

# توابع Visual Basic.NET

به وسیله: بهروز راد

کتاب الکترونیکی آموزشی - کاربردی فارسی توابع **VB.NET**

| درباره کتاب... |                                      |
|----------------|--------------------------------------|
| نام کتاب       | آموزش کاربردی توابع Visual Basic.NET |
| نویسنده        | بهروز راد                            |
| تحصیلات        | دانشجوی کامپیوتر                     |
| تاریخ تالیف    | زمستان ۱۳۸۲                          |
| تعداد صفحات    | ۶۵ صفحه                              |

توجه: استفاده از مطالب این کتاب با ذکر نام مؤلف بلامانع است.

## مقدمه:

سپس خدای را که در کنار نعمات بیکران خود، توفیق ارائه این کتاب آموزشی را به دوستداران زبان برنامه نویسی Visual Basic.NET عنایت فرمود.

در برابر انبوه کتاب های الکترونیکی آموزشی انگلیسی زبان در مورد زبان های برنامه نویسی، کمبود کتاب های فارسی در کشور ما به شدت احساس می شود.

از آن مهم تر، کتابی که بر روی یک قسمت از یک زبان برنامه نویسی تمرکز داشته و به نحوی شایسته و به گونه ای ساده به توضیح مبحث پردازد، به ندرت دیده شده است.

از این رو بر آن شدم تا با ارائه این کتاب الکترونیکی آموزشی - کاربردی، مهمترین توابع موجود در زبان VB.NET را که اکثراً با زبان VB.6.0 مشترک هستند، به صورت دسته بندی شده و به طور کامل و با جزئیات کامل تر، با زبانی ساده و ارائه مثال های متعدد جهت یادگیری و فهم بهتر، به دوستداران این زبان قدرتمند ارائه کنم.

از آنجا که هیچ نوشته ای به طور کامل خالی از نقص و ایراد نیست، لذا از تمامی خوانندگان گرامی خواهشمندم اشتباهات کتاب را به طور مستقیم یا بنده از طریق ایمیل زیر در میان بگذارند.

امید است که با این کار کوچک، گامی در جهت افزایش دانایی و سطح علمی دوستداران این زبان برداشته شود.

به امید موفقیت روز افزون.

بهروز راد  
زمستان ۱۳۸۲

[behrouz.rad@gmail.com](mailto:behrouz.rad@gmail.com)

## نکات مهم:

۱) اکثر توابع VB.NET، شبیه به توابع موجود در VB.6.0 هستند و تنها در برخی موارد، تغییراتی در نحوه ی فراخوانی و احیاناً سینتکس تابع وجود دارد. به عنوان مثال تابع Len() در VB.6.0 طول یک رشته را برگشت می دهد اما در VB.NET شما قادر به انجام این کار را با استفاده از متد Length متغیر رشته ای هستید. فرضاً اگر متغیر StrVar به عنوان یک متغیر رشته ای تعریف شده باشد، شما می توانید طول آن را با استفاده از متد Length این متغیر به دست آورید:

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

یا شما می توانید طول این متغیر رشته ای را با قرار دادن آن در آرگومان تابع Len() به دست آورید:

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

۲) در VB.NET توابعی هستند که کار تقریباً یکسانی انجام می دهند و این نکته بسیار مهم است که شما بنا به نیاز خود تشخیص دهید که باید از کدامیک از این توابع استفاده کنید.

۳) اگر شما یکی از آرگومان های اختیاری را حذف کنید، باید به جای آن از کاراکتر کاما استفاده کنید.

۴) آرگومان های اختیاری داخل کروشه قرار دارند. به عنوان مثال شکل کلی تابع Mid() که تعدادی از کاراکترهای یک رشته را از آن استخراج می کند به صورت زیر است:

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

محل شروع کاراکترهایی که باید استخراج شوند به وسیله آرگومان Start و تعداد کاراکترهایی که باید استخراج شوند را آرگومان length تعیین می کند. همانگونه که ملاحظه می شود آرگومان start یک آرگومان اختیاری بوده و چنانچه استفاده نشود به طور پیش فرض محل شروع استخراج کاراکترها از ابتدا یعنی اولین کاراکتر رشته در نظر گرفته می شود. اگر آرگومان length استفاده نشود، از مکان مشخص شده تا به انتهای رشته استخراج می شوند. تنها آرگومان اجباری در این تابع، اولین آرگومان یعنی String می باشد که رشته ای را تعیین می کند که کاراکترها باید از آن استخراج شوند.

در زیر، تمامی توابع توضیح داده شده در این کتاب به صورت دسته بندی شده لیست شده است:

**توجه:** توابعی که در انتهای آنها پرانتز وجود ندارد، توابع بدون آرگومان هستند.

### **توابع ورودی/خروجی (Input/output):**

InputBox(), MsgBox()

### **توابعی که با فایل ها و پوشه ها سر و کار دارند (File and Folder Manipulation):**

ChDir(), ChDrive(), CurDir(), Dir(), FileCopy(), FileDateTime(), FileLen(), GetAttr(), Kill(), Mkdir(), Rename(), Rmdir(), SetAttr()

### **توابعی که برای شناسایی نوع داده ها به کار می روند (Data Type Identification):**

IsArray(), IsDate(), IsDBNull(), IsNothing(), IsNumeric(), IsReference(), TypeName(), VarType()

### **توابع تبدیل نوع متغیر (Variable Type Conversion):**

CBool(), CByte(), CChar(), CDate(), CDBl(), CDec(), CInt(), CLng(), CObj(), CShort(), CSng(), CStr(), CType()

### **توابعی که با رشته ها سر و کار 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()

### **توابعی که برای قالب دهی به داده ها به کار می روند (Data Formatting):**

Format(), FormatCurrency(), FormatDateTime(), FormatNumber(), FormatPercent(), LSet(), RSet(), Str(), Val()

### **توابع ریاضی (Math):**

Abs(), Atan(), Cos(), Exp(), Fix(), Hex(), Int(), Log(), Oct(), Pow(), Round(), Sin(), Sqrt(), Tan()

### **توابعی که با تاریخ و زمان سر و کار دارند (Date and Time):**

DateAdd(), DateDiff(), DatePart(), DateSerial(), DateValue(), Day(), Hour(), Minute(), Month(), MonthName(), Now(), Second(), TimeSerial(), TimeValue(), Weekday(), WeekdayName(), Year()

### **توابعی که جهت انجام عملیات ورودی/خروجی (I/O) بر روی فایل ها مورد**

#### **استفاده قرار می گیرند (File I/O):**

EOF(), FileAttr(), FileClose(), FileOpen(), FileGet(), FilePut(), FreeFile(), Input(), LineInput(), Loc(), Lock(), LOF(), Print(), PrintLine(), Reset(), Seek(), Unlock(), Width(), Write(), WriteLine()

### **توابعی که برای درست کردن اعداد تصادفی به کار می روند (Random Numbers):**

Rnd(), Randomize

### **توابع گرافیکی (Graphics):**

QBColor(), RGB()

### **توابعی که با رجیستری سر و کار دارند (Registry):**

DeleteSetting(), GetAllSettings(), GetSetting(), SaveSetting()

### **توابع کاربردی (Application Collaboration):**

AppActivate(), Shell()

### **توابع متفرقه (Miscellaneous):**

Beep(), CallByName(), Choose(), Environ(), IIf(), Option, Switch()

در ادامه، تمامی توابع ذکر شده به ترتیب دسته بندی و تک تک، همراه با مثال توضیح 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 خواهیم داشت تا در مثال هایی که در توابع دیگر خواهیم زد از توابع دیگر نیز با همدیگر استفاده کنم.

## توابع ورودی/خروجی (Input/output):

ویژوال بیسیک دو تابع اساسی را جهت نمایش یا درخواست اطلاعات به یا از کاربر فراهم کرده است.

این دو تابع عبارتند از: **MsgBox()** , **InputBox()**  
برنامه های کاربردی باید به نحوی مطلوب با استفاده از فرم های جذاب و کاربرپسند با کاربر خود ارتباط برقرار کنند.  
یکی از این راه های ارتباط، استفاده از این دو تابع کاملاً کاربردی می باشد.

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

تابع **InputBox** جهت نمایش یک جعبه گفتگو (**DialogBox**) همراه با یک پیغام و یک جعبه متن به کار می رود.

این کادر محاوره تا زمان ورود اطلاعات از طرف کاربر و کلیک دکمه **OK** یا **Cancel** بر روی صفحه باقی می ماند.

آرگومان های این تابع به شرح زیر می باشند:

**prompt**: پیغامی که در هنگام ظاهر شدن کادر محاوره در آن ظاهر می شود.

نکته: اگر نیاز دارید که متن **Prompt** شما در چندین خط چاپ شود آن را با استفاده از کاراکترهای خطی **VbLf** یا **VbCr** به کار ببرید.

**Title**: عنوانی که در نوار عنوان کادر محاوره ظاهر می شود.

توجه: اگر شما از ذکر این آرگومان اختیاری خودداری کنید، نام برنامه شما در این قسمت نمایش داده می شود.

**Default**: مقدار ورودی پیش فرض کادر محاوره که در جعبه متن قرار خواهد گرفت.

توصیه: اگر قصد تسریع در تصمیم گیری کاربر در وارد کردن مقدار ورودی را دارید، مقداری را حتماً در این قسمت ذکر کنید.

**Xpos,ypos**: مختصات گوشه ی سمت چپ و بالای این کادر محاوره از صفحه نمایش را با این مقادیر تعیین کنید.

نکته: این واحدها بر حسب **Twips** می باشند.

توجه داشته باشید که هر **Twips** برابر با یک - چهارهزارو چهارصدو چهل هزارم اینچ و یک - پانصدو شصت و هفت صدم سانتی متر و یک بیستم نقطه (**Point**) است.

ساده ترین شکل کاربرد تابع **InputBox()** به شکل زیر است:

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

متنی را که کاربر در این کادر محاوره وارد می کند در متغیر **SSN** قرار خواهد گرفت.

مقدار برگشتی این تابع همیشه مقداری از نوع **String** یا رشته ای است، حتی اگر کاربر مقداری عددی را وارد کند.

نکته: همیشه مقداری را که توسط این تابع برگشت داده می شود چک کنید. به عنوان مثال ممکن است کاربر مقداری را در کادر محاوره وارد نکند.

در صورتی که فکر می کنید کاربر مقداری عددی را در کادر محاوره وارد می کند با استفاده از تابع **IsNumeric()** مقدار ورودی را چک کنید تا حتماً از اعداد تشکیل شده باشد.

در صورتی که فکر می کنید کاربر مقداری از نوع تاریخ را وارد می کند با استفاده از تابع **IsDate()** مقدار ورودی را چک کنید و به همین ترتیب، قالب های دیگر.

مثال زیر چک می کند که مقدار ورودی کاربر حتماً تاریخ 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()`، یک کادر محاوره را همراه با یک پیغام نمایش داده و تا زمان بسته شدن که با کلیک کاربر بر روی هر یک از دکمه های آن انجام می پذیرد، بر روی صفحه باقی می ماند. پیغامی که در این کادر محاوره نمایش داده خواهد شد به عنوان اولین آرگومان و با نام **Prompt** تعیین می شود. ساده ترین شکل استفاده از این تابع به شکل زیر است:

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

این دستور یک پیغام را در یک کادر محاوره همراه با یک دکمه `OK` نمایش خواهد داد. تابع `MsgBox()` همچنین قادر به نمایش دکمه ها و آیکن هایی نیز در کادر محاوره خود می باشد. این تابع مقداری عددی را برگشت می دهد که معرف مقدار دکمه ای است که کاربر بر روی آن کلیک کرده. مقادیر زیر کلیه مقادیری هستند که آرگومان **buttons** قادر به دریافت آنهاست:

| مقادیر دکمه ها در پارامتر buttons از تابع MsgBox()                                                     |                                                 |
|--------------------------------------------------------------------------------------------------------|-------------------------------------------------|
| تنها دکمه <code>OK</code> ظاهر خواهد شد. (مقدار پیش فرض)                                               | <code>0</code> یا <code>OKOnly</code>           |
| دکمه های <code>OK</code> و <code>Cancel</code> ظاهر خواهد شد.                                          | <code>1</code> یا <code>OKCancel</code>         |
| دکمه های <code>Abort</code> و <code>Retry</code> و <code>Ignore</code> نمایش داده خواهند شد.           | <code>2</code> یا <code>AbortRetryIgnore</code> |
| دکمه های <code>Yes</code> و <code>No</code> و <code>Cancel</code> نمایش داده خواهند شد.                | <code>3</code> یا <code>YesNoCancel</code>      |
| دکمه های <code>Yes</code> و <code>No</code> را نمایش می دهد.                                           | <code>4</code> یا <code>YesNo</code>            |
| دکمه های <code>Retry</code> و <code>Cancel</code> را نمایش می دهد.                                     | <code>RetryCancel</code>                        |
| مقادیر آیکن ها در پارامتر buttons از تابع MsgBox()                                                     |                                                 |
| آیکن پیغام حساس را نمایش می دهد.                                                                       | <code>16</code> یا <code>Critical</code>        |
| آیکن علامت سوال را نمایش می دهد.                                                                       | <code>32</code> یا <code>Question</code>        |
| آیکن پیغام هشدار را نمایش می دهد.                                                                      | <code>48</code> یا <code>Exclamation</code>     |
| آیکن پیغام اطلاعاتی را نمایش می دهد.                                                                   | <code>64</code> یا <code>Information</code>     |
| مقادیر دکمه های پیش فرض در پارامتر buttons از تابع MsgBox()                                            |                                                 |
| اولین دکمه را به عنوان دکمه پیش فرض انتخاب می کند.                                                     | <code>0</code> یا <code>DefaultButton1</code>   |
| دومین دکمه را به عنوان دکمه پیش فرض انتخاب می کند.                                                     | <code>256</code> یا <code>DefaultButton2</code> |
| سومین دکمه را به عنوان دکمه پیش فرض انتخاب می کند.                                                     | <code>512</code> یا <code>DefaultButton3</code> |
| مقادیر شرطی در پارامتر buttons از تابع MsgBox()                                                        |                                                 |
| تا زمانی که کاربر به کادر محاوره جوابی ندهد نمی تواند بر روی هیچکدام از فرم های برنامه جاری سوئیچ کند. | <code>0</code> یا <code>ApplicationModal</code> |
| تا زمانی که کاربر به کادر محاوره جوابی ندهد نمی تواند به هیچ کدام از برنامه های در حال اجرا سوئیچ کند. | <code>4096</code> یا <code>SystemModal</code>   |

مقادیر دکمه ها به شما این امکان را می دهد که تعیین کنید کدامیک از دکمه ها بر روی کادر  
محواره ظاهر شود.

توجه داشته باشید که شما نمی توانید یک مقدار منحصر به فرد برای این آرگومان تعیین کنید.  
زمانی که رویدادی استثنایی یا غیر عادی در ویندوز رخ دهد، به وسیله آیکون ها نوع این  
رویداد تعیین می شود. این رویداد ممکن است حساس، پرسشی، هشدار و یا اطلاعاتی  
باشد.

مقادیر آیکون ها نوع آیکون ظاهر شونده را برای ما تعیین می کنند.  
مقادیر پیش فرض دکمه ها تعیین می کنند که کدام دکمه در هنگام ظاهر شدن کادر محاوره  
به عنوان دکمه پیش فرض انتخاب شده باشد. در صورتی که تعیین نشود اولین دکمه از سمت  
چپ به عنوان دکمه پیش فرض انتخاب می شود.

مقادیر شرطی نیز نوع رفتار کادر محاوره را به عکس العمل کاربر مشخص می کنند.  
تابع `MsgBox()`، مقداری از نوع `Integer` را برگشت می دهد که معین می کند کدام دکمه  
توسط کاربر فشرده شده است. این مقادیر در جدول زیر لیست شده است:

| مقادیری که تابع <code>MsgBox()</code> برگشت می دهد |                            |
|----------------------------------------------------|----------------------------|
| مقدار                                              | توضیحات                    |
| OK یا 1                                            | دکمه OK فشرده شده است.     |
| Cancel یا 2                                        | دکمه Cancel فشرده شده است. |
| Abort یا 3                                         | دکمه Abort فشرده شده است.  |
| Retry یا 4                                         | دکمه Retry فشرده شده است.  |
| Ignore یا 5                                        | دکمه Ignore فشرده شده است. |
| Yes یا 6                                           | دکمه Yes فشرده شده است.    |
| No یا 7                                            | دکمه No فشرده شده است.     |

به عنوان مثال برای نمایش کادر محاوره ای با دکمه های OK و Cancel و آیکون پیغام هشدار  
به صورت زیر عمل می کنیم:

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

مقداری که توسط این کادر محاوره برگشت داده خواهد شد، یکی از مقادیر ذکر شده در جدول  
بالاست که بسته به انتخاب هر یک از دکمه های OK یا Cancel، یکی از مقادیر 1 یا 2 خواهد  
بود.

در مثال بعدی قصد ظاهر کردن کادر محاوره ای با دکمه های Yes و No همراه با آیکون پیغام  
حساس را داریم:

```
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
```

## توابعی که با فایل ها و پوشه ها سر و کار دارند (File and Folder Manipulation):

توابع زیر برخی از کارهای مدیریتی را در رابطه با فایل ها و پوشه ها همانند انتقال و تغییر نام فایل ها، ایجاد پوشه جدید، حذف فایل یا پوشه و ... انجام می دهند. فرمان های زیر با محتویات فایل ها کاری نداشته و اکثراً همانند فرمان های مدیریتی فایل ها و پوشه های DOS هستند.

### GetAttr(pathname):

این تابع مقداری را از نوع Integer برگشت می دهد که نمایانگر صفت فایل، دایرکتوری یا پوشه ای است که نام آن در آرگومان pathname آمده است. مقداری را که این تابع می تواند برگشت دهد در زیر لیست شده است:

| مقادیری که تابع GetAttr() برگشت می دهد |                                                  |
|----------------------------------------|--------------------------------------------------|
| مقدار                                  | توضیحات                                          |
| Normal یا 0                            | نرمال (هیچ صفتی ندارد)                           |
| ReadOnly یا 1                          | فقط خواندنی                                      |
| Hidden یا 2                            | مخفی                                             |
| System یا 4                            | سیستمی                                           |
| Volume یا 8                            | برچسب درایو                                      |
| Directory یا 16                        | دایرکتوری یا پوشه                                |
| Archive یا 32                          | آرشیو (فایل در آخرین دسترسی تغییر پیدا کرده است) |

نکات مهم در استفاده از این تابع: برای تشخیص اینکه عبارت ما دارای چه صفت یا صفاتی است از عملگر AND استفاده کنید. (ممکن است عبارت دارای چندین صفت باشد) اگر عبارت ما هیچگونه صفتی نداشته باشد، مقدار برگشتی، صفر خواهد بود. به عنوان مثال، جهت تشخیص اینکه عبارت ما دارای صفت فقط خواندنی است یا خیر به صورت زیر عمل می کنیم:

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

اگر عبارت FName دارای صفت فقط خواندنی باشد، مقدار برگشتی که در متغیر Result ذخیره می شود، 1 خواهد بود در غیر اینصورت با فرض اینکه عبارت ما هیچ صفت دیگری ندارد، مقدار Result، صفر خواهد بود. در مثال زیر تعیین می کنیم که آیا عبارت ما دارای صفت Archive است یا خیر:

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

اگر عبارت، دارای مقدار Archive باشد، مقدار Result، 32 خواهد بود. اگر عبارت ما هر دو مقدار فقط خواندنی و آرشیو را دارا باشد، مقدار ReadOnly و Archive با یکدیگر جمع می شوند، در نتیجه Result، 33 خواهد شد. از عملگر OR نیز می توانید استفاده کنید.

### SetAttr(pathname,attributes)

این تابع، صفتی را بر روی يك فایل قرار می دهد. آرگومان pathname، مسیر فایل است که قرار است صفت بر روی آن ایجاد شود و آرگومان attributes يك مقدار عددی است که برای تعیین نوع صفت یا صفتهایی است که قرار است بر روی فایل ایجاد شود. مقداری را که آرگومان attributes می تواند بپذیرد در جدول تابع GetAttr() ذکر شد.

نکات مهم در استفاده از این تابع: از آنجا که این تابع مقدار برگشتی ندارد، آن را در متغیر قرار ندهید.

در این تابع می توان يك يا چند صفت را همزمان ایجاد یا حذف کرد.  
مثال زیر را در نظر بگیرید:

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

این دستور صفت هاي مخفي و آرشیو را بر روي فایل MyFile.txt ایجاد و صفت فقط خواندني آن را بر مي دارد.  
توجه داشته باشید در صورتی که يك صفت را براي يك فایل تعیین مي کنید صفت یا صفات قبلي فایل حذف مي شوند.  
به مثال زیر دقت کنید:

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

در این مثال، ابتدا با استفاده از تابع GetAttr() چك مي شود که فایل مورد نظر ما دارای صفت مخفي است یا خير، اگر دارای صفت مخفي باشد صفت مخفي آن با استفاده از تابع SetAttr() برداشته مي شود.  
همچنین می توان با استفاده از تابع MsgBox() در مورد حذف صفت فایل تایید 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 این مثال، فایل در مورد داشتن صفت فقط خواندني چك مي شود و در صورت داشتن آن با پرسشي از کاربر در مورد حذف آن تایید گرفته مي شود.

## Kill(pathname)

این تابع، فایل یا فایل هايي را که مسیر و نام آنها در آرگومان pathname تعیین مي شود پاک مي کند.

توجه داشته باشید که فایل هاي پاک شده به سطل زباله ویندوز منتقل نمي شوند و برای همیشه پاک خواهند شد.

نکته مهم: در صورتی که با استفاده از تابع FileCopy() فایلي را به سطل زباله ویندوز انتقال دهید، فایل منتقل شده در پنجره سطل زباله ظاهر خواهد شد اما شما قادر به بازیابی آن نخواهید بود.

همچنین شما در این تابع قادر به استفاده از کاراکترهاي جانشيني هستید. (\* معرف همه) و (? معرف يك کاراکتر)

اگر فایلي که نام آن در آرگومان pathname ذکر مي شود وجود نداشته باشد، يك خطاي RunTime (زمان اجرا) اتفاق مي افتد.

سعی کنید از این تابع به شکل زیر استفاده کنید:

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

در این مثال با استفاده از تعریف ساختار Error در برنامه به وسیله Try-Catch، از رخ دادن خطاي زمان اجرا که ممکن است به دلیل وجود نداشتن فایل Log.txt در مسیر ذکر شده

ایجاد شود یا هر دلیل دیگری که ممکن است از حذف فایل جلوگیری کند، جلوگیری شده و در صورت رخ دادن خطا، کنترل برنامه به خط بعد منتقل می شود.

### FileDateTime(pathname)

این تابع تاریخ و زمان ایجاد يك فایل یا تاریخ و زمان آخرین زمانی که فایل تغییر پیدا کرده و ویرایش شده است را برمی گرداند.  
مثال زیر را در نظر بگیرید:

```
Console.WriteLine(FileDateTime("myDocument.txt"))
```

تاریخ و زمان برگشتی فرضاً به شکل زیر خواهد بود:

```
"21/11/01 14:13:02 PM"
```

### FileLen(pathname)

مقدار برگشتی این تابع، عددی از نوع Long Integer خواهد بود که میزان حجم فایلی را مشخص می کند که نام آن در آرگومان pathname آمده است.  
مثال زیر را در نظر بگیرید:

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

میزان حجم فایل تعیین شده در کادر محاوره ظاهر خواهد شد.  
نکته مهم: تابع FileLen() با تابع LOF() (که حجم فایلی را که هم اکنون باز است برمی گرداند) تفاوت دارد. جهت اطلاعات بیشتر، توضیحات تابع LOF() را که در گروه توابع ورودی/خروجی فایل ها (File I/O) دسته بندی شده است ببینید.

### MkDir(path)

این تابع يك پوشه (دایرکتوری) جدید را ایجاد می کند.  
آرگومان path، نام پوشه و مسیری است که قصد ایجاد آن را داریم.  
به مثال زیر توجه کنید:

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

این دستور، پوشه ای جدید به نام New Users در پوشه Users ایجاد می کند. (البته اگر پوشه Users وجود داشته باشد).  
همچنین شما می توانید به يك مسیر یا پوشه دلخواه سوئیچ کرده و پوشه های خود را در آنجا ایجاد کنید.  
مثال زیر را در نظر بگیرید:

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

نکته مهم: حتماً از ساختار Try-Catch در ساخت پوشه ها استفاده کنید چون در صورتی که پوشه ای که قصد ایجاد آن را دارید در مسیر ذکر شده وجود داشته باشد، خطایی رخ خواهد داد. (البته اگر پوشه وجود داشت، می توانید آن را حذف و اقدام به ایجاد پوشه ی جدید کنید)

### Rmdir(path)

این تابع برای حذف پوشه یا دایرکتوری که مسیر و نام آن در آرگومان Path آمده است به کار می رود.

نکات مهم در استفاده از این تابع: آرگومان Path نمی تواند دارای کاراکترهای جانشینی (\*) و (?) باشد. به عبارت دیگر شما نمی توانید چندین پوشه را با يك بار فراخوانی این تابع حذف کنید.

برای حذف پوشه، باید دو شرط زیر برقرار باشد:  
 (۱) پوشه خالی باشد. (۲) پوشه جاری نباشد.  
 در صورت رعایت نکردن هر یک از موارد فوق، یک خطای زمان اجرا اتفاق می افتد.  
 برای حذف پوشه ای که دارای تعدادی فایل است، ابتدا با تابع Kill() فایل های درون آن را حذف و سپس تابع Rmdir را فراخوانی کنید. علاوه بر آن اگر پوشه ی شما دارای تعدادی زیرپوشه نیز باشد، باید آنها را هم قبل از حذف پوشه ی اصلی حذف کنید.  
 فرمان زیر را در نظر بگیرید:

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

در صورتی که پوشه ی Users دارای زیرپوشه ای به نام New User باشد، خطایی اتفاق می افتد.

پس فرمان فوق را به صورت زیر اصلاح می کنیم:

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

## ChDir(path)

این تابع، پوشه (دایرکتوری) جاری را تغییر می دهد.  
 به عنوان مثال برای سوئیچ به پوشه C:\Windows به شکل زیر عمل می کنیم:

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

نکات مهم: اگر در آرگومان path این تابع از آوردن **نام درایو** خودداری شود، تابع ChDir() تلاش می کند تا در **درایو جاری** به پوشه ای که نام آن در آرگومان path مشخص شده سوئیچ کند و اگر پوشه وجود نداشته باشد، یک خطا رخ می دهد و مسیر جاری تغییر پیدا نمی کند.

توجه: تابع ChDir()، پوشه جاری را تغییر می دهد نه درایو جاری را.  
 به عنوان مثال اگر درایو جاری (C:) باشد، دستور زیر پوشه جاری را به پوشه دیگری در درایو (D:) تغییر می دهد اما درایو (C:) به عنوان درایو جاری باقی می ماند:

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

برای تغییر درایو جاری از تابع ChDrive() که به عنوان تابع بعدی توضیح داده شده است استفاده کنید.

شما همچنین می توانید **نام های نسبی** را به عنوان نام پوشه به کار ببرید.  
 به عنوان مثال دستور ChDir("..") پوشه جاری را یک مرحله به بالا در پوشه اصلی انتقال می دهد در صورتی که دستور ChDir("..MyFiles")، پوشه MyFiles را در پوشه اصلی به عنوان پوشه جاری در نظر می گیرد.  
 توجه: هم پوشه ی جاری و هم پوشه ی MyFiles، دو زیرپوشه در یک پوشه اصلی هستند.

## ChDrive(drive)

این تابع درایو جاری را تغییر می دهد. همانند: ChDrive("D:").  
 این دستور، درایو جاری را به درایو D تغییر می دهد.  
 نکات مهم: درایوی که نام آن در آرگومان drive ذکر می شود باید وجود داشته باشد.  
 اگر در آرگومان drive نام چندین درایو ذکر شود، نام اولین درایو جهت تغییر در نظر گرفته می شود.

## CurDir([drive ])

هنگامی که این تابع، بدون آرگومان اختیاری drive فراخوانی شود، نام پوشه جاری را در درایو جاری برمی گرداند.  
 برای اینکه بفهمید که کدام پوشه در درایو دیگری پوشه جاری است، نام درایو را در آرگومان drive ذکر کنید.

به عنوان مثال اگر شما در يك پوشه در درایو (C:) باشید، دستور `CDir = CurDir()` مسیر پوشه جاری را در درایو جاری برمی گرداند. برای اینکه بفهمیم که پوشه جاری در يك درایو دیگر مثلا (D:) چیست، تابع `CurDir()` را به صورت زیر فراخوانی می کنیم:

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

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

این تابع، دو آرگومان اختیاری را دریافت کرده و يك مقدار از نوع رشته ای را که نام فایل یا پوشه ای است که با آرگومان `pathname` (مسیر فایل یا پوشه) یا `attributes` (صفت یا صفات فایل) مطابقت دارد، برگشت می دهد.

نکات مهم در استفاده از این تابع:  
در این تابع مجاز به استفاده از علامت های جایگزین هستید. (\* و ؟)  
مثال زیر را در نظر بگیرید:

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

در این مثال، کلیه فایل هایی که در پوشه Windows وجود دارند (در پوشه اصلی Windows نه در زیر پوشه های آن) چاپ خواهند شد.

اگر هیچ فایلی با آرگومان های ذکر شده در تابع `Dir()` صدق نکند، مقدار برگشتی، يك مقدار خالی خواهد بود.

دومین آرگومان، يك مقدار عددی یا ثابت است که در توضیحات تابع `SetAttr()` آمدند و می تواند يك مقدار یا بیشتر باشد.

در صورتی که از ذکر این آرگومان اختیاری خودداری شود، تنها فایل های Normal (فایل های بدون صفت) برگشت داده خواهند شد.

یکی از کاربردهای این تابع برای تعیین وجود يك فایل یا پوشه در مسیر مشخص شده است. به فرمان زیر توجه کنید:

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

مقدار برگشتی این فرمان در صورت وجود فایل `MSCOMCTL.OCX` در مسیر ذکر شده، همان نام فایل است در غیر اینصورت يك مقدار تهی برگشت داده می شود.

در این مثال می خواهیم بدانیم که چند فایل با پسوند DLL در مسیری که مشخص می کنیم 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 این دستورات، تابع `Dir()` ابتدا با یکی از آرگومان هایش فراخوانی می شود.

اگر يك فایل DLL در مسیر مشخص شده وجود داشت، نام آن برگشت داده می شود و تابع مجدداً فراخوانی می شود اما این بار بدون آرگومان.

هر بار که تابع، نام يك فایل DLL را برگشت می دهد يك واحد به متغیر `DLLFiles` اضافه می شود. این کار برای تمامی فایل های DLL در مسیر مشخص شده ادامه می یابد و در آخر، مقدار متغیر `DLLFiles`، تعداد فایل های مشخص شده در مسیر تعیین شده را دارا می باشد.

خوب می خواهیم تابع را گسترش دهیم و فقط فایل های DLL ای را جستجو کنیم که دارای صفت Hidden (مخفی) هستند.

پس تابع به شکل زیر می شود:

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

شما همچنین قادر به ترکیب چندین صفت با هم هستید. مثلاً فایل های DLL ای که دارای صفات مخفی و فقط خواندنی و سیستمی هستند.

برای تعیین نام زیرپوشه هایی که در یک پوشه قرار دارند باید از صفت `FileAttribute.Directory` استفاده کنید که نام پوشه ای را برگشت می دهد که نرمال یا اصطلاحاً بدون صفت باشد. (در VB.6.0 با نام `vbDirectory` شناخته می شود) برای تعیین اینکه مقدار ورودی، فایل است یا پوشه باید خواص مقدار ورودی را با استفاده از تابع `GetAttr()` بررسی کنید.

مثال زیر تعداد زیرپوشه های موجود در پوشه `System` را به دست می 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)**

این تابع، یک فایل را به یک مکان جدید بر روی هارد دیسک کپی می کند. آرگومان `source_file`، نام فایلی است که قرار است کپی شود. اگر فایل در پوشه جاری باشد فقط ذکر نام فایل کفایت می کند در غیر اینصورت باید مسیر کامل فایل را ذکر کنید.

آرگومان `dest_file` نام فایل و مسیری است که فایل قرار است به آنجا کپی شود. سه نکته مهم در رابطه با استفاده از این تابع:

- ۱) می توانید برای فایلی که قرار است به مکان دیگری کپی شود نام جدیدی در نظر بگیرید.
- ۲) در صورتی که در موقع کپی فایل، برنامه ی دیگری در حال استفاده از فایل باشد، فایل کپی نخواهد شد.
- ۳) در این تابع مجاز به استفاده از کاراکترهای جایگزین نیستید. (\* و ؟) به عبارت دیگر نمی توانید تنها با یک بار فراخوانی این تابع، چندین فایل را همزمان کپی کنید.

مثال زیر را در نظر بگیرید:

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

این دستورات، فایل `Files.txt` را به مکان جدیدی کپی کرده و نام فایل کپی شده را به `NewName.txt` تغییر می دهد. همچنین می توانید همان نام قبلی فایل را بنویسید.

### **Rename(oldpath,newpath)**

با کمک این تابع می توانید نام یک فایل یا پوشه را تغییر دهید. آرگومان `oldpath`، نام و مسیر فایل یا پوشه ای را تعیین می کند که قرار است تغییر نام پیدا کند و نام جدید با استفاده از آرگومان `newpath` تعیین می شود. مثال زیر را در نظر بگیرید:

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

این دستور، پوشه `Users` را در درایو `C` به `All Users` تغییر نام می دهد.

نکات مهم در استفاده از این تابع:  
تغییر نام پوشه، تاثیری در شامل بودن زیرپوشه ها و فایل ها در آن ندارد.  
اگر شما قصد داشته باشید که دو پوشه تو در تو را همزمان تغییر نام دهید يك خطا رخ خواهد داد.  
به عنوان مثال:

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

با اجرای این دستور، يك خطا اتفاق خواهد افتاد.  
برای تغییر نام دو یا چند پوشه ي تو در تو باید آنها را یکی یکی تغییر نام دهید.  
تابع Rename همچنین در صورت نیاز، در هنگام تغییر نام يك فایل قادر به انتقال آن (Cut) به يك مکان جدید بر روی هارددیسك است.  
توجه داشته باشید که این تابع قادر به انتقال پوشه ها نیست.  
به عنوان مثال فرمان زیر فایل Profile1.cps را در مسیر مشخص شده در درایو C همراه با تغییر نام آن به UserProfile.cps به مسیر مشخص شده در درایو D منتقل می کند:

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

این تابع قادر به ایجاد يك پوشه ي جدید نیست.  
تابع Reaname بر روی فایل های باز یا فایل هایی که توسط برنامه های دیگر، همزمان در حال استفاده هستند کاربرد ندارد.  
در این تابع مجاز به استفاده از کاراکترهای جایگزین(\*) و (?) نیستید.

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

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

### IsArray(variable)

اين تابع تعيين مي کند که 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 که بايد بررسي شود. مثال هاي زير را در نظر بگيريد:

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

خروجي اين تابع True است چون که 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 کند که 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 که بايد چک 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 که يك مقدار، DBNull باشد، مقدار True را برگشت مي دهد. DBValue مقداري است که وجود ندارد و با مقدار Nothing تفاوت دارد. DBValue نمايانگر آن است که از ذکر يك مقدار خودداري شده و يا نوع آن تعريف نشده.

### IsNothing(expression)

اين تابع تعيين مي کند که آیا مقداري (متغيري) که از نوع Object (شي) تعريف شده داراي مقدار است يا خير. اگر مقداري 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 که براي ايجاد اين شي در اختيار برنامه قرار گرفته است به حافظه اصلي برگرداند.

به دو شکل مي توان اين تابع را به کار برد:

If IsNothing(obj) Then (۱)

If obj Is Nothing Then (۲)

در شکل دوم، از کلمه ي کليدي (Is) استفاده شده است.

### IsNumeric(expression)

اين تابع يك رشته را تست مي کند و چنانچه کليه کاراکترهاي 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 کاربر درخواست مي شود و سپس مقدار ورودی کاربر با تابع IsNumeric() چک مي شود. چنانچه کاربر کاراکتری غير عددی را وارد کرده بود، به او اطلاع داده مي شود.

### IsReference(expression)

اين تابع تعيين مي کند که 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. ممکن است متغيري که شما مي خواهيد نوع آن را با اين تابع به دست آوريد، تعريف شده يا نشده باشد. فرض مي کنيم که متغيرهاي زير تعريف شده اند:

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

دستورات زير نتايج زير را توليد مي کنند:

```
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 کند با اين تفاوت که اين تابع، يك مقدار عددی را برگشت مي دهد ولي تابع TypeName يك مقدار رشته اي را برگشت خواهد داد. آرگومان Variable، متغيري است که نوع آن بايد مشخص شود. مقاديري که اين تابع برگشت مي دهد در زير ليست شده است:

| مقاديري که 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 این مقادير توسط تابع VarType() برگشت داده خواهد شد.

## توابع تبدیل نوع متغیر (Variable Type Conversion):

متغیرها قابلیت تبدیل شدن به یکدیگر را دارند. تمامی توابع این بخش آرگومان های عددی می پذیرند و آنها را تبدیل به نوع دیگری از متغیرها می کنند. این توابع را در جدول زیر مشاهده می کنید:

| توابع تبدیل نوع متغیر |                                |
|-----------------------|--------------------------------|
| عملکرد                | نام تابع                       |
| تبدیل به Boolean      | <b>CBool(expression)</b>       |
| تبدیل به Byte         | <b>CByte(expression)</b>       |
| تبدیل به Char         | <b>CChar(expression)</b>       |
| تبدیل به Date         | <b>CDate(expression)</b>       |
| تبدیل به Decimal      | <b>CDec(expression)</b>        |
| تبدیل به Double       | <b>CDbl(expression)</b>        |
| تبدیل به Integer      | <b>CInt(expression)</b>        |
| تبدیل به Long         | <b>CLng(expression)</b>        |
| تبدیل به Object       | <b>CObj(expression)</b>        |
| تبدیل به Short        | <b>CShort(expression)</b>      |
| تبدیل به Single       | <b>CSng(expression)</b>        |
| تبدیل به String       | <b>CStr(expression)</b>        |
| نوع سفارشی            | <b>CType(expression, type)</b> |

### CType(varName, typeName)

این تابع، یک مقدار یا عبارت را که مقدار آن در آرگومان varName تعیین شده به نوعی که در آرگومان typeName مشخص شده تبدیل می کند. دستورات زیر را در نظر بگیرید:

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

اولین دستور، مقدار ۱۰۰۰ را که از نوع عدد صحیح (Integer) است و دستور دوم مقدار ۱۰۰۰ را که از نوع رشته ای (String) است به نوع عدد اعشاری با دقت مضاعف (Double) تبدیل می کند.

## توابعي که با رشته ها سر و کار دارند (String Manipulation):

در VB.NET توابع بسیار مهم، مفید و کارآمدی جهت کار با رشته ها وجود دارند. میانگینی که در رابطه با عملکرد برنامه های کاربردی گرفته شده نشان می دهد که توابع رشته ای بیش از توابع عددی کاربرد دارند. در این بخش دو تابع همنام به نام های Mid() و Mid وجود دارد که عملکردی شبیه به هم داشته و در جای خود بحث خواهند شد. تمامی توابع رشته ای VB.NET دارای متد و خواص مشابه و برابری در کلاس System.String همانند کلاس StringBuilder هستند.

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

تابع Asc()، کد اسکی یک کاراکتر را که نام در آرگومان character ذکر شده، برگشت می دهد. این تابع بر روی همه سیستم ها کار می کند حتی اگر آن سیستم از Unicode پشتیبانی نکند. اگر در آرگومان character به جای کاراکتر، یک کلمه یا رشته آورده شود، اولین کاراکتر رشته در نظر گرفته می شود. تابع AscW() عملکردی شبیه به تابع Asc() دارد با این تفاوت که کد اسکی کاراکتر را به صورت Unicode برگشت می دهد در صورتی که تابع Asc() کد اسکی را به صورت بایت برگشت می دهد. از این توابع می توانید جهت تبدیل کاراکتر کلید فشرده شده توسط کاربر به کد اسکی معادل آن استفاده کنید. به عنوان مثال نتیجه خروجی Asc("Ali")، عدد 65 است و نتیجه خروجی Asc("N")، عدد 78 است.

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

این دو تابع دقیقاً عکس توابع Asc() و AscW() عمل می کنند. تابع Chr()، کاراکتر معادل کد اسکی یک عدد از نوع بایت را برگشت می دهد. از این تابع می توانید جهت چاپ حروف یا کاراکترهایی که بر روی صفحه کلید وجود ندارند همانند شکلک ها یا حروفی که کد اسکی آنها از 127 بیشتر است استفاده کنید. تابع ChrW() نیز یک کاراکتر از نوع Unicode یک مقدار عددی را برگشت می دهد. از این توابع می توانید جهت تبدیل کد اسکی کلید فشرده شده توسط کاربر به کاراکتر معادل آن استفاده کنید.

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

تابع Lcase()، در یک رشته، حروف بزرگ رشته را به حروف کوچک تبدیل می کند. تابع Ucase()، در یک رشته، حروف کوچک رشته را به حروف بزرگ تبدیل می کند. مثال زیر را در نظر بگیرید:

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

بعد از اجرای این دستورات متغیرهای Ltitle و Utitle، دارای مقادیر زیر خواهند شد.

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

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

این تابع، زیررشته ای را در رشته دیگر جستجو کرده و محل وجود زیررشته در رشته را برمی گرداند. اولین آرگومان، اختیاری است و مکانی از رشته است که جستجو باید از آنجا آغاز شود. اگر ذکر نشود 1 در نظر گرفته می شود و جستجو از ابتدای رشته آغاز خواهد شد. این تابع، آرگومان string2 را در آرگومان string1 جستجو می کند. مثال زیر را در نظر بگیرید:

```
str1 = "The quick brown fox jumped over the lazy dog"
str2 = "the"
Pos = InStr(str1, str2)
```

بعد از اجرای دستورات فوق، متغیر Pos دارای مقدار 33 خواهد شد.

نکات مهم:

۱) اگر رشته ای که جستجو برای آن آغاز می شود بیش از یک بار در رشته جستجو شونده تکرار شده باشد و **کلمه باشد**، محل آخرین وجود آن برگشت داده می شود همان طور که در این مثال کلمه the، دوبار تکرار شده است اما اگر **قسمتی از یک کلمه باشد**، حتی اگر چندین بار هم تکرار شده باشد، محل اولین وقوع آن برگشت داده خواهد شد. ۲) جستجو به طور پیش فرض با توجه به کوچک یا بزرگ بودن حروف انجام خواهد شد یعنی فرضاً کلمات the، the و THE با هم تفاوت دارند. و این تنظیم به وسیله تغییر آخرین آرگومان یعنی compare که به طور پیش فرض مقدار CompareMethod.Binary را دارد می تواند تغییر کند. برای اینکه بزرگ یا کوچک بودن حروف، در جستجو نادیده گرفته شود، مقدار آرگومان Compare را به CompareMethod.Text تغییر دهید. مثال زیر را در نظر بگیرید:

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

پس از اجرای این دستورات مقدار متغیر pos، 1 خواهد شد. در صورتی که آرگومان compare را به CompareMethod.Binary تغییر دهیم مقدار متغیر pos، 33 خواهد شد.

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

این تابع همانند تابع InStr() عمل می کند با این تفاوت که در تابع InStr() جستجو از ابتدای رشته (چپ به راست) ولی در تابع InStrRev() جستجو از انتهای رشته (راست به چپ) انجام می شود. توجه داشته باشید که جای آرگومان start در این تابع عوض شده.

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

این تابع، دو رشته را با یکدیگر مقایسه کرده و حاصل آن را برمی گرداند. آرگومان های string1 و string2 رشته هایی هستند که باید با یکدیگر مقایسه شوند و آرگومان compare، روش مقایسه را تعیین می کند. اگر دو رشته با هم برابر باشند، تابع مقدار 0 را برمی گرداند. اگر رشته اول بزرگتر از رشته دوم باشد، تابع مقدار 1 را برمی گرداند. اگر رشته دوم بزرگتر از رشته اول باشد، تابع مقدار -1 را برمی گرداند. عمل مقایسه به طور پیش فرض با در نظر گرفتن بزرگ یا کوچک بودن حروف انجام می شود. برای اینکه بزرگ یا کوچک بودن حروف، در مقایسه نادیده گرفته شود، مقدار آرگومان Compare را به CompareMethod.Text تغییر دهید. مثال زیر را در نظر بگیرید:

```
Result=StrComp("Sybex", "SYBEX")
```

پس از اجراي اين دستور، مقدار Result، 1 خواهد شد زیرا عبارت Sybex از SYBEX بزرگتر است و این بدان علت است که در ترتیب حروف اسکي (ASCII)، حرف y کوچک بعد از حرف Y بزرگ قرار دارد. به عبارت دیگر حروف کوچک بر حروف بزرگ ارجعیت دارند.  
مثال دیگر:

```
Result=StrComp("Sybex", "SYBEX", CompareMethod.Text)
```

این همان مثال قبلي است با این تفاوت که در این مثال، بزرگ و کوچک بودن حروف تاثیری در نتیجه مقایسه نخواهد داشت پس دو رشته در این حال با یکدیگر برابر بوده و مقدار Result، 0 خواهد شد.

### Left(string,number)

این تابع، از سمت چپ یک رشته، رشته ای با طول معین را برگشت می دهد.  
آرگومان string، رشته ای است که باید رشته ای به طول تعیین شده در آرگومان number از سمت چپ آن برگشت داده شود.  
مثال زیر را در نظر بگیرید:

```
StrX="Visual Basic.NET"  
StrGet=Left(StrX, 6)
```

پس از اجراي دستورات فوق، 6 کاراکتر اول از سمت چپ متغیر StrX در متغیر StrGet قرار خواهد گرفت و مقدار متغیر StrGet، برابر با "Visual" خواهد شد.

### Right(string,number)

این تابع، از سمت راست یک رشته، رشته ای با طول معین را برگشت می دهد.  
آرگومان string، رشته ای است که باید رشته ای به طول تعیین شده در آرگومان number از سمت راست آن برگشت داده شود.  
مثال زیر را در نظر بگیرید:

```
StrX="Visual Basic.NET"  
StrGet=Right(StrX, 3)
```

پس از اجراي دستورات فوق، 3 کاراکتر اول از سمت راست متغیر StrX در متغیر StrGet قرار خواهد گرفت و مقدار متغیر StrGet، برابر با "NET" خواهد شد.

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

این تابع، بخشی از یک رشته را برگشت می دهد.  
آرگومان string رشته ای است که باید بخشی از آن برگشت داده شود.  
آرگومان start مکانی از رشته را مشخص می کند که استخراج رشته باید از آن مکان شروع شود و آرگومان length طول رشته برگشتی را مشخص می کند.  
نکات مهم:

- 1) استخراج رشته از سمت چپ به راست انجام می شود.
- 2) اگر از ذکر آرگومان اختیاری length خودداری شود، تمامی کاراکترها از مکانی که توسط آرگومان start مشخص شده تا به آخر رشته استخراج می شوند.
- 3) اگر مقدار آرگومان length بیش از تعداد کاراکترهایی باشد که بعد از آرگومان start در رشته وجود دارند، تمامی کاراکترهای مشخص شده در رشته، از مکان Start تا به انتها برگشت داده می شوند.

### Mid(string,start [,length ])=new\_string

در حالت پیشرفته تابع Mid() این تابع وجود دارد که تا حدی شبیه به تابع Mid() عمل می کند.

این تابع رشته ای را در یک رشته دیگر جایگزین می کند. آرگومان string رشته ای است که مقدار new\_String باید در آن جایگزین شود و آرگومان Start موقعیتی از آرگومان (رشته) string را مشخص می کند که عمل جایگزینی باید از آنجا شروع شود و آرگومان length، طول رشته ای را که باید جایگزین شود تعیین می کند. نکات مهم:

- ۱) اگر از ذکر مقدار new\_string خودداری شود، تمامی کاراکترها در رشته از نقطه تعیین شده در آن تا به انتها جایگزین می شوند.
  - ۲) اگر برنامه ای می نویسید که با رشته های طولانی و متن های زیاد سر و کار دارد حتما از کلاس StringBuilder استفاده کنید.
- متغیرهای کلاس StringBuilder از نوع متنی (Text) هستند و کامپایلر .NET. بسیار بهتر از متغیرهای رشته ای (String) قادر به کار با آنهاست. (کلاس برنامه را همیشه حفظ کنید!)

## Len(string)

این تابع، طول یک رشته را برمی گرداند. آرگومان string، رشته ای است که طول آن باید برگشت داده شود. مثال:

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

بعد از اجرای این دستورات، مقدار متغیر NameLen، طول رشته ای است که کاربر در کادر محاوره InputBox() وارد کرده. از این تابع می توانید برای چک کردن اینکه آیا کاربر مقداری را در مکانی وارد کرده یا نه، استفاده کنید. به عنوان مثال:

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

در این دستورات، مقدار TextBox چک می شود که آیا کاربر چیزی در آن وارد کرده یا نه و سپس پیغام مناسب ظاهر خواهد شد. تابع Len() قادر به دریافت هر نوع مقدار غیر رشته ای نیز در آرگومان string خود می باشد. در این حالت مقدار برگشتی این تابع، **تعداد بایت های** مورد نیاز جهت نگهداری مقدار وارد شده در حافظه می باشد **نه طول آن**. به عنوان مثال:

```
Len(12.01)
```

در VB.NET، مقادیر اعشاری به طور پیش فرض از نوع Double (اعشاری با دقت مضاعف) در نظر گرفته می شود پس خروجی تابع در این حالت، عدد 8 یعنی تعداد بایت های لازم جهت نگهداری یک مقدار از Double در حافظه خواهد بود. و به عنوان آخرین مثال:

```
Len(12)
```

در این حالت مقدار برگشتی، 4 خواهد بود. زیرا میزان فضای لازم جهت نگهداری یک عدد از نوع Integer، 4 بایت است.

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

این توابع، فضاهای خالی یک رشته را هر کدام به نحوی حذف می کنند. تابع LTrim()، فضای خالی سمت چپ یک رشته را حذف می کند. تابع RTrim()، فضای خالی سمت راست یک رشته را حذف می کند. تابع Trim()، فضای خالی سمت چپ و راست یک رشته را حذف می کند.

آرگومان string در هر سه تابع، رشته ای است که فضای خالی آن باید حذف شود. مقدار برگشتی هر سه تابع، مقدار رشته ای و از نوع String می باشد و عبارتی است که فضاهای خالی آن بنا به نوع تابع، حذف شده است. کاربرد این تابع معمولاً در چک کردن مقادیر ورودی توسط کاربر برای تشخیص خالی نبودن مقدار ورودی است. مثلاً دستورات زیر چک می کند که آیا مقدار متغیر E\_Mail، خالی است یا خیر و در صورت خالی بودن آن پیغامی ظاهر می گردد.

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

حال در نظر بگیرید که کاربر به جای تایی کاراکتری در مکان مورد نظر، یک یا چندین بار از کلید Space استفاده کند. از آنجا که Space نیز نوعی کاراکتر است پس دستورات بالا کاربرد ندارد.

پس در این حالت برای دقت چک کردن باید از توابع بالا استفاده کرد. معمولاً تابع Trim() از دو تابع دیگر کاربرد بیشتری دارد. پس دستورات بالا را به شکل زیر اصلاح می کنیم:

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

## Space(number)

این تابع، یک رشته کاراکتری از فضای خالی (Blank) ایجاد می کند. آرگومان number، تعداد کاراکتر فضای خالی است که باید ایجاد شود. استفاده از این تابع برای قالب دهی به خروجی های برنامه و ایجاد بافر برای برخی از توابع API مفید است.

## StrDup(number,character)

این تابع، یک کاراکتر را در یک رشته به تعداد معین بار تکرار می کند. آرگومان number عددی است که تعداد تکرار کاراکتر را که به وسیله آرگومان character تعیین می شود، مشخص می کند. آرگومان character می تواند کاراکتر یا کد اسکی باشد. این تابع جایگزین تابع String() در VB.6.0 شده است. مثال زیر را در نظر بگیرید:

```
Result=StrDup(12, "*")
```

با اجرای این دستور، کاراکتر (\*) به تعداد 12 بار تکرار خواهد شد و نتیجه در متغیر Result قرار خواهد گرفت. یعنی متغیر Result دارای مقدار زیر خواهد شد:

```
Result="*****"
```

## StrConv(string,conversion)

این تابع، یک مقدار رشته ای را که به شکل دیگری تبدیل شده است برگشت می دهد. آرگومان string، رشته ای است که باید تبدیل شود و آرگومان conversion نوع تبدیل را مشخص می کند که می تواند یکی از موارد زیر باشد:

| مقادیر پارامتر conversion در تابع StrConv()                |           |
|------------------------------------------------------------|-----------|
| عملکرد                                                     | مقدار     |
| تمامی کاراکترهای موجود در رشته به حروف بزرگ تبدیل می شوند. | UpperCase |
| تمامی کاراکترهای موجود در رشته به حروف                     | LowerCase |

|                                                                     |                    |
|---------------------------------------------------------------------|--------------------|
| کوچک تبدیل می شوند.                                                 |                    |
| اولین کاراکتر هر کلمه در رشته به حرف بزرگ تبدیل می شود.             | <b>ProperCase</b>  |
| کاراکترهای تکی بایستی در رشته به کاراکترهای دوبایستی تبدیل می شوند  | <b>Wide Narrow</b> |
| کاراکترهای دوبایستی در رشته به کاراکترهای تکی بایستی تبدیل می شوند. | <b>Narrow</b>      |
| کاراکترهای Hiragana به کاراکترهای Katakana تبدیل می شوند.           | <b>Katakana</b>    |
| کاراکترهای Katakana به کاراکترهای Hiragana تبدیل می شوند.           | <b>Hiragana</b>    |
| کاراکترهای ساده چینی به کاراکترهای سنتی چینی تبدیل می شوند.         | <b>Traditional</b> |
| کاراکترهای سنتی چینی به کاراکترهای ساده چینی تبدیل می شوند          | <b>Simplified</b>  |

برای تبدیل های چندتایی می توانید این مقادیر را با علامت + با هم AND کنید.  
به مثال زیر توجه کنید:

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

در این دستور، عبارت موجود در متغیر MyString، به حروف کوچک و قالب Unicode تبدیل خواهد شد.

### StrReverse(string)

این تابع، یک مقدار رشته ای را معکوس می کند.  
مثال:

```
Result = StrReverse("VisualBasic")
```

پس از اجرای این دستور، مقدار متغیر Result برابر با "lausiVcisaB" خواهد شد.

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

این تابع به بیان ساده یک رشته یا زیررشته را در یک آرایه رشته ای جستجو کرده و در صورت پیدا کردن آن، یک آرایه را به عنوان خروجی می دهد که خانه یا خانه های آن تا اندیس مقداری که در آرایه اصلی پیدا شده و با آن مقدار پر می شود.  
در حقیقت مقدار برگشتی که در آرایه قرار می گیرد، شامل یکی از مقادیر خانه های آرایه جستجو شونده است.

آرگومان inputstring، یک آرایه یک بعدی و رشته ای است که عمل جستجو باید در آن انجام شود. دقت داشته باشید که این آرایه حتما باید به صورت یک بعدی تعریف شود.  
آرگومان value مقداری رشته ای است که جستجو باید برای آن انجام شود.  
دو آرگومان آخر، اختیاری هستند.

آرگومان include مشخص می کند که آیا کل مقدار value باید در آرایه باشد یا قسمتی از آن هم می تواند به عنوان شرط جستجو به کار رود و یکی از مقادیر True یا False را می پذیرد.  
آرگومان compare، نوع عمل مقایسه را تعیین می کند و مشخص می کند که آیا بزرگ یا کوچک بودن حروف تاثیری در مقایسه و جستجو داشته باشد یا خیر و یکی از مقادیر CompareMethod.Binary (پیش فرض یعنی در نظر گرفته می شود) و یا CompareMethod.Text (یعنی در نظر گرفته نمی شود) را می پذیرد.  
تعداد خانه های آرایه برگشت داده شده بستگی به اندیس محل وجود رشته ای است که در آرایه پیدا شده. فرضا اگر مقدار مورد نظر ما در خانه ی 4 آرایه پیدا شد، تعداد خانه های آرایه برگشتی 4 خانه خواهد بود.

پس، از آنجا که خانه ای که مقدار مورد نظر در آن پیدا می شود از قبل قابل پیش بینی نیست نباید برای خانه های آرایه برگشتی، تعداد تعریف کنیم و یا به عبارت دیگر باید يك آرایه بدون بعد تعریف کنیم.  
مثال زیر را در نظر بگیرید:

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

در این مثال، آرایه ای به نام Names تعریف شده که پنج خانه ی اول آن با 5 مقدار رشته ای پر شده است. این آرایه ای است که ما در آن قصد جستجوی مقدار خود را داریم.  
حال می خواهیم مقدار ذخیره شده در متغیری رشته ای به نام myName را در این آرایه جستجو کنیم. پس تابع Filter() را به شکل زیر فراخوانی می کنیم:

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

پس از اجرای این دستور اگر مقدار ذخیره شده در متغیر myName در آرایه نباشد، در آن صورت آرایه selNames که آن را به صورت آرایه ای بدون بعد تعریف کردیم آرایه ای بدون خانه خواهد بود و یا به عبارت دیگر خاصیت Length آن برابر با 0 خواهد بود.  
اگر مقدار ذخیره شده در متغیر myName، "Ali" باشد، حد بالای آرایه selNames صفر خواهد بود و مقدار اولین خانه آرایه برابر با "Ali" می شود یعنی: selNames(0) = "Ali"  
اگر مقدار ذخیره شده در متغیر myName، "Mehdi" باشد، حد بالای آرایه selNames، يك خواهد بود و مقادیر خانه های اول و دوم آرایه برابر با "Mehdi" می شود یعنی:

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

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

این تابع، يك رشته را در رشته دیگر جستجو کرده و رشته سوم را جایگزین رشته جستجو شده می کند.  
آرگومان expression، رشته ای است که برای جایگزینی جستجو می گردد.  
آرگومان find، رشته ای است که جستجو می گردد.  
آرگومان replacewith، رشته ای است که جایگزین رشته پیدا شده می شود.  
آرگومان اختیاری start، عددی است که مکان شروع جستجو در رشته را تعیین می کند و اگر در نظر گرفته نشود، جستجو از ابتدای رشته شروع می شود.  
آرگومان count، مقداری عددی است که تعداد دفعاتی را که جایگزینی باید انجام گیرد، تعیین می کند. اگر ذکر نشود، کلیه جایگزینی ها انجام خواهد شد.  
و در نهایت آرگومان compare، نوع مقایسه را تعیین می کند که در توابع قبلی در مورد این آرگومان توضیح داده شد.  
مثال زیر را در نظر بگیرید:

```
S = "Visual Basic.NET as powerfull as C#.NET"
K = Replace (S, "NET", "bes")
Console.WriteLine(K)
```

خروجی این قطعه برنامه پس از اجرا به شکل زیر خواهد بود:

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

## Join(list [,delimiter ])

این تابع، مقادیر موجود در يك آرایه يك بعدی - رشته ای را به صورت يك رشته و پشت سر هم برگشت می دهد.  
آرگومان list، آرایه ای يك بعدی و شامل مقادیری رشته ای است که قصد بازگشت آنها به صورت يك رشته را دارید و آرگومان اختیاری delimiter، يك کاراکتر رشته ای است که برای جدا کردن رشته ها از هم در نتیجه خروجی به کار می رود و در صورتی که ذکر نشود، يك فاصله خالی در نظر گرفته می شود و اگر مقدار آن تهی یا "" تعیین شود، مقادیر خانه های آرایه پشت سر هم و بدون فاصله در خروجی ظاهر خواهند شد.  
مثال زیر را در نظر بگیرید:

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

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

بعد از اجرای این دستورات، خروجی به شکل زیر ظاهر خواهد شد:

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

دقت داشته باشید که اگر تمام خانه های آرایه پر نشده باشند، به جای آنها کاراکتری که در آرگومان delimiter تعیین شده قرار خواهد گرفت. تعداد کاراکترهای delimiter که ظاهر خواهد شد، تعداد خانه های آرایه منهای یک است.

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

این تابع، یک رشته را به تعدادی زیررشته ی کوچکتر تقسیم کرده و هر کدام را در یکی از خانه های آرایه قرار می دهد و یا به عبارت بهتر هر کدام از کلمات رشته را در یک خانه ی آرایه قرار می دهد.

خروجی این تابع، یک آرایه ی یک بعدی است که هر کدام از خانه های آن دارای یکی از مقادیر زیررشته هاست.

آرگومان expression، رشته ای است که باید به زیررشته های کوچکتر تقسیم شود. آرگومان اختیاری delimiter، رشته ای است که تقسیم بندی باید بر اساس آن انجام شود. اگر ذکر نشود فضای خالی در نظر گرفته می شود ولی اگر تهی یا "" ذکر شود، کل رشته در اولین عنصر آرایه قرار می گیرد.

آرگومان اختیاری count نیز تعداد زیررشته هایی را که باید برگشت داده شوند مشخص می کند.

اگر مقدار آن 1- باشد، تمامی زیررشته ها برگشت داده خواهند شد.

آرگومان compare که نوع مقایسه را تعیین می کند که در توابع قبلی توضیح داده شد. مثال زیر را در نظر بگیرید:

فرض می کنیم متغیری رشته ای به نام path تعریف کرده ایم که مقدار زیر در آن قرار گرفته است:

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

تابع Split() در این حالت می تواند هر یک اجزای این مقدار را که با کاراکتر ( \ ) از هم جدا شده اند با دستور زیر جدا کرده و در خانه های یک آرایه به نام parts قرار دهد:

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

حال با دستورات زیر می توانیم نتیجه کار را مشاهده کنیم:

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

## توابعي که براي قالب دهی به داده ها به کار می روند (Data Formatting):

VB.NET تمامی توابعی که در VB.6.0 برای قالب دهی به داده ها به کار می روند را پشتیبانی می کند.

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

این تابع به دلیل اهمیت و گستردگی زیادی که دارد، تابعی است که در این کتاب، بیشترین توضیحات در مورد آن داده شده است. این تابع یک مقدار را با فرمت خاصی برمی گرداند. به بیان ساده، این تابع شیوه نمایش اعداد، رشته ها و مقادیر تاریخ و زمان را مشخص می کند.

آرگومان expression، می تواند یک عدد، رشته یا مقداری از نوع تاریخ باشد و آرگومان Format، یک مقدار رشته ای است که مشخص می کند تابع ما مقدار خروجی را با چه قالب یا فرمتی برگشت و نمایش دهد.

به عنوان مثال اگر مقدار ورودی ما یعنی مقدار آرگومان expression از نوع زمان (Time) باشد و قالب آرگومان format را به شکل "hh:mm:ss" انتخاب کرده باشیم، خروجی تابع، زمان حال را نمایش می دهد. مثلاً: "18:22:37"

به عنوان مثالی دیگر، فرمان زیر، مقدار pi را به شکل عددی اعشاری با دقت مضاعف (Double) محاسبه می کند.

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

نتیجه خروجی، عدد "3.14159265358979" خواهد بود. این نتیجه ی خوبی است اما اگر قرار باشد این عدد در یک TextBox قرار بگیرد که برای آن محدودیتی از لحاظ تعداد کاراکترهای ورودی قائل شده آید و این مقدار کمتر از تعداد ارقام این عدد باشد، تمامی ارقام در TextBox قرار نمی گیرند پس باید برای این عدد قالب و فرمتی تعیین کرد. پس از تابع Format() به شکل زیر استفاده می کنیم:

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

این دستور نتیجه خروجی را به شکل 3.1416 نمایش خواهد داد. مثال دیگر: فرض کنیم که شما به عنوان یک حسابدار در یک بانک مشغول به کار هستید و احتیاج دارید تا در نتیجه حساب، یک علامت خاص مثلاً دلار (\$) یا ریال ظاهر شود. فرض کنید که عدد 13,454.332345201 را در نتیجه ی یک حساب به دست می آورید. می خواهیم با استفاده از تابع Format()، علامت (\$) در ابتدای نتیجه ظاهر و همچنین میزان خرده ی پول به دست آمده فقط تا دو رقم حساب شود. پس به شکل زیر عمل می کنیم:

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

نتیجه خروجی به صورت زیر ظاهر خواهد شد:

\$13,454.33

آرگومان های firstdayofweek و firstweekofyear، تنها در قالب دهی به مقادیری از نوع تاریخ به کار می روند. آرگومان firstdayofweek، نام اولین روز هفته را مشخص می کند و می تواند یکی از مقادیر زیر را بپذیرد:

| 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، اولین هفته از سال را مشخص می کند و یکی از مقادیر زیر را می پذیرد:

| 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 می شود.                        |
| 2 یا FirstFourDays                              | سال با هفته ای آغاز می شود که حداقل ۴ روز آن در سال جدید باشد. |
| 3 یا FirstFullWeek                              | سال با هفته ای آغاز می شود که کاملاً در سال جدید باشد.         |

قالب های مختلفی برای نمایش مقادیر عددی، رشته ای، تاریخ و زمان وجود دارد که در لیست زیر تمامی آنها را می بینید:

### کاراکترهایی که می توان برای قالب دهی به تاریخ و زمان به کار برد:

/ : این کاراکتر، جهت جدا کردن سال، ماه و روز از یک مقدار از نوع تاریخ به کار می رود. البته در بعضی مکان ها ممکن است از کاراکترهای دیگری جهت این کار استفاده شود.

: : این کاراکتر، جهت جداکردن ساعت، دقیقه و ثانیه از یک مقدار از نوع زمان به کار می رود. البته در بعضی از مکان ها ممکن است از کاراکترهای دیگری جهت این کار استفاده شود.

d : این کاراکتر، روز یک ماه را به وسیله یک عدد مشخص می کند. این عدد مقداری بین 1 تا 31 است.

dd : این کاراکتر، همانند کاراکتر d رفتار می کند با این تفاوت که این تابع، قبل از ارقام یک تا نه، صفر (0) می گذارد. 03،02،01 و... اما کاراکتر d، اعداد یک تا نه را به صورت معمول نشان می دهد یعنی 3،2،1 و...

ddd : این کاراکتر، نام اختصاری یک روز را نشان می دهد. مثلاً: (Sun-Sat)

dddd : این کاراکتر، نام کامل روز را نشان می دهد. مثلاً: (Sunday-Saturday)

w : این کاراکتر، عددی را که نمایانگر روز هفته است نشان می دهد. مثلاً عدد 1 برای Sunday و 7 برای Saturday

ww : این کاراکتر، عددی را که نمایانگر هفته سال است نمایش می دهد. این عدد بین 1 تا 54 است.

M : این کاراکتر، عددی را که نمایانگر ماه سال است نشان می دهد. اگر با کاراکترهای hh یا hh بیاید، به جای نمایش عدد ماه، دقیقه را نشان خواهد داد.

MM : این کاراکتر همانند کاراکتر M رفتار می کند با این تفاوت که این تابع قبل از ارقام 0 تا 9، صفر (0) می گذارد اما کاراکتر M، اعداد را به صورت معمول نمایش می دهد.

MMM : این کاراکتر، نام اختصاری یک ماه را نشان می دهد. مثلاً: (Jan-Dec)

MMMM : این کاراکتر، نام کامل ماه را نشان می دهد. مثلاً: (January-December)

**q** : این کاراکتر، یک چهارم یک سال را با اعدادی بین 1 تا 4 نمایش می دهد. (متناظر فصل)  
**y** : این کاراکتر، عددی را که نمایانگر روز سال است نمایش می دهد. (1 تا 366)  
**yy** : این کاراکتر، عددی دو رقمی را نشان می دهد که نمایانگر سال است. (00 تا 99)  
**yyyy** : این کاراکتر، عددی 4 رقمی را نمایش می دهد که نمایانگر سال است. (0000 تا 9999)

**h** : این کاراکتر، عددی را که نمایانگر ساعت است نمایش می دهد. (0 تا 12)  
**hh** : این کاراکتر همانند کاراکتر h رفتار می کند با این تفاوت که این تابع قبل از ارقام 0 تا 12، صفر (0) می گذارد اما کاراکتر h، اعداد را به صورت معمول نمایش می دهد.  
**H** : این کاراکتر، شکل کاملتر کاراکتر h است و ساعت را به صورت 0 تا 24 نمایش می دهد. به عنوان مثال اگر ساعت فعلی سیستم 3 باشد، 3 صبح را با عدد 3 و 3 بعد از ظهر را با عدد 15 نمایش می دهد.

**HH** : این کاراکتر، همانند کاراکتر H رفتار می کند با این تفاوت که قبل از ارقام 0 تا 9، صفر (0) می گذارد. (00 تا 09) ولی کاراکتر H، ارقام 0 تا 9 را به صورت معمول نمایش می دهد.  
**m** : این کاراکتر، دقیقه را نمایش می دهد، منتها بدون گذاشتن عدد صفر قبل از ارقام 0 تا 9. بازه ی این کاراکتر، عددی بین 0 تا 59 است.

**mm** : همانند کاراکتر m رفتار کرده با این تفاوت که قبل از اعداد 0 تا 9، صفر (0) می گذارد.  
**s** : این کاراکتر، ثانیه را نمایش می دهد، منتها بدون گذاشتن عدد صفر قبل از ارقام 0 تا 9. بازه ی این کاراکتر بین 0 تا 59 است.

**ss** : همانند کاراکتر s رفتار کرده با این تفاوت که قبل از اعداد 0 تا 9، صفر (0) می گذارد.  
**AM / PM** : مشخص می کند که هم اکنون قبل از ظهر است (AM) یا بعد از ظهر (PM)  
**am / pm** : همان کاراکتر AM/PM است، فقط قبل یا بعد بودن ظهر را با حروف کوچک نمایش می دهد. (am/pm)

**A / P** : قبل یا بعد بودن ظهر را با حروف اختصاری مشخص می کند. (A یا P)  
**a / p** : همان کاراکتر A/P است، فقط قبل یا بعد بودن ظهر را با حروف کوچک نمایش می دهد. (a یا p)

**AMPM** : این کاراکتر نیز قبل یا بعد بودن ظهر را مشخص می کند اما به جای این کاراکتر، کاراکتری که به وسیله سیستم تعیین شده قرار خواهد گرفت. به عنوان مثال در ویندوز های فارسی با کلمات "صبح" و "عصر" نمایش می دهد.

برای تغییر نحوه نمایش، به سربرگ Regional Options از قسمت Regional and Language Options در Control Panel مراجعه کنید.

کاراکترهایی که می توان برای قالب دهی به مقادیر عددی به کار برد، ۱۶ کاراکتر هستند که نحوه ی کار با آنها بر عهده ی خوانندگان عزیز است.

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

این تابع، یک مقدار عددی را به شکل یک مقدار ارزی (Currency) قالب بندی شده همراه با نمادی که در سربرگ Regional Options از قسمت Regional and Language Options در سربرگ Control Panel مشخص شده است، برگشت می دهد. به عنوان مثال در ویندوزهای فارسی، نماد مقادیر ارزی با کلمه "ریال" و در ویندوزهای انگلیسی با کاراکتر "\$"، مشخص می شود.

برای تغییر نحوه نمایش این مقادیر، به قسمت گفته شده در بالا مراجعه کنید. در این تابع، تمامی آرگومان ها به جز اولین آرگومان، اختیاری هستند. آرگومان expression، مقداری عددی است که قصد برگشت آن با فرمت Currency را داریم. آرگومان numDigitsAfterDecimal، مقداری عددی است که مشخص می کند چند عدد در سمت راست بعد از اعشار باید قرار بگیرند. مقدار پیش فرض، 1- است یعنی تنظیمات Regional and Language Options را به کار می برد.

آرگومان includeLeadingDigit، یک مقدار ثابت Tristate است. (ثابت Tristate، مقداری است که می تواند True، False یا UseDefault باشد) و مشخص می کند که آیا اگر مقداری اعشاری کمتر از 1 بود، در سمت چپ اعشار، عدد صفر قرار گیرد یا خیر. به عنوان مثال عدد هفت دهم که می تواند به شکل "0.7" و یا "7." نمایش داده شود.

آرگومان `useParensForNegativeNumbers`، نیز يك مقدار ثابت `Tristate` است که مشخص مي کند آیا اعداد مثبت را با دو پرانتز در دو طرف آنها برگشت دهد یا خير. آرگومان `groupDigits` هم يك مقدار ثابت `Tristate` است و مشخص مي کند آیا اعداد بیش از 5 رقمي با کاراکتر ي که در قسمت `Regional and Language Options` مشخص شده از یکدیگر جدا شوند یا خير. به عنوان مثال در صورت `True` بودن این آرگومان، عدد 234,456 به طريقي که مشاهده مي کنید با کاراکتر (,) از یکدیگر جدا شده و اعداد از سمت راست، سه تا سه تا جدا مي شوند. در صورت `False` بودن این آرگومان، اعداد پشت سر هم و بدون کاراکتر جداکننده ظاهر مي شوند.

### **FormatDateTime(date [,namedFormat ])**

این تابع، يك مقدار از نوع تاريخ (Date) یا زمان (Time) را قالب بندي مي کند. آرگومان `date`، يك مقدار از نوع تاريخ یا زمان است که باید قالب بندي شود. آرگومان اختياري `namedFormat`، مشخص مي کند که چه نوع قالب بندي باید براي تاريخ یا زمان به کار رود. این آرگومان مي تواند يکي از مقادير زیر را بپذیرد:

**GeneralDate**: اگر در آرگومان `date`، يك مقدار از نوع تاريخ آمده باشد، تاريخ به شکل کوتاه مثلا "14/02/2004" نمایش داده خواهد شد و اگر مقدار از نوع زمان باشد، زمان به شکل کامل مثلا "عصر 06:47:55" نمایش داده مي شود.  
توجه: اگر هر دو مقدار تاريخ و زمان به کار روند، پشت سر هم نمایش داده خواهند شد.  
**LongDate**: این مقدار ثابت، يك مقدار از نوع تاريخ را به صورت طولاني نمایش و برگشت مي دهد. مثلا: "شنبه، ۱۴ آبان، ۱۳۸۲"  
**ShortDate**: این مقدار ثابت، يك مقدار از نوع تاريخ را به صورت کوتاه نمایش و برگشت مي دهد. مثلا: "۱۴/۰۲/۲۰۰۴"  
**LongTime**: این مقدار ثابت، يك مقدار از نوع زمان را به صورت طولاني نمایش مي دهد. مثلا: "۰۶:۵۸:۳۴ عصر"  
**ShortTime**: این مقدار ثابت، يك مقدار از نوع زمان را به صورت کوتاه نمایش مي دهد. مثلا: "۱۹:۲۳"

**نکته مهم:** تمامی مقادير ثابت بالا، بر اساس تنظيماتي که در قسمت `Regional and Language Options` از `Control Panel` صورت گرفته است عمل مي کنند.

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

این تابع، يك مقدار عددي را با قالب بندي دیگری به صورت عددي برگشت مي دهد. تمامی آرگومان هاي این تابع در توضیحات تابع `FormatCurrency()` بيان شدند.

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

این تابع، يك عدد را بر حسب درصد برگشت مي دهد يعني عدد را در ۱۰۰ ضرب کرده و با نماد درصد (%) و با قالبی که مشخص شده است برگشت و نمایش مي دهد. تمامی آرگومان هاي این تابع در توضیحات تابع `FormatCurrency()` بيان شدند.

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

این توابع، به تعدادي که مشخص مي شود، در سمت چپ (`Lset`) یا راست (`Rset`) يك عبارت، فضاي خالي قرار مي دهند. آرگومان `string` که مي تواند از هر نوع مقداري باشد، رشته اي را مشخص مي کند که باید در سمت چپ یا راست آن، فضاي خالي اضافه شود.

آرگومان len نیز مقداری عددی است که تعداد فضای خالی را که باید ایجاد شود مشخص می کند.

مثال های زیر را در نظر بگیرید:

```
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))
```

خروجی پس از اجرا به شکل زیر خواهد شد:

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

### Val(string) , Str(number)

تابع Val()، یک مقدار رشته ای را به عنوان ورودی دریافت کرده و آن را به صورت مقداری عددی برگشت می دهد.

تابع Str() دقیقاً عکس تابع Val() رفتار کرده و یک مقدار عددی را دریافت کرده و آن را به صورت مقداری عددی برگشت می دهد.

مکانیزم کار تابع Val() به این شکل است که از ابتدای رشته ورودی شروع به خواندن آن می کند و زمانی که به کاراکتری **غیر عددی** رسید جستجو متوقف و نتیجه برگشت داده می شود.

مثال زیر را در نظر بگیرید:

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

نتیجه ی خروجی، عدد 18 خواهد بود زیرا کاراکتر ":" کاراکتری غیر عددی است.

مثالی برای تابع Str()

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

نتیجه خروجی، رشته "18" خواهد بود.

## توابع ریاضی (Math):

در VB.NET تمامی توابع ریاضی VB.6.0 به عنوان متدهایی در کلاس Math قرار داده شده اند. پس در این بخش آنها را با نام متد صدا می زنیم. برای استفاده از متدهای ریاضی در VB.NET باید کلاس Math را به برنامه ی خود اضافه یا قبل از استفاده از این متدها، کلمه Math را قبل از نام متد ذکر کنید مثل: Math.Abs، Math.Cos

### Abs(expression)

این متد، قدر مطلق یک عبارت عددی را محاسبه می کند. آرگومان expression، عددی است که باید قدر مطلق آن محاسبه گردد. اگر expression بزرگتر یا مساوی صفر باشد، نتیجه ی متد همان عبارت expression خواهد بود وگرنه عبارت expression در عدد -1 ضرب شده و نتیجه برگشت داده می شود. مثال زیر را در نظر بگیرید:

Abs(1.01)  
Abs(-1.01)

نتیجه هر دو این دستورات، عدد 1.01 خواهد بود.

### Atan(expression)

این متد، آرک تانژانت یک مقدار را محاسبه و برگشت می دهد. آرگومان expression مقداری است که آرک تانژانت آن باید محاسبه گردد. نتیجه این متد، مقداری بین -1.57079 تا 1.57079 است. مثال:

Atan(2.3)=1.16

### Cos(expression)

این متد، کسینوس یک زاویه را بر حسب رادیان محاسبه می کند. آرگومان expression، عبارتی است که باید کسینوس آن محاسبه شود. حاصل متد Cos()، مقداری بین -1 تا 1 است. مثال:

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

### Exp(expression)

این متد، توانی از e (عدد نپر) را برمی گرداند. نکته: تقریباً  $e=2.7182$  آرگومان expression، مقداری عددی (توانی) است که باید عدد نپر آن محاسبه گردد و می تواند یک عدد صحیح، اعشاری، مثبت یا منفی باشد. مثال:

Exp(2)=7.31

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

هر دو این متدها، مقداری عددی را دریافت کرده و عددی از نوع Integer (صحیح) را برگشت می دهند. آرگومان expression عبارتی عددی است که به عنوان ورودی این متدها داده می شود.

اگر مقدار ورودی، صحیح و مثبت یا منفی باشد، همان مقدار expression برگشت داده می شود.

اگر مقدار ورودی، اعشاری و مثبت باشد، قسمت اعشاری حذف شده و قسمت صحیح برگشت داده می شود.

اگر مقدار ورودی، اعشاری و منفی باشد دو حالت دارد:

در حالت اول اگر از متد `Int()` استفاده شود، اولین عدد صحیح منفی کوچکتر یا مساوی عبارت ورودی برگشت داده می شود.

در دومین حالت اگر از متد `Fix()` استفاده شود، اولین عدد صحیح منفی بزرگتر یا مساوی عبارت ورودی برگشت داده می شود.

مثال های زیر را در نظر بگیرید:

```
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
```

پیشنهاد می کنم اگر می خواهید که از دست قسمت اعشاری يك عدد خلاص شوید و آن را اصطلاحاً گرد کنید، به شکل زیر عمل کنید:

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

Value، مقداری عددی است که قصد گرد کردن آن را دارید.

این دستور، فرمان بسیار کارآمدی برای گرد کردن هر نوع مقداری عددی است با این وجود، متد `Round()` که در قسمت بعد توضیح داده شده است به شیوه ای آسانتر، بهتر و کارآمدتر این کار را انجام می دهد.

### **Round(expression [,numdecimalplaces ])**

این متد، يك عدد اعشاری را گرد می کند.

آرگومان expression، مقداری است که باید گرد شود.

آرگومان اختیاری numdecimalplaces مشخص می کند که این مقدار تا چند رقم بعد از اعشار باید گرد شود. اگر از ذکر این آرگومان خودداری شود، يك مقدار از نوع صحیح (Integer) برگشت داده می شود.

مثال های زیر را در نظر بگیرید:

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

### **Log(expression)**

این متد، لگاریتم طبیعی (یا لگاریتم پایه e که تقریباً  $e=2.7182$  است) مقداری را برمی گرداند.

آرگومان expression، مقداری عددی است که لگاریتم طبیعی آن باید محاسبه شود و حتماً باید مقداری مثبت باشد.

عبارت `Log(Exp(N))` و `Exp(Log(N))` هر دو عدد N را برمی گردانند.

لگاریتم طبیعی، لگاریتم پایه عدد e (نپر) است.

مقدار دقیق عدد نپر را می توانید با دستور `Exp(1)` به دست بیاورید.

برای محاسبه لگاریتم ها در پایه های دیگر، عدد لگاریتم طبیعی را بر لگاریتم پایه تقسیم کنید.

به عنوان مثال، دستور زیر لگاریتم يك عدد در پایه ی 10 را محاسبه می کند.

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

## Hex(expression) , Oct(expression)

این متدها عددی را به عنوان ورودی دریافت کرده و آن را به مبنای ۸ (Oct) یا مبنای ۱۶ (Hex) تبدیل می کنند و به صورت یک مقدار رشته ای (String) برگشت می دهند. به عنوان مثال عبارت Hex(47)، مقدار "2F" و عبارت Oct(47) مقدار "57" را برگشت می دهند.

برای تعیین کردن اعداد در مبنای ۱۶، ابتدای آنها کاراکتر "&H" را قرار دهید. کاراکتر و نماد معادل برای اعداد در مبنای ۸، کاراکتر "&O" می باشد. فرض کنید مقادیر زیر تعریف شده باشند:

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

حال دستور Oct(Dvalue)، مقدار رشته ای "307" و دستور Hex(Dvalue) مقدار رشته ای "3F" را برگشت می دهد.

می توانید از کاراکترهای معادل این متدها نیز استفاده کنید. مثال زیر را در نظر بگیرید:  
MsgBox ("The number 3F in decimal is " & &H3F)  
مقداری که نشان داده می شود، عدد 63 خواهد بود.

## Pow(value1,value2)

این متد، دو عدد را به عنوان ورودی دریافت کرده و عدد اول را به توان عدد دوم می رساند. دستور زیر را در نظر بگیرید:

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

در خروجی، عدد ۸ نمایش داده خواهد شود. ۲ به توان ۳ برابر با ۸ است.

## Sin(expression)

این متد، سینوس زاویه ای را بر حسب رادیان محاسبه و برگشت می دهد. آرگومان expression، عبارتی است که سینوس آن باید محاسبه شود. نتیجه متد Sin()، مقداری بین 1 تا -1 است. مثال:

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

## Sqrt(expression)

این متد، مربع (عدد به توان ۲) یک مقدار عددی را محاسبه و برگشت می دهد. آرگومان expression، عددی است که باید مربع آن محاسبه شود.

## Tan(expression)

این متد، تانژانت زاویه ای را بر حسب رادیان محاسبه می کند. مثال:

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

## توابعي که با تاريخ و زمان سر و کار دارند (Date and Time):

توابع کاربردي و کارآمدی جهت کار با تاريخ و زمان در VB.NET وجود دارد که ممکن است در برخورد اول با این توابع، از تنوع آنها تعجب کنید!

نکته مهم: در VB، مقاديري که از نوع تاريخ هستند باید حتماً بين دو علامت (#) قرار گیرند.

### Now()

این تابع، تاريخ و زمان جاری سیستم را برمي گرداند.  
مثال:

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

به عنوان مثال، خروجي دستور فوق می تواند "02/10/2004 09:23:10 PM" باشد.  
توجه داشته باشید که تاريخ و زمان با کاراکتر خالي (Blank) از هم جدا شده اند.  
نکته مهم: برای استخراج تنها تاريخ یا زمان برگشت داده شده از این تابع، از خواص Date و TimeOfDay این تابع به شکل زیر استفاده کنید.

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

### Day(date)

این تابع، يك مقدار از نوع تاريخ را به عنوان ورودی می پذیرد و روز آن را برمي گرداند.  
به عنوان مثال اگر مقدار ورودی، تاريخ 12/15/2004 باشد، مقدار خروجی عدد 15 يعني روز تاريخ خواهد بود.  
مثال زیر روز جاری سیستم را برگشت می دهد.

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

### Weekday(date [,firstdayofweek ])

این تابع، عددی بين 1 تا 7 را برمي گرداند که مشخص کننده روز هفته است.  
مثلاً، 1 برای Sunday، 2 برای Monday و به همین ترتیب.  
آرگومان date يك مقدار از نوع تاريخ است.  
آرگومان اختیاری firstdayofweek، اولین روز هفته را مشخص می کند و می تواند یکی از مقادیر زیر را بپذیرد:  
مقدار این آرگومان به طور پیش فرض Sunday يعني یکشنبه است.

مطابق با تنظیمات سیستم >>> 0 یا System  
یکشنبه (مقدار پیش فرض) >>> 1 یا Sunday  
دوشنبه >>> 2 یا Monday  
سه شنبه >>> 3 یا Tuesday  
چهارشنبه >>> 4 یا Wednesday  
پنج شنبه >>> 5 یا Thursday  
جمعه >>> 6 یا Friday  
شنبه >>> 7 یا Saturday

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

این تابع همانند تابع Weekday() عمل می کند با این تفاوت که این تابع، نام روز معادل يك عدد که بين 1 تا 7 است را برمي گرداند.  
آرگومان weekday يك مقدار عددی صحیح است و عددی بين 1 تا 7 را می پذیرد.  
آرگومان اختیاری abbreviate، يك مقدار منطقی (Boolean) است که مشخص می کند آیا نام روزی که برگشت داده می شود به صورت کوتاه و اختصاری باشد یا خیر. به طور پیش فرض این مقدار False است يعني نام به صورت کامل برگشت داده می شود.

آرگومان اختیاری firstdayofweek نیز اولین روز هفته را مشخص می کند و می تواند یکی از مقادیری را که آرگومان firstdayofweek از تابع Weekday می پذیرد، بپذیرد. مقدار این آرگومان به طور پیش فرض Sunday یعنی یکشنبه است.

### Month(date)

این تابع از مقدار تاریخ، ماه آن را برمی گرداند که عددی بین 1 تا 12 است. آرگومان date، مقداری از نوع تاریخ است که ماه آن باید برگشت داده شود. به عنوان مثال برای به دست آوردن عدد متناظر ماه جاری سیستم از دستور زیر استفاده کنید:

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

### MonthName(month [,abbreviate ])

این تابع نیز همانند تابع Month() عمل می کند با این تفاوت که به جای عدد ماه، نام ماه را برگشت می دهد. آرگومان month، مقداری عددی و بین 1 تا 12 است که باید ماه متناظر آن برگشت داده شود. آرگومان اختیاری abbreviate، یک مقدار منطقی (Boolean) است که مشخص می کند آیا نام ماهی که برگشت داده می شود به صورت کوتاه و اختصاری باشد یا خیر. به طور پیش فرض این مقدار False است یعنی نام ماه به صورت کامل برگشت داده می شود.

### Year(date)

این تابع از مقدار تاریخ، سال آن را برمی گرداند. مقداری که این تابع برمی گرداند، عددی بین 0 تا 9999 می باشد. آرگومان date، مقداری از نوع تاریخ است که سال آن باید برگشت داده شود. به عنوان مثال برای به دست آوردن عدد متناظر سال جاری سیستم از دستور زیر استفاده کنید:

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

### Hour(time)

این تابع از مقدار زمان، بخش ساعت آن را برمی گرداند. مقداری که این تابع برمی گرداند بین 0 تا 24 است. آرگومان Time، مقداری از نوع زمان (Time) است که ساعت آن باید برگشت داده شود. به عنوان مثال دستور زیر ساعت جاری سیستم را نشان می دهد:

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

### Minute(time)

این تابع از مقدار زمان، دقیقه آن را برمی گرداند. مقداری که این تابع برمی گرداند بین 0 تا 59 است. آرگومان time، مقداری از نوع زمان است که دقیقه آن باید برگشت داده شود. به عنوان مثال دستور زیر دقیقه جاری سیستم را نشان می دهد:

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

### Second(time)

این تابع از مقدار زمان، ثانیه آن را برمی گرداند. مقداری که این تابع برمی گرداند بین 0 تا 59 است. آرگومان time، مقداری از نوع زمان است که ثانیه آن باید برگشت داده شود. به عنوان مثال دستور زیر ثانیه جاری سیستم را نشان می دهد:

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

## DateSerial(year,month,day)

این تابع، سه عدد صحیح را به عنوان ورودی دریافت کرده و آن را به تاریخ تبدیل می کند. آرگومان های month, year و day به ترتیب سال، ماه و روز هستند. دستور زیر را در نظر بگیرید:

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

خروجی این دستور، رشته ی "01/10/2002" خواهد بود. همچنین از این تابع می توانید جهت برخی محاسبات بر روی مقدار تاریخ استفاده کنید. به عنوان مثال فرض کنید می خواهید بدانید که نودمین روز سال در چه تاریخی از سال قرار خواهد گرفت. می توانید از دستور زیر جهت آگاهی از این موضوع استفاده کنید:

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

خروجی این دستور، عبارت "03/30/1996" خواهد بود. به عنوان مثالی دیگر می خواهیم بدانیم که ۱۰۰۰ روز بعد از تاریخ فعلی سیستم در چه تاریخی قرار خواهد گرفت. پس به شکل زیر عمل می کنیم:

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

شما همچنین قادر به اضافه کردن یا کم کردن (تفریق کردن) یک عدد متناظر ماه ها به آرگومان month و همچنین یک عدد متناظر سال به آرگومان year هستید. توجه داشته باشید که تعداد روزهای هر ماه در این تابع ۳۰ روز است.

## DateValue(date)

این تابع، یک عبارت رشته ای را به عنوان ورودی دریافت کرده و آن را به یک مقدار از نوع تاریخ تبدیل می کند. به عنوان مثال اگر این تابع را به شکل زیر به کار ببریم:

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

عبارتی که در خروجی چاپ می شود، "12/25/2002" خواهد بود. یکی از کاربردهای سودمند و مفید این تابع در هنگامی است که با تابع DateDiff() به کار رود. تابع DateDiff() در ادامه توضیح داده می شود. فرض کنید می خواهیم تعداد روزهای بین دو تاریخ را محاسبه کنیم: دستور زیر این کار را برای ما انجام می دهد:

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

عددی که در خروجی ظاهر خواهد شد، 1096 یعنی تعداد روزهای بین این دو تاریخ خواهد بود.

## TimeSerial(hours,minutes,seconds)

این تابع، سه عدد صحیح را به زمان مشخصی تبدیل می کند. آرگومان های Hour، Minute و Second به ترتیب اعداد صحیحی هستند که ساعت، دقیقه و ثانیه را که باید به زمان تبدیل شوند، تعیین می کند. مثال زیر را در نظر بگیرید:

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

این تابع، مقدار "4:10:55 AM" را برگشت می دهد. این تابع معمولاً برای محاسبه ی زمان به کار می رود.

به عنوان مثال، فراخوانی تابع `TimeSerial()` به شکل زیر، ۲ ساعت و ۱۵ دقیقه و ۳۲ ثانیه قبل از ساعت 4:13:40 PM را برمی گرداند.

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

زمان برگشتی، 1:58:08 PM خواهد بود.

## TimeValue(time)

این تابع، یک مقدار رشته ای را دریافت کرده و آن را به مقدار زمانی تبدیل می کند. همانند تابع `DateValue()`، این تابع نیز می تواند جهت انجام عملیاتی نظیر جمع یا تفریق زمان ها به کار رود.

## DateAdd(interval,number,date)

این تابع همان طور که از نام آن نیز پیداست، مقداری را به تاریخی مشخص اضافه کرده و تاریخ جدید را برگشت می دهد. آرگومان `interval`، یک واحد زمانی همانند روز، ساعت، هفته و ... است که فاصله دوره ای را مشخص می کند که تاریخ جدید باید بر حسب آن محاسبه و برگشت داده شود و می تواند یکی از مقادیر زیر را بپذیرد:

**Year:** سال

**Quarter:** یک چهارم سال یا ۳ ماه یا یک فصل

**Month:** ماه

**DayOfYear:** روز از سال

**Day:** روز

**WeekDay:** روز از هفته

**WeekOfYear:** هفته از سال

**Hour:** ساعت

**Minute:** دقیقه

**Second:** ثانیه

آرگومان `number`، تعداد هر دوره ی زمانی را مشخص می کند. مثلاً ممکن است بخواهیم بدانیم که ۳ ماه بعد از تاریخ فعلی سیستم چه تاریخی می شود پس این پارامتر را برابر با ۳ قرار می دهیم.

نکته: اگر این آرگومان عددی مثبت باشد، تاریخی که برگشت داده خواهد شد مربوط به آینده است و اگر این آرگومان، منفی باشد، تاریخی که نشان داده می شود مربوط به گذشته می باشد.

آرگومان `date` نیز تاریخی را مشخص می کند که عملیات باید بر روی آن انجام شود.

به عنوان مثال می خواهیم بدانیم که ۱ ماه بعد از ۳۱ دسامبر سال ۲۰۰۲، چه تاریخی می شود. پس به شکل زیر عمل می کنیم:

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

نتیجه این دستور، عبارت "1/31/2003 12:00:00 AM" خواهد بود. این تابع تا حد زیادی شبیه به تابع `DateSerial()` است که توضیح داده شد با این تفاوت که تعداد روزهای هر ماه در این تابع بنا به نام ماه فرق می کند مثلاً ممکن است یک ماه ۳۰ یا ۳۱ روز باشد اما تعداد روزهای ماه در تابع `DateSerial()`، ثابت و ۳۰ روز می باشد. مثال زیر را در نظر بگیرید:

```
day1 = #1/31/2002#
```

```
Console.WriteLine(DateSerial(year(day1),month(day1)+1,day(day1)))
```

نتیجه عبارت، "3/2/02" خواهد بود که یک تاریخ در ماه مارس است نه فوریه.

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

این تابع، شکل پیشرفته تر تابع DateAdd() است و اختلاف بین دو تاریخ را برمی گرداند. آرگومان interval یک مقدار ثابت است و فواصل بین دوره ای را مشخص می کند که شما برای محاسبه اختلاف بین دو تاریخ به کار می برید. مقادیری را که این آرگومان می تواند بپذیرد همان مقادیری هستند که آرگومان interval تابع DateAdd() می تواند بپذیرد. آرگومان های date1 و date2 تاریخ هایی هستند که اختلاف آنها باید برگشت داده شود. نکته: همیشه مقدار date2 از date1 کم می شود. آرگومان های اختیاری firstdayofweek و firstweekofyear نیز به ترتیب، اولین روز هفته و اولین هفته سال را تعیین می کنند. مقادیری را که این آرگومان ها می توانند بپذیرند، همان مقادیر آرگومان firstdayofweek تابع Format() است که پیشتر توضیح داده شد. (به توضیحات تابع format() مراجعه کنید) مثلاً شما می توانید از این تابع برای اطلاع از تعداد روزها، هفته ها، ثانیه ها و ... بین دو تاریخ استفاده کنید. مثال زیر تعداد روزهای بین یک تاریخ مشخص شده و تاریخ فعلی سیستم را نمایش می دهد:

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

نکته مهم: توابع DateDiff و TimeSpan در پروژه های وب و مخصوصاً فوروم هایی که به وسیله ی زبان ASP.NET ایجاد می شوند، کاربرد فراوانی دارند. یک نکته ی بسیار مهم: نکته ای که متأسفانه در اکثر پروژه های وب در ایران رعایت نمی شود این است که ساعت رسمی ایران در روز اول فروردین یک ساعت به جلو کشیده می شود و در این حالت، ساعت کشور ما از گرینویچ، ۴:۳۰ ساعت جلوتر است، در حالی که در روز ۳۱ شهریور یک ساعت به عقب کشیده می شود و به حالت اولیه خود باز می گردد یعنی ۳.۵+ GMT

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

این تابع، بخشی از یک تاریخ را برمی گرداند. آرگومان interval بخشی است که باید از تاریخ استخراج شود و همان مقادیری را می پذیرد که می توان برای قالب دهی به تاریخ و زمان در تابع Format() به کار برد. آرگومان date نیز تاریخی است که قسمتی از آن باید استخراج شود. آرگومان های اختیاری firstdayofweek و firstweekofyear نیز در تابع DateDiff() توضیح داده شدند. به عنوان مثال فرض می کنیم تاریخ جاری سیستم، ۲۳ اکتبر سال ۲۰۰۱ میلادی است. دستورات زیر نتایج مختلفی را به ما نشان می دهند که نتیجه به صورت پررنگ تر نمایش 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
```

## توابعی که جهت انجام عملیات ورودی/خروجی (I/O) بر روی فایل ها مورد استفاده قرار می گیرند (File I/O):

یکی از جنبه های مهم برنامه نویسی در هر زبان، قابلیت دسترسی و مدیریت فایل ها می باشد.

در ویژوال بیسیک، ۳ نوع فایل وجود دارد:

- ۱) فایل های ترتیبی.
- ۲) فایل های با دسترسی تصادفی.
- ۳) فایل های باینری.

فایل های ترتیبی اغلب فایل های متنی (Text) هستند. به عبارت دیگر شما می توانید اینگونه فایل ها را با هر ویرایشگر متنی همانند Notepad باز کنید.

در اینگونه فایل ها هر کاراکتر ذخیره شده در فایل، یک بایت را اشغال می کند.

تمامی اطلاعات ذخیره شده در فایل های متنی به صورت رشته ای ذخیره خواهند شد.

مثلا مقادیر عددی به صورت Integer یا Double ذخیره نمی شوند بلکه به صورت String عمل ذخیره سازی آنها انجام می شود.

عمل خواندن فایل های ترتیبی، به ترتیب یعنی از ابتدا تا انتهای فایل و خط به خط انجام می شود بنابراین شما نمی توانید در یک زمان در یک فایل ترتیبی هم اطلاعات را بخوانید و هم اطلاعات را بنویسید.

اگر قصد خواندن و نوشتن همزمان در و از یک فایل ترتیبی را دارید، باید از دو فایل ترتیبی استفاده کنید. یکی برای خواندن اطلاعات و یکی برای نوشتن اطلاعات درون فایل ترتیبی.

اگر در برنامه ای که می نویسید به طور مکرر با فایل ها سرو کار دارید به گونه ای که باید مرتباً محتویات یک فایل خوانده شود و پس از انجام تغییرات مجدداً ذخیره شود، باید از فایل های با دسترسی تصادفی استفاده کنید.

این فایل ها در مقایسه با فایل های ترتیبی از سرعت بیشتری برخوردارند.

فایل های تصادفی نیز همانند فایل های ترتیبی برای مقادیری از نوع رشته ای به ازای هر کاراکتر، یک بایت را اختصاص می دهند اما برخلاف فایل های ترتیبی، در این نوع فایل ها مقادیر عددی با نوع خاص خود ذخیره می شوند مثلاً Integer یا Double.

برای نمایش فایل های تصادفی در محیط Dos می توانید با نوشتن دستور TYPE در خط فرمان، مقادیر رشته ای ذخیره شده در این فایل ها را مشاهده کنید اما قادر به دیدن اعداد ذخیره شده نیستید.

فایل های تصادفی برای ذخیره کردن داده هایی که در بخش هایی با طول مشخص دسته بندی شده اند به کار می روند.

این بخش ها را اصطلاحاً **رکورد** می نامیم.

فایل های تصادفی به شما این امکان را می دهند که به هر رکورد دلخواه در فایل به سرعت دسترسی داشته باشید.

از آنجایی که تمامی رکوردها در فایل تصادفی دارای طول ثابتی هستند، اضافه کردن رکورد جدید در فایل با اختصاص دادن یک شماره شاخص (index) به این رکورد بسیار آسان است.

فایل های با دسترسی تصادفی برخلاف فایل های ترتیبی می توانند در یک زمان، هم فایل را بخوانند و هم در آن بنویسند.

در فایل های تصادفی اطلاعات را می توان در هر جای فایل نوشت بدون آنکه همجواری رکوردها از بین برود یا لطمه ای بخورد.

**فایل های باینری:** اینگونه فایل ها شبیه فایل های ترتیبی هستند.

محتویات این فایل ها می تواند هر چیزی باشد.

کاراکتر، عدد یا حتی تصاویر می توانند در فایل های باینری ذخیره شوند.

در این نوع فایل ها، داده ها به صورت باینری ذخیره می شوند.

مثلاً عدد ۴۲۶ عددی از نوع صحیح است و چون اعداد صحیح دو بایت از حافظه را اشغال می کنند پس میزان فضای اختصاص یافته به این عدد در این نوع فایل، ۲ بایت است.

انجام عملیات بر روی فایل ها بدون در نظر گرفتن نوع آنها، به طور کلی شامل ۳ مرحله می باشد:

**بازکردن فایل:** در این مرحله، سیستم عامل میزان معینی از حافظه را برای نگهداری داده های فایل کنار می گذارد. اگر فایل وجود نداشته باشد، ابتدا ساخته شده و سپس باز می شود.

برای بازکردن یا در صورت لزوم، ایجاد یک فایل از تابع `FileOpen()` استفاده کنید. **پردازش فایل:** یک فایل می تواند برای خواندن، نوشتن و یا خواندن-نوشتن همزمان داده هایش باز شود.

داده هایی که خوانده می شوند، پردازش شده و سپس در همان جایی قبلی یا در فایل دیگری ذخیره می شوند.

**بستن فایل:** هنگامی که کار با فایل به اتمام رسید باید آن را بست. هنگامی که فایل بسته شد، سیستم عامل فضای اختصاص داده شده به فایل را به حافظه اصلی باز می گرداند. برای بستن یک فایل باز از تابع `FileClose()` استفاده کنید.

### FreeFile(file\_number)

هر فایلی که باز می شود باید شماره ای به آن اختصاص یابد. اگر برنامه ی شما در طول دوره کاری خود تعداد زیادی فایل را باز یا بسته می کند، ممکن نیست که شما همیشه شماره فایل یا فایل هایی را که هم اکنون باز هستند بدانید. در VB، تابع `FreeFile()` شماره فایلی بعدی را که قرار است باز شود تعیین می کند. این تابع همیشه با تابع `FileOpen()` جهت بازکردن یک فایل به کار می رود. نکته: تابع `FreeFile()` را نمی توان به طور مستقیم در تابع `OpenFile()` به کار برد. دستور زیر را در نظر بگیرید:

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

این دستور از نظر نحوی درست کار می کند اما شماره فایل را در اختیارات قرار نمی دهد تا بتوانید اعمال خواندن یا نوشتن را بر روی فایل انجام دهید.

دستور زیر را در نظر بگیرید:

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

بعد از اجرای این دو دستور، هر فرمانی که با فایل سروکار داشته باشد می تواند با آوردن نام `fNum` در آرگومان مربوطه ی خود، به این فایل دسترسی داشته باشد. همان طور که گفته شد این تابع، شماره فایل بعدی را تعیین می کند مگر اینکه این شماره به فایلی اختصاص داده شده باشد. در اینصورت در صورت فراخوانی مجدد این تابع، عددی مشابه فراخوانی قبلی برگشت داده می شود. به عنوان مثال دستورات زیر کار نمی کنند:

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

دستور دوم اشتباه است چون هر وقت که تابع `FreeFile()` برای تعیین شماره ی یک فایل جدید فراخوانی می شود باید مورد استفاده قرار بگیرد و نمی توان قبل از به کارگیری آن دوباره از همین تابع جهت ایجاد یک شماره جدید استفاده کرد. پس دستورات فوق را به شکل زیر اصلاح می کنیم:

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

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

برای استفاده از يك فايل، بايد ابتدا آن را باز يا در صورت نیاز آن را ایجاد کنید. تابع FileOpen() با دریافت مقاديري این کار را برای شما انجام مي دهد. این تابع، جانشين دستور Open در VB.6.0 شده است. توجه داشته باشید که دستور Open در VB.NET قابل استفاده نیست. آرگومان path مسیر و نام فایلي است که باید باز شود و آرگومان number شماره اي است که شما به این فايل اختصاص مي دهید (معمولا با تابع FreeFile()). آرگومان mode، نوع عملیاتي را که قرار است بر روي فايل انجام شود تعیین مي کند و مي تواند يکي از مقادير زیر باشد:

| مقادير پارامتر Mode در تابع FileOpen()                                                                                                                                                                                                                                                                                                                             |        |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------|
| عملکرد                                                                                                                                                                                                                                                                                                                                                             | مقدار  |
| این مقدار برای بازکردن فايل ترتيبی که وجود دارد به کار مي رود. اگر فايل را با این مقدار باز کنیم فقط مي توانيم اطلاعات را از آن بخوانيم. چنانچه فایلي که قبلا بر روي دیسک وجود نداشته و بخواهيم با مد Input باز کنیم، خطايي اتفاق مي افتد.                                                                                                                         | Input  |
| این مقدار، فايل ترتيبی را طوري باز مي کند که بتوان اطلاعاتي را در آن نوشت. اگر فایلي که با مد Output باز مي شود وجود نداشته باشد، آن را ایجاد مي کند ولي اگر وجود داشته باشد، اطلاعات قبلي آن را از بین مي برد و آن را برای دریافت اطلاعات جديد آماده مي کند. به طور کلي این مقدار فايل را در حالت خروجي باز مي کند يعني فقط مي توان اطلاعات را به فايل اضافه کرد. | Output |
| فايل ترتيبی که قبلا وجود داشته باشد و بخواهيم اطلاعاتي را به آن اضافه کنیم، باید از این مقدار استفاده کنیم. چنانچه فايل قبلا وجود نداشته باشد، آن را ایجاد مي کند.                                                                                                                                                                                                 | Append |
| این مقدار، فايل را به صورت باینري باز مي کند. در حالت باینري مي توان به هر يك از بایت هاي فايل دسترسی داشت و مي توان يك بایت را از فايل خواند یا در فايل نوشت.                                                                                                                                                                                                     | Binary |
| این مقدار، يك فايل را به طور تصادفي باز مي کند. در این حالت مي توان در فايل نوشت یا از فايل خواند.                                                                                                                                                                                                                                                                 | Random |

سه مقدار اول با فايل هاي ترتيبی، مقدار Random با فايل هاي با دسترسی تصادفي و مقدار Binary برای فايل هاي باینري به کار مي روند. آرگومان اختیاري access نوع دسترسی به فايل را تعیین مي کند. يعني فايل برای خواندن یا نوشتن یا هر دو عمل باز شود. مقاديري که این آرگومان مي تواند بپذیرد در جدول زیر آمده است:

| مقادير پارامتر access در تابع FileOpen()                                                            |           |
|-----------------------------------------------------------------------------------------------------|-----------|
| عملکرد                                                                                              | مقدار     |
| فايل را در حالت ورودی باز مي کند. يعني فقط مي توان فايل را خواند.                                   | Read      |
| فايل را در حالت خروجي باز مي کند. يعني فقط مي توان در فايل نوشت.                                    | Write     |
| فايل را در حالت ورودی-خروجي باز مي کند. يعني هم مي توان اطلاعات را از فايل خواند و هم در فايل نوشت. | ReadWrite |

این آرگومان بیشتر برای بالا بردن امنیت کار با فايل به کار مي رود و ضرورت چنداني در استفاده از آن وجود ندارد. مثلا فايل هاي ترتيبی مي توانند برای فقط خواندن یا فقط نوشتن باز شوند نه هر دو. پس استفاده از آرگومان ReadWrite این آرگومان در هنگام باز کردن يك فايل ترتيبی بي اثر است.

ولي جهت بالا بردن ضريب امنيت استفاده از فايل، هنگامی که فايل را در هر حالي باز می کنید، با توجه به نوع فايل استفاده از این آرگومان نیز ضرري ندارد. آرگومان اختیاري share، نحوه عکس العمل فایلی که توسط برنامه شما باز شده است را به برنامه های دیگر ویندوز، در همان زمان که شما فايل را باز کرده اید مشخص می کند. به عبارت دیگر شما می توانید نحوه دسترسی برنامه های دیگر را به فایلی که شما باز کرده و در حال استفاده از آن هستید تعیین کنید. آرگومان share می تواند یکی از مقادیر زیر را بپذیرد:

| مقادیر پارامتر share در تابع FileOpen()                                                             |               |
|-----------------------------------------------------------------------------------------------------|---------------|
| عملکرد                                                                                              | مقدار         |
| برنامه های دیگر می توانند از فايل استفاده کنند.                                                     | Shared        |
| برنامه های دیگر فقط در صورتی مجاز به استفاده از فايل هستند که آن را فقط برای خواندن باز کرده باشند. | LockRead      |
| برنامه های دیگر فقط در صورتی مجاز به استفاده از فايل هستند که آن را فقط برای نوشتن باز کرده باشند.  | LockWrite     |
| برنامه های دیگر مجاز به استفاده از فايل نیستند                                                      | LockReadWrite |

آرگومان share يك پارامتر بسیار مهم برای برنامه هایی است که تحت شبکه نوشته می شوند. به عنوان مثال دو کاربر به طور همزمان قصد نوشتن در يك فايل را دارند. و اما آخرین آرگومان یعنی آرگومان اختیاري recordlen که فقط در فايل های با دسترسی تصادفی کاربرد دارد. اگر فایلی که قصد بازکردن آن را دارید از این نوع باشد باید طول هر رکورد فايل را که بر حسب بایت است به وسیله این آرگومان مشخص کنید. هنگامی که در ویژوال بیسیک فایلی با دسترسی تصادفی ایجاد می کنید، هیچ اطلاعاتی راجع به طول رکورد یا ساختار آن در فايل ذخیره نمی شود. این مورد را قبل از بازکردن این نوع فايل ها باید بدانید. طول رکورد، مجموع بایت های تمامی فیلدهای آن فايل است که می توانید با دستور Len(record) آن را محاسبه کنید. آرگومان record در این دستور نام ساختاری است که برای بازکردن فايل ترتیبی به کار برده اید.

دستور زیر فايل C:\samples\vb\cust.dat را که يك فايل ترتیبی است با اختصاص دادن عددی که با استفاده از تابع FreeFile() به دست آورده، در حالت ورودی-خروجی باز می کند:

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

### FileClose(file\_number)

پس از اینکه کار با فايل تمام شد، باید آن را بست. تابع FileClose() فایلی را که شماره ی آن به وسیله ی آرگومان file\_number مشخص شده می بندد.

دستور زیر، فایلی را که شماره ی آن در عبارت fNum1 وجود دارد می بندد.

```
FileClose(fNum1)
```

### Reset()

این تابع در حقیقت کار شما را در بستن فايل ها بسیار آسانتر می کند. این تابع تمامی فايل هایی را که با استفاده از تابع FileOpen() در برنامه شما باز شده اند می بندد.

## EOF(file\_number) , LOF(file\_number)

این دو تابع معمولاً در توابعی که با فایل‌ها سر و کار دارند بسیار به کار می‌روند. تابع EOF() برای تست شرط پایان فایل به کار می‌رود. این تابع فایلی را که شماره آن به وسیله آرگومان file\_number مشخص شده چک می‌کند و چنانچه اشاره گر فایل به انتهای آن رسیده باشد، این تابع مقدار True وگرنه مقدار False را برمی‌گرداند. تابع LOF()، اندازه یک فایل باز را که شماره آن به وسیله آرگومان file\_number مشخص شده است بر حسب بایت برمی‌گرداند.

شبهه کد زیر با استفاده از تابع EOF() و یک حلقه تکرار، پایان یک فایل را چک می‌کند:

```
{رفتن به اولین رکورد فایل}
While Not EOF(fNum)
{پردازش رکورد جاری}
{رفتن به رکورد بعد}
End While
```

با استفاده از تابع LOF() قادر به محاسبه ی تعداد رکورد های یک فایل با دسترسی تصادفی هستید. دستور زیر این کار را برای ما انجام می‌دهد:

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

## Print (file\_number,output\_list) , PrintLine(file\_number,output\_list)

برای نوشتن در یک فایل ترتیبی که به صورت Output یا Append باز شده است از تابع Print() استفاده می‌کنیم.

آرگومان file\_number، شماره فایلی است که قبلاً با یکی از دو حالت بالا باز شده است و آرگومان output\_list نیز یک آرایه ی پارامتری است، یعنی مقادیر یا متغیرهای مقداردهی شده ای که باید در فایل نوشته شوند.

**نکته:** منظور از آرایه ی پارامتری در VB، آرگومانی است که می‌تواند مقادیر نامحدودی بپذیرد. این مقادیر با استفاده از کاراکتر (,) از هم جدا می‌شوند. دستور زیر را در نظر بگیرید:

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

این دستور، ۴ مقدار شامل مقدار ۲ متغیر، یک مقدار رشته ای و یک مقدار عددی را در فایلی که شماره آن در عبارت fNum وجود دارد ذخیره می‌کند.

تابع Print() داده ها را در خطوط مجزا ذخیره نمی‌کند یعنی اینطور نیست که با هر بار فراخوانی این تابع، داده ها در خط بعدی فایل ذخیره شوند بلکه پشت سر هم و بدون فاصله قرار خواهند گرفت.

به عنوان مثال، دستور زیر فقط یک خط متنی در فایلی که شماره آن در fNum وجود دارد می‌نویسد:

```
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 کاری را انجام می‌دهد که تابع Print() انجام می‌دهد با این تفاوت که با هر بار فراخوانی، داده را در خطوط مجزا قرار می‌دهد همانند دستور WriteLine در زبان پاسکال که اطلاعات را در خطوط جدا، ذخیره می‌کند یا نمایش می‌دهد.

برای خواندن داده هایی که با تابع Print() در فایل ذخیره شده اند از توابع LineInput() و Input() استفاده می‌کنیم.

تابع Print() به طور پیش فرض بین فیلدهای مختلف در یک فایل فاصله ای قرار نمی‌دهد. برای قالب دهی به فیلدهایی که در هر خط فایل متنی ذخیره می‌شوند می‌توانید از کلمه کلیدی (Tab) استفاده کنید تا VB در هنگام ذخیره داده ها در فایل، بین فیلدهای مختلف فاصله قرار دهد.

همچنین خود می‌توانید برای این فاصله مقداری تعیین کنید. مثلاً: Tab(12) مثال زیر یک فایل متنی را ساخته و مقداری داده را در آن ذخیره می‌کند:

```

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 زیر 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)

بدیهی است که پس از نوشتن در فایل ترتیبی، باید بتوان اطلاعات را از آن خواند و پردازش کرد. برای خواندن اطلاعات از فایل ترتیبی، باید آن را به صورت Input باز کرد. تابع Input() داده های یک فایل ترتیبی را خوانده و آنها را در متغیری که به عنوان دومین آرگومان این تابع وجود دارد یعنی var می ریزد. آرگومان file\_number شماره فایلی است که به صورت Input باز شده است و آرگومان var، یک آرایه پارامتری است که شامل اسامی فیلدها یا لیستی از متغیرها است که اطلاعات خوانده شده در آنها قرار می گیرد. در این دستور، اگر هنگام خواندن فایل تعداد فیلدهایی که شما نام آنها را در آرگومان var می آورید از تعداد فیلدهای فایل کمتر باشد، فیلدهای بعدی در فراخوانی بعدی فرمان Input() نمایش داده می شوند. ولی چنانچه تعداد فیلدهایی که نام آنها در آرگومان var می نویسید، بیشتر از تعداد فیلدهای یک رکورد فایل باشد، به تعداد لازم از رکوردهای بعدی خوانده می شوند. به عنوان مثال، دستورات زیر یک مقدار عددی و یک مقدار از نوع تاریخ را از فایلی که باز شده است می خوانند:

```

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

```

### LineInput(file\_number)

پیشنهاد می کنم برای خواندن فایل ترتیبی از این تابع استفاده کنید. آرگومان file\_number شماره فایل باز شده است و این تابع یک سطر از فایل ترتیبی را برگشت می دهد. این تابع، در اولین بار فراخوانی، اولین خط فایل ترتیبی را برگشت می دهد و در فراخوانی بعدی دومین خط و به همین منوال. دستورات زیر سطرهای اول و دوم یک فایل ترتیبی را در متغیرهای Line1 و Line2 قرار می دهند.

```

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

```

برای ذخیره سازی متن های ساده در یک فایل، یک فایل ترتیبی درست کنید و داده های خود را در آن ذخیره کرده و در موقع لزوم برای خواندن خط به خط داده های ذخیره شده با استفاده از تابع `LineInput()` یا برای خواندن کل فایل از تابع `FileGet()` استفاده کنید.

**`FilePut(file_number,value [,record_number ])` ,  
`FileGet(file_number,value [,record_number ])`**

این دو تابع برای خواندن یا نوشتن رکوردها به یا از فایل با دسترسی تصادفی به کار می روند. تابع `FilePut()` برای نوشتن رکورد و تابع `FileGet()` برای خواندن رکورد استفاده می شوند. برای کار با این توابع نیاز به دانستن شماره رکوردی دارید که می خواهید آن را بخوانید یا در فایل بنویسید.

آرگومان `file_number`، شماره فایل است. آرگومان `record_number`، شماره رکوردی است که باید خوانده یا نوشته شود. آرگومان `record_number` یک پارامتر اختیاری است و چنانچه از ذکر آن خودداری شود، رکورد در مکان فعلی اشاره گر فایل نوشته می شود. (در تابع `FilePut()`) آرگومان `value`، متغیر رکوردی است که در فایل نوشته شده است یا می شود. پس از اینکه یک رکورد به فایل نوشته یا از فایل خوانده شد، رکورد بعد از آن به عنوان رکورد جاری در نظر گرفته می شود. به عنوان مثال اگر تابع `FilePut()`، ۱۰ بار پشت سر هم و بدون تعیین کردن شماره رکورد فراخوانی شود، ۱۰ رکورد اول فایل با دسترسی تصادفی، ایجاد یا بازنویسی می شود.

از آنجایی که فایل با ساختار دسترسی تصادفی، انعطاف پذیرترین ساختار و بیشترین کاربرد را دارد، با ذکر چند مثال با نحوه عملکرد این ساختار بیشتر آشنا می شویم: در اینجا من قصد ایجاد یک فایل نوع دسترسی تصادفی را جهت نگهداری مشخصات محصول یک شرکت دارم. هر مشخصه از این محصول در یک متغیر به نام `ProductRecord` قرار خواهد گرفت. ساختار `ProductRecord` را به شکل زیر تعریف می کنم: توجه: در VB.NET از کلمه کلیدی `Structure` جهت تعریف یک ساختار سفارشی استفاده می کنیم. این کلمه جایگزین کلمه کلیدی `Type` در VB.6.0 شده است.

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

اطلاعات کالا قبل از ذخیره شدن در فایل، در هر یک از این ۳ متغیر قرار خواهند گرفت. حالا برای دسترسی به فیلدهای این ساختار باید متغیری از نوع این ساختار یعنی `ProductRecord` تعریف کنیم:

```
Dim PRec As ProductRecord
```

حال می توانیم مقادیر خود را به فیلدهای متغیر `PreC` نسبت دهیم:

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

اکنون همه چیز برای ذخیره کردن مقادیر وارد شده در فایل ما آماده است اما قبل از انجام این کار باید فایل را ایجاد کنیم:

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

توجه داشته باشید که من در تابع `FileOpen()` از ذکر متغیر آخر که طول رکورد را تعیین می کند خودداری کردم. از آنجائیکه فیلدهای ما دارای مقادیر رشته ای هستند و طول آنها از قبل

مشخص نیست (متغیر است)، اجازه می‌دهیم که تابع به طور خودکار رکوردها را اداره و طول آنها را مدیریت کند. اصول کار به این شکل است که هنگام ذخیره متغیر رشته‌ای، طول آن نیز همراه با مقدار آن در فایل ذخیره می‌شود پس شما نیازی به دانستن طول هر رکورد ندارید. و حالا مقادیر نسبت داده شده را با دستور زیر در فایل ذخیره می‌کنیم:

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

توجه کنید که شما می‌توانید از ذکر آخرین آرگومان تابع FilePut() که شماره‌ی رکوردی که باید نوشته شود را مشخص می‌کند، خودداری کنید. پس از ذخیره تمامی مقادیر در فایل باید فایل را بست. پس با دستور زیر این کار را می‌کنیم:

```
FileClose(fNum)
```

برای خواندن رکوردها، همان تابع FileOpen() که برای ذخیره سازی رکوردها استفاده کردیم را به کار می‌بریم:

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

پس از باز کردن فایل جهت خواندن محتویات آن، می‌توانیم با یک ساختار تکرار (loop) این کار را انجام دهیم. مثال زیر نشان می‌دهد که چگونه رکوردهایی با طول متفاوت و متغیر را با استفاده از تابع FilePut() در یک فایل با دسترسی تصادفی ذخیره و سپس آنها را با استفاده از تابع FileGet() بخوانیم. ابتدا کد زیر را که ساختار رکورد ما را تعیین می‌کند در قسمت تعاریف فرم (declarations) بنویسید.

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

حال کدهای زیر را در رویداد Click() یک دکمه‌ی فرم قرار دهید.

```
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 که مشاهده می کنید، فیلدهای ProductID و Description محصولات مختلف، که از نوع رشته ای هستند دارای طول های مختلف هستند. سه قسمت اول کد بالا، ۳ رکورد مختلف را در فایل تصادفی ذخیره کرده و سپس فایل را می بندد. قسمت آخر کد نیز، دومین رکورد فایل را خوانده و مقادیر آن را در خروجی نمایش می دهد. نکته: در VB.6.0، هر رکورد در فایل تصادفی باید دارای طول ثابت می بود اما در VB.NET، می توانید رکوردهایی ایجاد و در فایل تصادفی قرار دهید که دارای طول متغیر باشند.

### Write(file\_number,output\_list) , WriteLine(file\_number,output\_list)

تابع Write() جهت ذخیره داده ها در یک فایل ترتیبی به کار می رود. آرگومان file\_number شماره فایلی است که داده ها در آن ذخیره خواهند شد. آرگومان output\_list نیز که یک آرایه ی پارامتری است، لیستی از داده هایی است که می خواهیم در فایل ترتیبی نوشته شوند و می توانند متغیرهای مقداردهی شده یا خود مقدار مورد نظر باشند. داده ها در آرگومان output\_list، با کاراکتر (,) از یکدیگر جدا می شوند. داده هایی که با تابع Write() در فایل نوشته می شوند معمولاً با تابع Input() خوانده می شوند. دستورات زیر، یک مقدار عددی و یک مقدار از نوع تاریخ را در یک فایل ترتیبی می نویسند:

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

تابع WriteLine() نیز همان کار تابع Write() را انجام می دهد با این تفاوت که با هر بار فراخوانی این تابع، داده ها در سطر جدیدی از فایل نوشته می شوند. **نکته مهم:** تفاوت توابع Print() و Write() و WriteLine() در این است که توابع Write() یا WriteLine() باعث می شود فیلدها در هنگام نوشته شدن در فایل ترتیبی با کاما (,) از یکدیگر جدا شوند ولی در تابع Print() چنانچه فیلدها با کاما از یکدیگر جدا شوند، در هنگام ذخیره در فایل، بین فیلدها ۱۴ فاصله قرار خواهد گرفت اما اگر فیلدها با کاراکتر (;) از یکدیگر جدا شوند، فیلدها بدون فاصله از یکدیگر در فایل قرار می گیرند.

مثال زیر شبیه مثالی است که در توضیحات تابع Print() آمد اما این بار با استفاده از تابع WriteLine() آن را می نویسیم:

```
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 ")
```

خروجی دستورات فوق به شکل زیر 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() مکان جاری اشاره گر فایل را برگشت می دهد. تابع Seek() نیز در صورتی که بدون ذکر آرگومان اختیاری position به کار رود، عملی مشابه تابع Loc() را انجام می دهد. در فایل های با دسترسی تصادفی، مقدار برگشتی این دو تابع، شماره آخرین رکوردی است که از فایل خوانده شده یا در فایل نوشته شده. در فایل های ترتیبی، این مقدار، با تقسیم بایت خوانده شده جاری بر عدد ۱۲۸ به دست می آید. در فایل های باینری نیز، شماره آخرین بایت خوانده شده از یا نوشته شده به فایل به عنوان مقدار برگشتی خواهد بود. شما قادر به تعیین مکان شروع خواندن یا مکان نوشته شدن رکورد در فایل با مقداردهی به آرگومان position از تابع Seek() هستید. مثلاً برای آنکه اشاره گر رکورد را در یک فایل با دسترسی تصادفی به سومین رکورد ببرید یا به عبارت دیگر آن را به عنوان رکورد فعال منظور کنید، می توانید از دستور زیر استفاده کنید:

Seek(fNum,3)

## Lock(file\_number [,fromRecord ],[toRecord ]), Unlock(file\_number [,fromRecord ],[toRecord ])

تابع Lock() این امکان را برای شما فراهم می کند که یک فایل با دسترسی تصادفی یا تعدادی از رکوردهای آن را قفل کنید. آرگومان file\_number شماره فایل را مشخص می کند. توجه: آرگومان های اختیاری fromRecord و toRecord را زمانی به کار ببرید که قصد قفل کردن تعدادی از رکوردهای فایل را دارید نه همه رکوردها را. با استفاده از آرگومان های fromRecord و toRecord قادر به تعریف محدوده ای جهت قفل کردن و به انحصار در آوردن رکوردها هستید. آرگومان fromRecord تعیین می کند که قفل گذاری از کدام رکورد شروع شده و آرگومان toRecord نیز تعیین می کند که قفل گذاری تا کدام رکورد ادامه پیدا کند. مثلاً می توانید تعیین کنید که رکوردهای ۵ تا ۱۰ در فایل، قفل شوند.

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

- ۱) در صورتی که از ذکر دو آرگومان اختیاری این تابع خودداری شود، تمام رکوردهای فایل قفل خواهند شد.
- ۲) در صورتی که فقط برای آرگومان fromRecord مقدار ذکر شود و آرگومان toRecord مقداردهی نشود، رکوردها از مقدار fromRecord تا به انتهای فایل قفل می شوند.
- ۳) رکوردهایی که با این تابع قفل می شوند، رکوردهای انحصاری هستند که فقط توسط برنامه شما قابل استفاده بوده و برنامه های دیگر در حال اجرا مجاز به استفاده از آنها نیستند.
- ۴) اگر برنامه ای دیگر در جهت دسترسی به فایل یا رکوردهایی که توسط شما قفل شده اند اقدام نماید، خطایی در برنامه اتفاق می افتد که باید توسط تعریف ساختار خطا در برنامه، آن را کنترل کرده و از پایان برنامه جلوگیری کنید.

تابع Unlock() نیز دقیقاً بالعکس تابع Lock عمل کرده و فایل یا رکوردهای قفل شده را آزاد می کند.

## Width(fNum,length)

این تابع نیز يك بسیار بسیار کارآمد است که جهت تعیین بیشترین طول خط یعنی تعداد کاراکترهای قابل ذخیره سازی در هر خط که می تواند در يك **فایل ترتیبی** نوشته شود، به کار می رود. بیشترین طول خط به وسیله ی آرگومان length تعیین می شود. در صورتی که کاراکترهای شما از تعداد تعیین شده در آرگومان length بیشتر باشد، کاراکترهای اضافی به طور اتوماتیک به خط بعد منتقل می شوند. این تابع را می توانید با توابع Print() و Write() (که داده ها را در يك خط و پشت سر هم ذخیره می کنند) جهت تعیین تعداد کاراکترهای ذخیره شونده در هر خط، به کار ببرید. همچنین می توانید تابع Width() را با توابع WriteLine() و WriteLine() نیز (که داده ها را در خطوط جدا ذخیره می کنند) به کار ببرید.

## FileAttr(file\_number)

این تابع، يك عدد از نوع Integer را برگشت می دهد که مشخص کننده ی حالت باز شدن يك فایل باز است. مقادیری را که این تابع برگشت می دهد، در جدول زیر مشاهده می کنید:

| مقادیری که تابع FileAttr() برگشت می دهد |              |
|-----------------------------------------|--------------|
| مد باز شدن                              | مقدار برگشتی |
| Input                                   | 1            |
| Output                                  | 2            |
| Random                                  | 4            |
| Append                                  | 8            |
| Binary                                  | 32           |

## توابعي که براي ايجاد اعداد تصادفي به کار مي روند (Random Numbers):

VB.NET دو تابعي که در VB.6.0 براي درست کردن اعداد تصادفي به کار مي روند (Rnd و Randomize) را پشتيباني مي کند اما از آنجا که در VB.NET، اکثر توابع، دسته بندي و در کلاس خاص خودشان قرار داده شده اند، اين دو تابع نيز در کلاس System.Random قرار داده شده اند.

### Rnd([seed ])

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

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

اگر مقداري در آرگومان seed ذکر شود، عدد تصادفي که ايجاد خواهد شد، بين صفر تا آن عددي است که در آرگومان seed ذکر شده. گاهي اوقات نياز داريد تا بين محدوده اي دلخواه مثلاً: 5 تا 10، عددي تصادفي ايجاد کنيد. بدین منظور مي توانيد به شکل زير عمل کنيد:

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

در اين دستور، متغير upper، حد بالاي محدوده ي ما (مثلاً 10) و متغير lower، حد پايين محدوده (مثلاً 5) است.

نکته: در کد بالا، جمع کردن نتيجه قسمت اول با عدد 1، از توليد عدد تصادفي بين صفر و يك جلوگيري مي کند. پيشنهاده مي کنم هميشه در توليد اعداد تصادفي، مقدار seed را با عدد يك جمع کنيد.

توجه: تابع Next نيز که در کلاس 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 کار مي رود و معمولاً قبل از آن نوشته مي شود. آرگومان اختياري seed، يك مقدار عددي است که براي توليد هسته ي اعداد تصادفي به کار مي رود. بهتر است از ذکر اين مقدار خودداري کنيد.

## توابع گرافیکی (Graphics):

در این قسمت، دو تابع در VB.NET که برای تعریف رنگ از آنها استفاده می شود را توضیح می دهیم.  
توجه: توابع LoadPicture() و SavePicture() که در VB.6.0 وجود داشتند، در VB.NET وجود ندارند.

### QBColor(color)

این تابع، برای تولید تعدادی رنگ از پیش تعریف شده به کار می رود و مقداری از نوع Integer که معرف کد رنگ ایجاد شده است را برمی گرداند.  
آرگومان Color، کد رنگی است که باید تولید شود و عددی بین 0 تا 15 است.  
مقادیری را که این آرگومان می تواند بپذیرد، در جدول زیر آمده است:

| مقادیر پارامتر color در تابع QBColor() |       |
|----------------------------------------|-------|
| رنگ                                    | مقدار |
| سیاه                                   | 0     |
| آبی                                    | 1     |
| سبز                                    | 2     |
| لاجوردی                                | 3     |
| قرمز                                   | 4     |
| ارغوانی                                | 5     |
| زرد                                    | 6     |
| سفید                                   | 7     |
| خاکستری                                | 8     |
| آبی روشن                               | 9     |
| سبز روشن                               | 10    |
| لاجوردی روشن                           | 11    |
| قرمز روشن                              | 12    |
| ارغوانی روشن                           | 13    |
| زرد روشن                               | 14    |
| سفید روشن (با شدت بالا)                | 15    |

کاربرد این تابع بیشتر در سیستم هایی است که کارت گرافیک آنها نصب نشده یا تنها قادر به پشتیبانی از ۱۶ رنگ می باشند که در سیستم های قدیمی کاربرد بیشتری دارد.

### RGB(red,green,blue)

رنگ ها دارای ۴ مدل اصلی هستند.  
مدل، روش تعریف یک رنگ را مشخص می کند.  
مدلی که مانیتور و تلویزیون خانه ی ما جهت نمایش رنگ ها از آن استفاده می کند مدل RGB نام دارد. در این مدل، ۳ رنگ اصلی وجود دارد که بقیه رنگ ها از این ۳ رنگ مشتق می شوند.  
رنگ های این مدل عبارتند از: قرمز (Red)، سبز (Green)، آبی (Blue)  
نام این مدل از حروف اول این ۳ رنگ گرفته شده است.  
دامنه ی هر کدام از این ۳ رنگ، عددی بین 0 تا 255 است.  
به وسیله این تابع، قادر به تولید هر یک از رنگ های مدل RGB هستیم.  
تعداد رنگ های موجود در مدل RGB، ۱۶۷۷۷۲۱۶ رنگ است که از ضرب ۳ عدد (256\*256\*256) به دست می آید.  
مقدار برگشتی این تابع، عددی از نوع Long است که کد رنگ تولید شده را نشان می دهد.

در جدول زیر، کد تعدادی از رنگ های متداول را در مدل RGB، ملاحظه می کنید:

| کد معادل تعدادی از رنگ های پرکاربرد در مدل RGB |                |                  |                 |
|------------------------------------------------|----------------|------------------|-----------------|
| رنگ                                            | کد پارامتر red | کد پارامتر green | کد پارامتر blue |
| سیاه                                           | 0              | 0                | 0               |
| آبی                                            | 0              | 0                | 255             |
| سبز                                            | 0              | 255              | 0               |
| لاجوردی                                        | 0              | 255              | 255             |
| قرمز                                           | 255            | 0                | 0               |
| ارغوانی                                        | 255            | 0                | 255             |
| زرد                                            | 255            | 255              | 0               |
| سفید                                           | 255            | 255              | 255             |

به عنوان مثال، دستور زیر، یک رنگ قرمز خالص را ایجاد و به پس زمینه ی عنصر TextBox، نسبت می دهد:

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

فرمول تبدیل مقدار RGB به معادل Long:

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

## توابعي که با رجیستري سر و کار دارند (Registry):

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

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

این تابع، جهت ذخیره سازی يك مقدار دلخواه در رجیستري یا به هنگام کردن آن مقدار از طریق نوشتن آن بر روی مقدار قبلي، به کار می رود. از این تابع معمولاً برای ذخیره تنظیمات برنامه، استفاده می شود. مقادير در يك مسیر اصلي، يك شاخه، يك زیرشاخه و در نهایت به صورت يك مقدار رشته اي، در زیرشاخه ذخیره می شوند.

نکته: مسیر اصلي توسط VB تعیین می شود و نیازی به وارد کردن آن نیست. آرگومان appname، نام شاخه یا پوشه ي اصلي است که تمام زیرپوشه ها و مقادير آنها در این شاخه ذخیره می شوند. می توانید هر نام دلخواهي به این نام بدهید اما برای راحتی کار بهتر است نام برنامه ي خود را در این آرگومان ذکر کنید. آرگومان section نیز نام زیر شاخه اي در شاخه appname است که مقادير مربوط به هم در این زیرشاخه ذخیره می شوند. نام این آرگومان نیز دلخواه است. آرگومان key، نام مقدار و آرگومان setting نیز خود مقداري است که باید ذخیره شود. اگر مقدار setting به هر دليلي ذخیره نشود، يك خطاي زمان اجرا در برنامه اتفاق می افتد. به عنوان مثال فرض کنید که مقدار طول و عرض فرم برنامه ي ما، توسط کاربر در زمان اجرای برنامه تغییر پیدا کند و بخواهیم دفعه ي بعد که برنامه توسط کاربر اجرا می شود، فرم با طول و عرض جدید (یعنی همان انداز هاي تغییر پیدا کرده توسط کاربر) نمایش داده شود. دستورات زیر، مقدار طول و عرض تغییر یافته ي فرم را در رجیستري ذخیره می کنند:

```
SaveSetting("MyApp", "Startup", "Height", Me.Height)
SaveSetting("MyApp", "Startup", "Width", Me.Width)
```

دقت کنید که بنده جهت راحتی کار، مقدار ۳ پارامتر اول این دستورات را متناسب با مقداري که قرار است ذخیره شود انتخاب کردم. نکته: مناسب ترین مکان برای نوشتن این کدها، زمانی است که برنامه شما پایان می پذیرد و مناسبترین زمان جهت فراخواني آنها، زمانی است که برنامه آغاز می شود. (با استفاده از تابع GetSetting که در ادامه توضیح داده شده است)

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

این تابع دقیقاً عکس تابع SaveSetting() عمل کرده و يك مقدار از زیرشاخه یا کل زیرشاخه را از رجیستري حذف می کند. آرگومان هاي این تابع، همان آرگومان هاي تابع SaveSetting() هستند. اگر از ذکر آرگومان اختیاري key خودداری شود، تمامی مقادير موجود در زیرشاخه حذف خواهند شد. به عنوان مثال می خواهیم مقاديري را که در مثال تابع SaveSetting() در رجیستري ذخیره کردیم، حذف کنیم. پس دستورات زیر را می نویسیم:

```
DeleteSetting("MyApp", "Startup", "Height")
DeleteSetting("MyApp", "Startup", "Width")
```

می توانید این دستورات را در يك خط به شکل زیر خلاصه کنید:

```
DeleteSetting("MyApp", "Startup")
```

دقت کنید اگر مقادير دیگری نیز در زیرشاخه ي Startup وجود داشته باشند، آنها نیز حذف می شوند.

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

وقتي كه مقادير در رجیستري نوشته شد طبعاً باید آنها را خواند. این تابع، مقدار ذخیره شده ی کاربر را برمی گرداند. ۳ آرگومان اول این تابع همان پارامترهای توابع SaveSetting() و DeleteSetting() هستند كه مسیر ذخیره داده در رجیستري را مشخص می کنند. آرگومان اختیاری default، مقداری پیش فرض است كه در صورت نبودن مقداری در مسیر ذكرشده، توسط تابع برگشت داده می شود. به عنوان مثال می خواهیم مقادیری را كه در مثال تابع SaveSetting() در رجیستري ذخیره كردیم، به دست آورده و طول و عرض جدید فرم را در هنگام اجرای برنامه تعیین كنیم:

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

در این دستورات حتماً باید مقدار آرگومان default را ذكر كنید چون در صورتی كه مقداری در رجیستري وجود نداشته باشد، فرم ممكن است با سایزی بسیار كوچك یا بسیار بزرگ ظاهر شود.

## GetAllSettings(appname,section)

این تابع، لیستی از تمامی مقادیر موجود در زیرشاخه تعیین شده را با قرار دادن آنها در يك آرایه ی دو بعدی برمی گرداند. آرگومان های appname و section قبلاً توضیح داده شدند. خانه ی (0,0) آرایه، نام اولین مقدار زیرشاخه و خانه ی (0,1) آرایه، مقدار آن را در خود نگه می دارد. خانه ی (1,0) آرایه، نام دومین مقدار زیرشاخه و خانه ی (1,1) آرایه، مقدار آن را نگهداری می کند و به همین ترتیب... نکته: برای آگاهی از تعداد مقادیر ذخیره شده در آرایه یا به عبارت بهتر تعداد مقادیر موجود در زیرشاخه، از خاصیت Length آرایه استفاده کنید. فرض می کنیم آرایه AllSettings به عنوان يك آرایه دو بعدی معرفی شده است. دستور زیر، تمامی مقادیر موجود در زیر شاخه ی Startup از شاخه MyApp را در آرایه AllSettings قرار می دهد.

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

حالا می توانیم با دستورات زیر، تمامی مقادیر موجود در زیرشاخه را كه در آرایه ذخیره شده اند را ببینیم:

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

## Shell(path\_name [,style ][,wait ][,timeout ])

با استفاده از این تابع، قادر به اجرای یک برنامه دیگر از درون برنامه ی خود هستید. در صورتی که این تابع کار خود را با موفقیت انجام دهد، هندل برنامه ی اجرا شده را که توسط سیستم عامل تعیین می شود برگشت می دهد در غیر اینصورت، مقدار صفر را برگشت می دهد.

آرگومان path\_name، مسیر و نام فایل است که می خواهیم اجرا شود. اگر فایل اجرایی نیاز به آرگومان هایی همانند command های مخصوص خود داشت، می توان نام آنها را نیز در ادامه مسیر فایل ذکر کرد که این مورد معمولاً ندرتا وجود دارد. نکته مهم: این تابع تنها فایل های اجرایی را اجرا می کند اما می توان با نسبت دادن نام و مسیر فایل دیگر در ادامه نام و مسیر فایل اصلی، در صورتی که این فایل با برنامه مورد نظر ما قابل اجرا باشد، آن را اجرا کرد مثلاً یک فایل با پسوند TXT را می توان با دستور زیر اجرا کرد:

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

این دستور، فایل MyFile.txt را به وسیله برنامه Notepad باز می کند. آرگومان اختیاری style، وضعیت فرم برنامه ای که اجرا خواهد شد را معین می کند و می تواند یکی از مقادیر زیر را بپذیرد:

| مقادیر پارامتر Style در تابع Shell()                                                                                                                        |                  |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|
| عملکرد                                                                                                                                                      | مقدار            |
| برنامه به صورت مخفی اجرا می شود اما به عنوان برنامه فعال در نظر گرفته می شود.                                                                               | Hide             |
| برنامه به صورت معمول و با اندازه خودش اجرا می شود و به عنوان برنامه فعال در نظر گرفته می شود.                                                               | NormalFocus      |
| برنامه به صورت مینیمایز اجرا خواهد شد و به عنوان برنامه فعال در نظر گرفته می شود.                                                                           | MinimizedFocus   |
| برنامه به صورت تمام صفحه اجرا می شود و به عنوان برنامه فعال در نظر گرفته می شود.                                                                            | MaximizedFocus   |
| برنامه به صورت معمول و با اندازه خودش اجرا می شود اما به عنوان برنامه فعال در نظر گرفته نمی شود و برنامه ای که به عنوان برنامه فعال است، فعال باقی می ماند. | NormalNoFocus    |
| برنامه به صورت مینیمایز اجرا می شود اما به عنوان برنامه فعال در نظر گرفته نمی شود و برنامه ای که به عنوان برنامه فعال است، فعال باقی می ماند.               | MinimizedNoFocus |

آرگومان اختیاری wait تعیین می کند آیا برنامه شما برای اجرای کامل برنامه ای که فراخوانی خواهد شد صبر کند یعنی دستورات بعد از تابع Shell()، پس از فراخوانی برنامه مورد نظر اجرا شوند یا بلافاصله.

به عبارت ساده تر، تعیین می کند که قبل از اجرای تمامی دستورات بعد از تابع Shell()، برنامه اجرا شود یا در همان زمان فراخوانی.

این پارامتر، مقدار True یا False را می پذیرد که به طور پیش فرض، False است یعنی برنامه ای که قرار است اجرا شود باید صبر کند تا تمامی دستورات بعد از تابع shell() اجرا شوند.

آخرین آرگومان اختیاری که timeout نام دارد، تعیین می کند پس از چند ثانیه برنامه ای که فراخوانی می شود باید اجرا شود.

این مقدار، عددی بر حسب میلی ثانیه است و هر ۱۰۰۰ میلی ثانیه، ۱ ثانیه است.

به طور پیش فرض مقدار این آرگومان، 1- است یعنی بدون وقفه و بلافاصله پس از فراخوانی اجرا می شود.  
به عنوان مثال، برای اجرای برنامه Notepad ویندوز از دستور زیر استفاده کنید:

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

توجه: برای اجرای برنامه هایی که در پوشه اصلی ویندوز هستند، نیازی به ذکر مسیر آنها نیست و تنها آوردن نام، کفایت می کند.  
نکته مهم: تابع SendKeys() در VB.NET وجود ندارد.

## AppActivate(title [,wait ])

این تابع (که مقدار برگشتی ندارد) به شما اجازه می دهد برنامه ای را که قبلاً توسط تابع Shell() فراخوانی و اجرا کرده اید، به عنوان برنامه فعال (پنجره فعال) سیستم عامل معرفی کنید.

در آرگومان title، که مقداری از نوع رشته ای است باید نام نوار عنوان برنامه ای که قصد فعال شدن آن را دارید ذکر کنید. مثلاً: "Untitled – Notepad"  
در این آرگومان همچنین با ذکر کردن مقداری که تابع Shell() در متغیر فراخوانی کننده خود قرار می دهد، قادر به فعال کردن برنامه هستید. مثلاً در آخرین مثال توضیحات تابع Shell()، می توانید نام متغیر NPAD را در آرگومان title این تابع ذکر کنید.  
آرگومان اختیاری wait تعیین می کند که آیا برنامه شما باید به عنوان برنامه فعال باشد تا بتواند برنامه دیگری را با این تابع فعال کند یا خیر.

این پارامتر، مقدار True یا False را می پذیرد. مقدار پیش فرض، False است یعنی حتی در صورت فعال نبودن برنامه شما نیز، برنامه درخواستی به سرعت فعال می شود.  
در صورتی که True باشد، برنامه شما باید صبر کند تا فعال شود یا اصطلاحاً فوکاس به آن انتقال پیدا کند تا بتواند برنامه دیگری را فعال کند.

توجه: این تابع به مینیمایز بودن یا اندازه ی پنجره ها کاری ندارد و مثلاً اگر برنامه ای مینیمایز باشد، آن را در همان حالت مینیمایز به عنوان برنامه فعال در نظر می گیرد.  
نکته مهم: همیشه سعی کنید ابتدا برنامه مورد نظر را با تابع Shell() اجرا و سپس متغیری که تابع Shell() را در آن قرار داده اید در آرگومان title این تابع قرار داده و اقدام به فعال کردن آن کنید. مثلاً فرض کنید که برنامه Notepad را ۲ بار اجرا کرده اید و عنوان نوار هر دو پنجره Notepad، یکی است. در این حالت اگر نام نوار عنوان را در تابع AppActivate() ذکر کنید، آن یکی که آخر اجرا شده است به عنوان پنجره فعال در نظر گرفته می شود که ممکن است دلخواه شما نباشد.

## دستورات اختیاری (Option Statements):

دستورات Option یا اختیاری، سهولت کار در کدنویسی را برای شما فراهم می کنند.  
این دستورات در ابتدای کد فرم یا ماژول نوشته می شوند و در تمام آن فرم یا ماژول قابل دسترسی هستند. این دستورات را در زیر مشاهده می کنید:

## Option Compare

این دستور تعیین می کند که VB چگونه عملیات مقایسه ای رشته ها را انجام دهد.  
مقایسه ی رشته ای می تواند به طور معمول یا با در نظر گرفتن بزرگ یا کوچک بودن حروف انجام شود.

مقایسه رشته ای می تواند به یکی از دو شکل زیر صورت پذیرد:  
**Option Compare Binary:** در این نوع مقایسه که به صورت باینری انجام می شود، کاراکترهای بزرگ بر کاراکترهای کوچک ارجعیت دارند و کاراکترهای Symbol یا نماد در انتها قرار می گیرند. مثلاً به شکل زیر:

$A < B < C \dots < Z < a < b < c \dots < z < A' < A^ < a' < a^$

**Option Compare Text:** در این نوع مقایسه، عمل مقایسه با توجه به بزرگ و کوچک بودن حروف انجام می شود. این نوع مقایسه، مقایسه پیش فرض رشته ها است. مثلاً:  
 $A=a < A'=a' < B=b$

### Option Explicit

همان طور که می دانید، در VB می توان یک متغیر را بدون تعریف کردن آن به کار برد. می توان با استفاده از این دستور به کامپایلر ویژوال بیسیک گفت که از تعریف نکردن متغیرها در طول فرم یا ماژول که ای دستور قرار می گیرد، جلوگیری کند. در این حالت، در زمان اجرای برنامه کامپایلر VB تمامی متغیرها را از لحاظ تعریف شدن چک می کند و در صورتی که هر یک از آنها تعریف نشده باشند، به شما اخطار می دهد. می توانید با فعال کردن این گزینه در سربرگ Build از قسمت Project Property Pages در VB.NET، به طور خودکار در تمامی فرم ها و ماژول های برنامه این دستور را فعال کنید. توجه: در صورتی که در VB.NET، نوع متغیر تعریف نشود به طور پیش فرض از نوع Object در نظر گرفته می شود. (در VB.6.0 از نوع Variant در نظر گرفته می شد)

### Option Strict

همان طور که اشاره شد در صورتی که نوع متغیری در VB.NET ذکر نشود، به طور پیش فرض از نوع Object در نظر گرفته می شود و در هنگام مقداردهی به آن متغیر، نوع آن به صورت خودکار توسط VB تعیین می شود. دستورات زیر را در نظر بگیرید:

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

در دستورات فوق ابتدا یک متغیر بدون نوع تعریف می شود که به طور پیش فرض از نوع Object در نظر گرفته می شود و سپس در هنگام مقداردهی به آن، VB به طور خودکار نوع مقدار نسبت داده شده را تشخیص و آن را تبدیل به آن نوع می کند که در مثال فوق از نوع String در نظر می گیرد. چنانچه مایل هستید که عمل تبدیل خودکار نوع متغیر تعریف نشده توسط VB انجام نشود و همان مقدار Object در نظر گرفته شود، دستور Option Strict را در اولین خط کد فرم بنویسید.

### توابع متفرقه (Miscellaneous):

توابعي هستند که نمی توان آنها را در هیچ گروهی دسته بندی کرد. این توابع در این قسمت مورد بحث قرار می گیرند.

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

این تابع، مقداری را از یک سری آرگومان، انتخاب و برگشت می دهد. آرگومان index، یک مقدار عددی بین ۱ تا تعداد آرگومان های Choice است که نمایانگر شماره آرگومانی است که می خواهیم برگشت داده شود. آرگومان های choice1، choice2 و...، لیستی از آرگومان هایی هستند که می خواهیم مقداری را از آنها انتخاب و برگشت دهیم. به عنوان مثال، اگر مقدار index، ۱ باشد مقدار برگشتی، مقدار آرگومان choice1 و اگر index، ۲ باشد، مقدار برگشتی تابع، مقدار آرگومان choice2 و به همین ترتیب خواهد بود. مثلاً یکی از کاربردهای این تابع برای برگشت معادل حرفی اعداد تک رقمی است. به عنوان مثال، تابع زیر که IntToString() نام دارد، نام معادل عددی که به آن داده می شود را برمی 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 زیر با استفاده از تابع بالا، مقدار حرفی معادل عدد ۸ را برگردانده و با استفاده از تابع MsgBox() نمایش می دهد:

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

توجه: در صورتی که مقدار آرگومان index از ۱ کمتر یا مقدار آن از تعداد آرگومان های choice بیشتر باشد، مقدار برگشتی این تابع یک مقدار Null خواهد بود.

#### IIf(expression,truepart,falsepart)

این تابع، شکل ساده تر استفاده از ساختار تصمیم گیری If Then Else است. معادل ساختار If Then Else این تابع به شکل زیر است:

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

آرگومان expression، یک عبارت شرطی است که درست یا غلط بودن آن باید چک شود. اگر عبارت شرطی موجود در expression، صحیح باشد، عبارت truepart وگرنه عبارت falsepart اجرا خواهد شد.

در مثال زیر، چنانچه مقدار A از B کوچکتر باشد، A در Min وگرنه B در Min قرار می گیرد:

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

دستور زیر معادل دستور فوق است:

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

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

این تابع، لیستی از عبارات ورودی را با یکدیگر مقایسه کرده و مقدار اولین عبارتی که صحیح باشد را برمی گرداند. در صورتی که هیچکدام از عبارات صحیح نباشند، این تابع يك مقدار Null را برگشت خواهد داد.

مثال زیر، مقادیر موجود در متغیرهای X و Y را چک کرده و متناسب با نوع عبارات شرطی تعریف شده، مقداری را برگشت می 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 ۱ در متغیر Test قرار خواهد گرفت. اگر X مثبت و Y منفی باشد، متغیر Test، مقدار ۲ را خواهد داشت و بقیه شروط به همین ترتیب.

توجه: اگر X و Y، صفر باشند، هیچکدام از عبارات صحیح نخواهند بود و مقدار Test، Null خواهد شد.

## Environ(expression)

این تابع جهت برگشت متغیرهای محلی به کار می رود. متغیرهای محلی، متغیرهای سیستم عامل هستند که با فرمان SET در آن قرار داده شده اند.

برای دسترسی به محتویات يك متغیر محلی در سیستم، از يك عدد یا نام آن در آرگومان expression استفاده کنید. دستو زیر را در نظر بگیرید:

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

به کمک دستور فوق، هم، نام متغیر محلی و هم مقدار آن در خروجی نمایش داده می شود. به عنوان مثال:

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

برای به دست آوردن تنها، مقدار متغیر محلی TMP، باید نام آن را در آرگومان expression ذکر کنید همانند زیر:

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

دستور فوق، عبارت زیر را نمایش خواهد داد:

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

مثلا دستور زیر، مسیر پوشه ویندوز شما را نمایش می دهد:

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

با استفاده از دستور فوق نیازی به استفاده از تابع API برای به دست آوردن مسیر نصب ویندوز ندارید.

برای به دست آوردن مقادیر متغیرهای محلی موجود در سیستم عامل خود، به ترتیب از عدد ۱ شروع کرده و این اعداد را در آرگومان expression قرار دهید تا زمانی که این تابع، يك مقدار خالی را برگشت دهد.

نکته: در صورتی که عدد یا نام متغیر محلی که ذکر می کنید وجود نداشته باشد، ای تابع يك مقدار خالی را برگشت می دهد.

این تابع جهت ایجاد يك صدای بیپ کوتاه به کار می رود. طول زمان پخش این صدای بیپ، قابل تنظیم نیست. استفاده از این تابع نیازی به نصب بودن یا داشتن کارت صدا ندارد.

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

این تابع جهت اجرای یکی از متدهای يك شی یا تعیین مقدار خواص شي یا برگرداندن مقدار یکی از خواص شي به کار می رود. آرگومان object، نام شي مورد نظر است. آرگومان procedurename، نام يك خواص یا متد شي است. آرگومان calltype، نوع عملی که می خواهیم انجام شود را مشخص می کند و یکی از مقادیر زیر را می پذیرد:

| مقادیر پارامتر calltype در تابع CallByName() |        |
|----------------------------------------------|--------|
| عملکرد                                       | مقدار  |
| مقدار يك خاصیت شي را می خواند                | Get    |
| مقدار يك خاصیت شي را تعیین می کند            | Set    |
| يك متد از شي را فراخوانی و اجرا می کند       | Method |

آخرین آرگومان این تابع که arguments نام دارد، يك آرایه از مقادیر است. اگر از مقدار Set استفاده کنیم، باید مقداری را که می خواهیم به یکی از خاصیت های شي نسبت دهیم در این آرگومان ذکر کنیم. اگر از مقدار Method استفاده می کنیم، ممکن است متد تابع ما برای انجام کار خود نیاز به پارامترها یا مقادیری داشته باشد، پس آنها را به ترتیب در این آرگومان یعنی arguments ذکر می کنیم. به عنوان مثال به طور معمول برای آنکه به خاصیت Text از شي TextBox، مقداری را نسبت دهیم، به شکل زیر عمل می کنیم:

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

حالا همین کار را با استفاده از تابع CallByName() انجام می دهیم:

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

توجه: اولین آرگومان که باید نام شي مورد نظر در آن ذکر شود، يك مقدار رشته ای نیست بلکه نام اصلی شي را بدون هیچ کاراکتر اضافی باید ذکر کرد. به عنوان مثال، دستور زیر اشتباه است و يك خطای زمان اجرا اتفاق خواهد افتاد:

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

در صورتی که می خواهید نام شي را در يك متغیر قرار دهید باید ابتدا آن را از نوع Object تعریف کنید. پس دستور فوق را به شکل زیر اصلاح می کنیم:

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

برای به دست آوردن مقدار خاصیت Text از شي TextBox نیز به شکل زیر عمل کنید:

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

| تاریخچه ویژال بیسیک |              |
|---------------------|--------------|
| Visual Basic 1.0    | ۲۰ می ۱۹۹۱   |
| Visual Basic 2.0    | مارس ۱۹۹۲    |
| Visual Basic 3.0    | ژوئن ۱۹۹۳    |
| Visual Basic 4.0    | اکتبر ۱۹۹۶   |
| Visual Basic 5.0    | آوریل ۱۹۹۷   |
| Visual Basic 6.0    | ۱۶ ژوئن ۱۹۹۸ |
| Visual Basic .NET   | ۲۰۰۱         |