout نام دارد، تعيين مي كند پس از چند ثانيه برنامه اي كه فراخواني مي شود بايد اجرا شود.
اين مقدار، عددي بر حسب ميلي ثانيه است و هر ۱۰۰۰ ميلي ثانيه، ۱ ثانيه است.

به طور پيش فرض مقدار اين آرگومان، 1- است يعني بدون وقفه و بلافاصله پس از فراخواني اجرا مي شود.
به عنوان مثال، براي اجراي برنامه Notepad ويندوز از دستور زير استفاده كنيد:

```
NPAD = Shell("notepad.exe", AppWinStyle.NormalFocus)
```

توجه: براي اجراي برنامه هايي كه در پوشه اصلي ويندوز هستند، نيازي به ذكر مسير آنها نيست و تنها آوردن نام، كفايت مي كند.
نكته مهم: تابع SendKeys() در VB.NET وجود ندارد.

## AppActivate(title [,wait ])

اين تابع (كه مقدار برگشتي ندارد) به شما اجازه مي دهد برنامه اي را كه قبلا توسط تابع Shell() فراخواني و اجرا كرده ايد، به عنوان برنامه فعال (پنجره فعال) سيستم عامل معرفي كنيد.
در آرگومان title، كه مقداري از نوع رشته اي است بايد نام نوار عنوان برنامه اي كه قصد فعال شدن آن را داريد ذكر كنيد. مثلا: "Untitled – Notepad"
در اين آرگومان همچنين با ذكر كردن مقداري كه تابع Shell() در متغير فراخواني كننده خود قرار مي دهد، قادر به فعال كردن برنامه هستيد. مثلا در آخرين مثال توضيحات تابع Shell()، مي توانيد نام متغير NPAD را در آرگومان title اين تابع ذكر كنيد.
آرگومان اختياري wait تعيين مي كند كه آيا برنامه شما بايد به عنوان برنامه فعال باشد تا بتواند برنامه ديگري را با اين تابع فعال كند يا خير.
اين پارامتر، مقدار True يا False را مي پذيرد. مقدار پيش فرض، False است يعني حتي در صورت فعال نبودن برنامه شما نيز، برنامه درخواستي به سرعت فعال مي شود.
در صورتي كه True باشد، برنامه شما بايد صبر كند تا فعال شود يا اصطلاحا فوكاس به آن انتقال پيدا كند تا بتواند برنامه ديگري را فعال كند.
توجه: اين تابع به مينيمايز بودن يا اندازه ي پنجره ها كاري ندارد و مثلا اگر برنامه اي مينيمايز باشد، آن را در همان حالت مينيمايز به عنوان برنامه فعال در نظر مي گيرد.
نكته مهم: هميشه سعي كنيد ابتدا برنامه مورد نظر را با تابع Shell() اجرا و سپس متغيري كه تابع Shell() را در آن قرار داده ايد در آرگومان title اين تابع قرار داده و اقدام به فعال كردن آن كنيد. مثلا فرض كنيد كه برنامه Notepad را ۲ بار اجرا كرده ايد و عنوان نوار هر دو پنجره Notepad، يكي است. در اين حالت اگر نام نوار عنوان را در تابع AppActivate() ذكر كنيد، آن يكي كه آخر اجرا شده است به عنوان پنجره فعال در نظر گرفته مي شود كه ممكن است دلخواه شما نباشد.

## دستورات اختياري (Option Statements):

دستورات Option يا اختياري، سهولت كار در كدنويسي را براي شما فراهم مي كنند.
اين دستورات در ابتداي كد فرم يا ماژول نوشته مي شوند و در تمام آن فرم يا ماژول قابل دسترسي هستند. اين دستورات را در زير مشاهده مي كنيد:

## Option Compare

اين دستور تعيين مي كند كه VB چگونه عمليات مقايسه اي رشته ها را انجام دهد.
مقايسه ي رشته اي مي تواند به طور معمول يا با در نظر گرفتن بزرگ يا كوچك بودن حروف انجام شود.
مقايسه رشته اي مي تواند به يكي از دو شكل زير صورت پذيرد:
**Option Compare Binary:** در اين نوع مقايسه كه به صورت باينري انجام مي شود، كاراكترهاي بزرگ بر كاراكترهاي كوچك ارجعيت دارند و كاراكترهاي Symbol يا نماد در انتها قرار مي گيرند. مثلا به شكل زير:
A < B < C... < Z < a < b < c... < z < A' < A^ < a' < a^

**Option Compare Text:** در اين نوع مقايسه، عمل مقايسه با توجه به بزرگ و كوچك بودن حروف انجام مي شود. اين نوع مقايسه، مقايسه پيش فرض رشته ها است. مثلا:
A=a < A'=a' < B=b

## Option Explicit

همان طور كه مي دانيد، در VB مي توان يك متغير را بدون تعريف كردن آن به كار برد.
مي توان با استفاده از اين دستور به كامپايلر ويژوال بيسيك گفت كه از تعريف نكردن متغيرها در طول فرم يا ماژولي كه اي دستور قرار مي گيرد، جلوگيري كند.
در اين حالت، در زمان اجراي برنامه كامپايلر VB تمامي متغيرها را از لحاظ تعريف شدن چك مي كند و در صورتي كه هر يك ار آنها تعريف نشده باشند، به شما اخطار مي دهد.
مي توانيد با فعال كردن اين گزينه در سربرگ Build از قسمت Project Property Pages در VB.NET، به طور خودكار در تمامي فرم ها و ماژول هاي برنامه اين دستور را فعال كنيد.
توجه: در صورتي كه در VB.NET، نوع متغير تعريف نشود به طور پيش فرض از نوع Object در نظر گرفته مي شود. (در VB.6.0 از نوع Variant در نظر گرفته مي شد)

## Option Strict

همان طور كه اشاره شد در صورتي كه نوع متغيري در VB.NET ذكر نشود، به طور پيش فرض از نوع Object در نظر گرفته مي شود و در هنگام مقداردهي به آن متغير، نوع آن به صورت خودكار توسط VB تعيين مي شود.
دستورات زير را در نظر بگيريد:

```
Dim MyVar
MyVar="Hello World"
```

در دستورات فوق ابتدا يك متغير بدون نوع تعريف مي شود كه به طور پيش فرض از نوع Object در نظر گرفته مي شود و سپس در هنگام مقداردهي به آن، VB به طور خودكار نوع مقدار نسبت داده شده را تشخيص و آن را تبديل به آن نوع مي كند كه در مثال فوق از نوع String در نظر مي گيرد.
چنانچه مايل هستيد كه عمل تبديل خوكار نوع متغير تعريف نشده توسط VB انجام نشود و همان مقدار Object در نظر گرفته شود، دستور Option Strict را در اولين خط كد فرم بنويسيد.

## توابع متفرقه (Miscellaneous):

توابعي هستند كه نمي توان آنها را در هيچ گروهي دسته بندي كرد.
اين توابع در اين قسمت مورد بحث قرار مي گيرند.

### Choose(index,choice1 [,choice2,...])

اين تابع، مقداري را از يك سري آرگومان، انتخاب و برگشت مي دهد.
آرگومان index، يك مقدار عددي بين ١ تا تعداد آرگومان هاي Choice است كه نمايانگر شماره آرگوماني است كه مي خواهيم برگشت داده شود.
آرگومان هاي choice1، choice2 و...، ليستي از آرگومان هايي هستند كه مي خواهيم مقداري را از آنها انتخاب و برگشت دهيم.
به عنوان مثال، اگر مقدار index، ١ باشد مقدار برگشتي، مقدار آرگومان choice1 و اگر index، ٢ باشد، مقدار برگشتي تابع، مقدار آرگومان choice2 و به همين ترتيب خواهد بود.
مثلا يكي از كاربردهاي اين تابع براي برگشت معادل حرفي اعداد تك رقمي است.
به عنوان مثال، تابع زير كه IntToString() نام دارد، نام معادل عددي كه به آن داده مي شود را برمي گرداند:

```
Function IntToString(int As Integer)As String
IntToString =Choose (int + 1, “zero”, “one”, “two”, “three” , _
                    “four”, “five”, “six”, “seven”, “eight”, “nine”)
End Function
```

دستور زير با استفاده از تابع بالا، مقدار حرفي معادل عدد ٨ را برگردانده و با استفاده از تابع MsgBox() نمايش مي دهد:

```
MsgBox(IntToString(8))
```

توجه: در صورتي كه مقدار آرگومان index از ١ كمتر يا مقدار آن از تعداد آرگومان هاي choice بيشتر باشد، مقدار برگشتي اين تابع يك مقدار Null خواهد بود.

### IIf(expression,truepart,falsepart)

اين تابع، شكل ساده تر استفاده از ساختار تصميم گيري If Then Else است.
معادل ساختار If Then Else اين تابع به شكل زير است:

```
If expression Then
  result =truepart
Else
  result =falsepart
End If
```

آرگومان expression، يك عبارت شرطي است كه درست يا غلط بودن آن بايد چك شود.
اگر عبارت شرطي موجود در expression، صحيح باشد، عبارت truepart وگرنه عبارت falsepart اجرا خواهد شد.
در مثال زير، چنانچه مقدار A از B كوچكتر باشد، A در Min وگرنه B در Min قرار مي گيرد:

```
Min=Iif(A < B, A, B)
```

دستور زير معادل دستور فوق است:

```
If A < B Then
  Min=A
Else
  Min=B
End If
```

## Switch(expression1,value1,expression2,value2,...)

اين تابع، ليستي از عبارات ورودي را با يكديگر مقايسه كرده و مقدار اولين عبارتي كه صحيح باشد را برمي گرداند. در صورتي كه هيچكدام از عبارات صحيح نباشند، اين تابع يك مقدار Null را برگشت خواهد داد.
مثال زير، مقادير موجود در متغيرهاي X و Y را چك كرده و متناسب با نوع عبارات شرطي تعريف شده، مقداري را برگشت مي دهد:

```
Test =Switch(X>0 and Y>0, 1, X<0 and Y>0, 2, X<0 and Y<0, 3, X<0 and Y<0, 4)
```

در دستور فوق چنانچه X و Y، مثبت باشند مقدار ١ در متغير Test قرار خواهد گرفت.
اگر X مثبت و Y منفي باشد، متغير Test، مقدار ٢ را خواهد داشت و بقيه شروط به همين ترتيب.
توجه: اگر X و Y، صفر باشند، هيچكدام از عبارات صحيح نخواهند بود و مقدار Test، Null خواهد شد.

## Environ(expression)

اين تابع جهت برگشت متغيرهاي محلي به كار مي رود.
متغيرهاي محلي، متغيرهاي سيستم عامل هستند كه با فرمان SET در آن قرار داده شده اند.
براي دسترسي به محتويات يك متغير محلي در سيستم، از يك عدد يا نام آن در آرگومان expression استفاده كنيد.
دستو زير را در نظر بگيريد:

```
Console.WriteLine(Environ(2))
```

به كمك دستور فوق، هم، نام متغير محلي و هم مقدار آن در خروجي نمايش داده مي شود.
به عنوان مثال:

```
TMP=C:\WINDOWS \TEMP
```

براي به دست آوردن تنها، مقدار متغير محلي TMP، بايد نام آن را در آرگومان expression ذكر كنيد همانند زير:

```
Console.WriteLine(Environ("TMP "))
```

دستور فوق، عبارت زير را نمايش خواهد داد:

```
C:\WINDOWS \TEMP
```

مثلا دستور زير، مسير پوشه ويندوز شما را نمايش مي دهد:

```
Console.WriteLine(Environ("WinDir")
```

با استفاده از دستور فوق نيازي به استفاده از تابع API براي به دست آوردن مسير نصب ويندوز نداريد.
براي به دست آوردن مقادير متغيرهاي محلي موجود در سيستم عامل خود، به ترتيب از عدد ١ شروع كرده و اين اعداد را در آرگومان expression قرار دهيد تا زماني كه اين تابع، يك مقدار خالي را برگشت دهد.
نكته: در صورتي كه عدد يا نام متغير محلي كه ذكر مي كنيد وجود نداشته باشد، اي تابع يك مقدار خالي را برگشت مي دهد.

## Beep

اين تابع جهت ايجاد يك صداي بيپ كوتاه به كار مي رود.
طول زمان پخش اين صداي بيپ، قابل تنظيم نيست.
استفاده از اين تابع نيازي به نصب بودن يا داشتن كارت صدا ندارد.

## CallByName(object,procedurename,calltype [,arguments()])

اين تابع جهت اجراي يكي از متدهاي يك شي يا تعيين مقدار خواص شي يا برگرداندن مقدار يكي از خواص شي به كار مي رود.
آرگومان object، نام شي مورد نظر است.
آرگومان procedurename، نام يك خواص يا متد شي است.
آرگومان calltype، نوع عملي كه مي خواهيم انجام شود را مشخص مي كند و يكي از مقادير زير را مي پذيرد:

| مقادير پارامتر calltype در تابع CallByName() | |
|---|---|
| **مقدار** | **عملكرد** |
| **Get** | مقدار يك خاصيت شي را مي خواند |
| **Set** | مقدار يك خاصيت شي را تعيين مي كند |
| **Method** | يك متد از شي را فراخواني و اجرا مي كند |

آخرين آرگومان اين تابع كه arguments نام دارد، يك آرايه از مقادير است.
اگر از مقدار Set استفاده كنيم، بايد مقداري را كه مي خواهيم به يكي از خاصيت هاي شي نسبت دهيم در اين آرگومان ذكر كنيم.
اگر از مقدار Method استفاده مي كنيم، ممكن است متد تابع ما براي انجام كار خود نياز به پارامترها يا مقاديري داشته باشد، پس آنها را به ترتيب در اين آرگومان يعني arguments ذكر مي كنيم.
به عنوان مثال به طور معمول براي آنكه به خاصيت Text از شي TextBox، مقداري را نسبت دهيم، به شكل زير عمل مي كنيم:

```
TextBox1.Text = “Welcome To VB.NET"
```

حالا همين كار را با استفاده از تابع CallByName() انجام مي دهيم:

```
CallByName(TextBox1, “Text ”, CallType.Set, “Welcome to VB.NET ”)
```

توجه: اولين آرگومان كه بايد نام شي مورد نظر در آن ذكر شود، يك مقدار رشته اي نيست بلكه نام اصلي شي را بدون هيچ كاراكتر اضافي بايد ذكر كرد.
به عنوان مثال، دستور زير اشتباه است و يك خطاي زمان اجرا اتفاق خواهد افتاد:

```
ObjectName =“TextBox1”
CallByName ObjectName, “Text”, CallType.Set, “Welcome to VB.NET”
```

در صورتي كه مي خواهيد نام شي را در يك متغير قرار دهيد بايد ابتدا آن را از نوع Object تعريف كنيد. پس دستور فوق را به شكل زير اصلاح مي كنيم:

```
Dim MyObject As Object
MyObject =TextBox1
CallByName MyObject, “Text”, CallType.Set, “Welcome to VB.NET”
```

براي به دست آوردن مقدار خاصيت Text از شي TextBox نيز به شكل زير عمل كنيد:

```
MyText = CallByName(TextBox1, “Text”, CallType.Get)
```

| تاريخچه ويژوال بيسيك | |
|---|---|
| Visual Basic 1.0 | ٢٠ مي ١٩٩١ |
| Visual Basic 2.0 | مارس ١٩٩٢ |
| Visual Basci 3.0 | ژوئن ١٩٩٣ |
| Visual Basci 4.0 | اكتبر ١٩٩٦ |
| Visual Basci 5.0 | آوريل ١٩٩٧ |
| Visual Basic 6.0 | ١٦ ژوئن ١٩٩٨ |
| Visual Basic .NET | ٢٠٠١ |