

## الفصل العاشر المعاملات ودوال السلاسل الحرفية والتاريخ

كما أن العمليات الحسابية أمر هام في حياتنا اليومية، فإنها أمر حيوى بالنسبة للبرامج، وفي هذا الفصل ستتعرف على أساسيات إنجاز العمليات الحسابية والمنطقية، وستعرف أيضا كيفية معالجة سلاسل البيانات النصية والتاريخ والوقت. بانتهاء هذا الفصل ستتعرف على:

- ◆ المعاملات الحسابية
- ◆ معاملات ربط (وصل) العبارات
- ◆ المعاملات العلائقية
- ◆ المعاملات المنطقية
- ◆ أولويات تنفيذ المعاملات
- ◆ استخدام دوال السلاسل الحرفية
- ◆ التعامل مع التاريخ والوقت

## استخدام المعاملات Operators

يتيح Visual Basic لمبرمجيه إنجاز العديد من العمليات الحسابية والمنطقية والتعامل مع العبارات النصية والتي يصعب أن تجدها مجتمعه في لغات أخرى بنفس الإمكانيات والسهولة المصاحبة لكل عملية ومن أشهرها:

- العمليات الحسابية Arithmetic Operations
- عمليات ربط العبارات String Operations
- العمليات العلائقية Relational Operations
- العمليات المنطقية Logical Operations

وتستخدم كل عملية من العمليات السابقة عدة عوامل أو معاملات (Operators). فمثلا يلزم لإنجاز العمليات الحسابية عدة معاملات مثل معاملات الجمع "+" أو الطرح "-" أو الضرب "\*" والقسمة المشهورة "/". ستتعلم في هذا الفصل المعاملات التي تلزم لإنجاز العمليات الأخرى التي ذكرناها.

كلمة معامل أو عامل تقابل كلمة Operator التي يستخدمها Visual Basic وقد تجدها هنا في الشرح مرة عامل ومرة معامل وكلها بمعنى Operator.



وفيما يلي نوضح كيفية إنجاز كل عملية من هذه العمليات الأساسية وسنشرح أيضا استخدام معاملات كل مجموعة من هذه المجموعات.

## العمليات الحسابية Arithmetic Operations

كما أن العمليات الحسابية ضرورية في حياتنا اليومية فإن العمليات الحسابية أمر حيوي بالنسبة لتطوير البرامج. تستخدم لغة Visual Basic العمليات الحسابية الآتية:

### عملية الجمع Addition

تستخدم المعامل (الرمز) "+" (كلمة معامل أو رمز هنا تقابل كلمة Operator) وتكون بين قيمتين ثابتتين كما في هذا المثال:

Debug .Write Line(3.12 + 12)

وعند تنفيذ هذا الأمر ستحصل على الناتج 15.12 داخل نافذة Immediate Window بالجزء السفلى من نافذة بيئة التطوير (لإظهار النافذة الفورية Immediate Window إن لم تكن ظاهرة بالفعل، اضغط الاختصار Ctrl+ G بلوحة المفاتيح أو افتح قائمة Debug ثم اختر Immediate من القائمة الفرعية Windows).  
وقد تكون إحدى القيمتين متغير أو أحدهما ثابت والأخر متغير كما في المثال التالي (اكتب الأوامر التالية داخل أى إجراء):

```
Dim A = 5: Dim B = 6.5
Debug.WriteLine(A + B)
Debug.WriteLine(A + B + 2)
```

وعند التنفيذ سيكون الناتج كما يلي

```
11.5
13.5
```

وكما تلاحظ فإن عملية الجمع تتعامل مع قيم من جميع الأنواع.

### عملية الطرح Subtraction

تستخدم المعامل (الرمز) (-) وبنفس الطريقة السابقة تتعامل مع جميع القيم بجميع أنواعها، انظر المثال التالي والنتيجة التي ستحصل عليها بعد التنفيذ (اكتب الأوامر في أحد الإجراءات).

```
Dim A = 10: Dim B = 8.5
Debug.WriteLine(A - B)
Debug.WriteLine(B - A)
```

وعند التنفيذ سيكون الناتج كما يلي:

```
1.5
-1.5
```

وتدل العلامة السالبة (-) قبل القيمة من اليسار على أن القيمة الأولى (8.5) كانت أقل من القيمة الثانية (10).

### عملية الضرب والأس Multiplication / Exponentiation

يستخدم في عملية الضرب (Multiplication) المعامل "\*" ويستخدم في عملية الأس

**(Exponentiation) المعامل "^"** وتستخدم العمليتان بنفس الطريقة السابقة.

مثال (اكتب التعليمات التالية في أحد الإجراءات):

```
Debug.WriteLine(2*3)
Debug.WriteLine(2 ^ 3)
Debug.WriteLine(1.5 * 7)
Debug.WriteLine(3.4 ^ 1.29)
```

ستحصل على الناتج التالي (على الترتيب):

```
6
8
10.5
4.84850449955586
```

**عمليات القسمة والناتج الصحيح من القسمة وباقي القسمة**

يستخدم مع عملية القسمة (Division) الرمز "/" . انظر المثال التالي والنتيجة التي تحصل عليها.

مثال:

```
Debug.WriteLine( 8/2 )
Debug.WriteLine( 10/3)
```

الناتج هو:

```
4
3.33
```

كما يتيح لك Visual Basic عمليتين أخرتين الأولى هي عملية الناتج الصحيح من القسمة (Integer Division) ويرمز لها بالرمز "\" بدلاً من الرمز السابق "/" ويكون ناتج هذه العملية الجديدة هو الجزء الصحيح (دون تقريب) فقط.

مثال:

```
Debug.WriteLine( 14\5 )
Debug.WriteLine( 10\3)
```

الناتج هو:

```
2
3
```

أما العملية الثالثة فهي عملية باقي القسمة (Reminder) ويرمز لها بالرمز (mod) ويكون

ناتج هذه العملية هو الجزء المتبقى من القسمة فقط.

مثال:

```
Debug.WriteLine( 14\5 )  
Debug.WriteLine( 14 Mod 5)
```

الناتج هو:

2  
4

حيث أن ناتج قسمة الرقم (14) على الرقم (5) هو العدد الصحيح (2) ويتبقى القيمة (4).

ومن السمات الجديدة في Visual Basic 2012 استخدام الصيغ المختصرة في العمليات الحسابية. يوضح جدول ١٠-١ التالي الصيغة المختصرة للعمليات الحسابية المختلفة.

جدول ١٠-١ الصيغ الحسابية المختصرة

الصيغة المختصرة	الصيغة الطويلة	العملية
$X+=2$	$X=X+2$	الجمع
$X-=2$	$X=X-2$	الطرح
$X*=2$	$X=X*2$	الضرب
$X/=2$	$X=X/2$	القسمة
$X\2$	$X=X\2$	قسمة الرقم الصحيح
$X^=2$	$X=X^2$	الرفع للأس
$X\&="Text"$	$X=X\& "Text"$	ربط السلاسل

### وصل (ربط) العبارات

إذا أردت إدماج متغيرين (كل متغير له قيمة حرفية معينة) في متغير آخر جديد فعليك باستخدام المعامل "&" ويسمى Ampersand ويوضع بين القيمتين كما في المثال التالي:

```
First = "Compuscience"
```

Second = "Company"  
Result = First & Second

الناتج هو :

Compuscience Company

كما يمكنك استخدام المعامل "+" بدلاً من المعامل "&" للحصول على نفس النتيجة السابقة.

First ="Compuscience"  
Second = "Company "  
Result = First + Second

الناتج هو أيضاً:

Compuscience Company

## عمليات المقارنة

نقصد بما عمليات مقارنة بين قيمتين ويكون الناتج بالقيمة True وتعني نتيجة صحيحة أو القيمة False وتعني نتيجة خاطئة. وتعرف القيمة True أو القيمة False على أنها قيمة منطقية. ويوضح الجدول ١٠-٢ التالي المعاملات العلائقية وأمثلة على استخدامها. جدول ١٠-٢ المعاملات العلائقية وأمثلة على استخدامها.

المعامل	معناه	أمثله
=	Equals يساوى	"2" = "2" 3=3
<>	Not equal لا يساوى	"A"<"B" 4 <>2
>	Greater than أكبر من	"B">"A" 8 > 5.4
<	Less than اصغر من	"C"<"B" 4 < 6.5
> =	Greater than or equal أكبر من أو يساوى	"A">= "A" "D">="C" 5 > = 5 6 > = 5

"B" <= "B" "A" <= "B"	7 <= 7 6 <= 7	Less than or equal اصغر من أو يساوى	< =
--------------------------	------------------	--	-----

### العمليات المنطقية

يستخدم Visual Basic ستة أنواع من العوامل المنطقية تتعامل جميعها مع قيمتين منطقيتين (ما عدا العملية Not فتتعامل مع قيمة واحدة منطقية فقط) والجدول ١٠-٣ التالي يبين وظيفة كل عامل من العوامل المنطقية.

جدول ١٠-٣ العوامل المنطقية.

المعامل	اسمه	وظيفته
Not لا	Complement	في حالة (True) Not يكون الناتج False وفي حالة (False) Not يكون الناتج True.
And و	Conjunction	لا بد أن يكون التعبيران الشرطيان صحيحان لتكون النتيجة True وفي الأحوال الأخرى تكون النتيجة False.
OR أو	Disjunction	تكون النتيجة True إذا كان أحد التعبيرين الشرطيين صحيحا وتكون النتيجة False في الأحوال الأخرى.
XOR أو المقتصرة	Exclusive OR	تكون النتيجة True إذا كان واحد فقط من التعبيرين الشرطيين صحيحا، وتكون النتيجة False إذا كان كلاهما صح أو خطأ
Eqv تكافؤ	Equivalence	تكون النتيجة True إذا كان كلا التعبيرين الشرطيين صحيحا أو خطأ.

## أولويات تنفيذ المعاملات

إذا اشتمل الأمر على أكثر من نوع من العوامل السابقة فان تنفيذ هذه العوامل يتم طبقاً لأولويات محددة والجدول ١٠-٤ التالي يوضح ترتيب تنفيذ هذه الأولويات، فمثلا الترتيب رقم ١ في الجدول يسبق الترتيب رقم ٢ في التنفيذ... وهكذا بمعنى أن عمليات فك الأقواس تأتي قبل عمليات الأس يليها عملية الإشارة السالبة ثم عمليات الضرب والقسمة... وهكذا.

جدول ١٠-٤ أولويات المعاملات

الترتيب	معناه	المعامل
١	الأقواس Parantheses	()
٢	الأس Exponentiation	^
٣	الإشارة السالبة Negation	-
٤	الضرب والقسمة Multiplication, division	*, /
٥	الناتج الصحيح من القسمة Integer division	\
٦	باقي القسمة Modulo Arithmetic	Mod
٧	الجمع والطرح Addition, Subtraction	+, -
٨	عوامل علائقية Relational Operators	=, <, >, <>, <=, >=
٩	عوامل منطقية Logical Negation	Not

الترتيب	معناه	المعامل
١٠	Conjunction "و"	And
١١	Inclusive OR "أو"	OR
١٢	Exclusive OR "أو المقتصرة"	XOR
١٣	Equivalence "تكافؤ"	EqV

أمثلة:

مثال ١: العملية  $4 + 3 * 2$

يتم تنفيذها طبقاً للترتيب التالي:

١. يتم حساب عملية الضرب أولاً (الضرب يسبق الجمع)  $6 = 3 * 2$

٢. يتم حساب عملية الجمع ثانياً  $10 = 4 + 6$

مثال ٢: العملية  $(4+3) * 2$

يتم تنفيذها طبقاً للترتيب التالي:

١. يتم حساب ما بين القوسين أولاً (حيث أن الأقواس تسبق عملية الضرب)  $7=4+3$

٢. يتم حساب عملية الضرب ثانياً  $14 = 7 * 2$  ويكون الناتج هو 14.

### التعامل مع السلاسل الحرفية

تعرضنا فيما سبق للسلاسل الحرفية **Strings** وقلنا أنها الصورة التي يتم بها تخزين النصوص في ذاكرة الحاسب. حيث يشغل كل حرف أو رمز بايت واحد من الذاكرة. والسلاسل الحرفية نوع هام جداً من البيانات، حتى أن البيانات يمكن تقسيمها إلى نوعين أساسيين هما البيانات الرقمية والسلاسل الحرفية. قد لا تشعر بأهمية السلاسل الحرفية في الوقت الحاضر لو كنت من المبرمجين المبتدئين ولكن مع الوقت ستدرك أهميتها وتحتاج إلى هذا الشرح. فيما بقي من هذا الفصل ستعرف كيف تتعامل مع السلاسل الحرفية. لاحظ أن أي نص، مثل اسم الكائن **Name** وعنوانه **Text** الذي تكتبه في مربع هو في النهاية سلسلة حرفية.

بالرغم من سهولة تعديل النصوص داخل مربعات النصوص **Text Boxes** إلا أنه في كثير من الأحيان تحتاج إلى التعديل في النص بواسطة البرنامج وهناك العديد من الوظائف (الدوال) التي يمكن أن تستخدم لذلك. ولغة **Visual Basic** من اللغات الغنية في هذا المجال، حيث يوجد بها عدد كبير من الدوال المتخصصة في التعامل مع السلاسل الحرفية. جميع هذه الدوال يتم تمرير نص إليها في صورة متغير حرفي **String** وترجع متغيراً من النوع **.String**.

- تنقسم الوظائف (الدوال) التي تتعامل مع السلاسل الحرفية إلى عدة فئات:
- منها ما يقوم بحذف أو استقطاع جزء من السلسلة الأصلية ويكون بها سلسلة جديدة، يشمل الوظائف: **Left()** و **Right()** و **Mid()** و **SubString()**.
  - ومنها ما يقوم باستخراج سلسلة حرفية من أخرى بعد حذف المسافات الخالية **Spaces** وتشتمل على: **Trim()** و **TrimEnd()** و **PadLeft()**.
  - ومنها الذي يقوم بالتحويل بين الأحرف اللاتينية الكبيرة والصغيرة وتضم **ToUpper()** و **ToLower()** و **UCase()** و **LCase()**.
  - وهناك الوظيفة **InStr()** والوظيفة **IndexOf()** التي تبحث في سلسلة حرفية عن سلسلة أخرى وتعطي مكانها. والوظيفة **Len()** أو الخاصية **Length** التي تعطي طول السلسلة الحرفية (عدد حروفها).
  - وأخيراً الوظيفة **Replace()** المستخدمة في استبدال جزء داخل نص بجزء آخر والوظيفة **strReverse()** المستخدمة لعكس النص وغيرها من دوال تفكيك النص وتجميعه.

ونوضح في الفقرات التالية كيفية استخدام هذه الدوال.

## تغيير حالة الأحرف

هناك أربع وظائف تستخدم في تغيير حالة الأحرف اللاتينية وهي **ToUpper()** و **UCase()** و **ToLower()** و **LCase()** وكما يتضح من الأسماء الأولى

والثانية ترجع نص حروفه كلها تم تحويلها إلى حروف كبيرة **Capitals** والثالثة والرابعة تفعل العكس أي ترجع نص كل حروفه الهجائية **Small**. من الاستخدامات الجيدة لهذه الوظيفة، مقارنة قيمتين حرفيتين لا نهتم فيها بحالة الحروف. المثال التالي يوضح كيف يمكن مقارنة مدخلات بقيمة حرفية داخل البرنامج (مكتوبة بحروف كبيرة):

```
Select Case Ucase(txtInput.Text)
```

```
Case "TEACHER"
```

```
    'Login as Teacher
```

```
Case "STUDENT"
```

```
    'Login as Student
```

```
Case "OFFICER "
```

```
    'Login as Officer
```

```
End Select
```

في المثال السابق لا يهم أن يدخل المستخدم النص بحروف كلها كبيرة بل يكفي أن تتطابق الأحرف سواء كبيرة أو صغيرة.

يمكنك استخدام الوظيفة **ToUpper()** بدلاً من الوظيفة **UCase()** ليصبح الكود السابق كما يلي:

```
Select Case txtInput.Text.ToUpper
```

```
Case "TEACHER"
```

```
    'Login as Teacher
```

```
Case "STUDENT"
```

```
    'Login as Student
```

```
Case "OFFICER "
```

```
    'Login as Officer
```

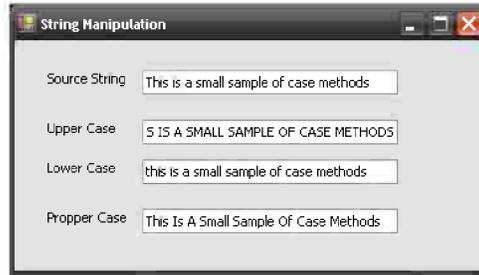
```
End Select
```

وبنفس الطريقة يمكنك استخدام الوظيفتين **LCase()** و **ToLower()** من الوظائف الأخرى المستخدمة في تغيير حالة الأحرف، الوظيفة **StrConv()** وهي

تستخدم في تحويلات متعددة منها التحويل بين الأحرف الكبيرة والصغيرة، ومنها تحويلات لا تهمنا مثل تحويلات بين حروف يابانية. ولكن أحد التحويلات الممكن القيام بها يستحق الذكر وهو تحويل الأحرف إلى "الحالة المناسبة" **Proper Case** ، أي تحويل الحرف الأول من الكلمة إلى الشكل الكبير وتحويل بقية الأحرف إلى الشكل الصغير في كافة الكلمات في النص. مثال لذلك في السطر التالي :

**StrConv(TextBox1.Text, VbStrConv.ProperCase)**

يوضح شكل ١٠-١ التالي التحويلات المختلفة للأحرف.



شكل ١٠-١ تأثير التحويلات المختلفة على حالة الأحرف.

## حذف وإضافة أجزاء إلى النص

من الوظائف الهامة المطلوبة لمعالجة المتغيرات الحرفية إمكانية إضافة أو حذف جزء من النص، وكذلك البحث عن نص داخل نص كبير. أو استبدال جزء من نص بآخر. وهذا هو ما سنتناوله فيما يلي.

*البحث عن نص داخل آخر*

نحتاج إلى البحث عن نص داخل نص آخر للعديد من الأسباب منها أن يكون النص مكون من العديد من المقاطع كل منها يحمل معلومة فنحتاج إلى تقطيعها للحصول على كل معلومة على انفراد. أول الوظائف المستخدمة في البحث عن نص داخل نص آخر هي الوظيفة **InStr()**.

صيغة استخدام هذه الوظيفة كالتالي:

**InStr(sourceStr,searchStr)**

والشكل البسيط لاستخدامها أن نحدد النص الذي سنبحث فيه `SourceStr` والذي سنبحث عنه `SearchStr` وسترجع لنا الوظيفة ترتيب الحرف الذي وجدت عنده النص المراد ، حيث دليل الحرف الأول من النص يكون ١ . أما إن لم تجد النص الذي تبحث عنه فسترجع القيمة صفر.

مثال:

`chpos=InStr("My Name Is Ahmed","Is")` ' Returns 9

`chpos=InStr("My Name Is Ahmed","he")` ' Returns 0

لو أردت البحث عن القيمة الحرفية بدءاً من حرف معين في النص المصدر ( الذي سنبحث فيه ) وليس من أول حرف يمكن أن تحدد هذا الحرف كالتالي:

`chpos=InStr("My Name Is Ahmed","Is",4)` 'Returns 9

لاحظ أن القيمة المرجعة من الوظيفة لا تتأثر، كل ما هنالك أنك تحاول اختصار زمن البحث.

توضيح أكثر لتأثير هذا المعامل في المثال التالي:

`chpos=InStr("My Name Is Ahmed","m")` 'Returns 6

`chpos=InStr("My Name Is Ahmed","m",7)` 'Returns 14

في المثال السابق ترجع الوظيفة عند البحث عن `m` القيمة ٦ إذا بدأت البحث من أول حرف. أما عند البدء من الحرف السابع فلن ترى الحرف السادس ولذا سترجع القيمة 14 وهو أول حرف `m` تالي.

هناك معامل اختياري آخر لهذه الوظيفة وهو يحدد إن كان البحث سيأخذ في اعتباره حالة الأحرف `Case-Sensitive` أم لا `Case-Insensitive`. القيمة الافتراضية هي `CompareMethod.Binary` (أو 0) ويتم فيها أخذ حالة الأحرف في الاعتبار، بينما القيمة `CompareMethod.Text` (أو 1) تحمل حالة الأحرف عند البحث.

مثال:

**chpos=InStr("My name Is Ahmed", "N") 'Returns 0**

**chpos=InStr("My name Is Ahmed", "N", CompareMethod.Text)**

**'Returns 4**

في المثال الأول اهتم البحث الافتراضي (القيمة صفر) بحالة الأحرف، ولذلك اعتبر حرف "n" غير موجود عند البحث عن حرف "N". أما في المثال الثاني فاعتبر الحرف موجوداً. المثال التالي يوضح استخدام هذه الوظيفة في عملية "تقطيع" نص إلى كلمات منفردة. يتضمن ذلك بحث متكرر عن الحروف واستقطاع للكلمات:

**Dim inCounter As Integer**

**Dim inFoundPos As Integer**

**Const PARSECHAR = " " 'تعريف المسافة**

'المتغير الحرفي فارغ

**If Len(stInput.Text) = 0 Then Exit Sub**

'البدء من الحرف الأول

**inCounter = 1**

'البحث عن المسافة

**inFoundPos = InStr(stInput.Text, PARSECHAR, 0)**

'عند العثور عليها نطبع الكلمة ونكمل البحث

**While inFoundPos <> 0**

**Debug.WriteLine(Mid(stInput.Text, inCounter, inFoundPos - inCounter))**

**inCounter = inFoundPos + 1**

**inFoundPos = InStr(inCounter, stInput.Text, PARSECHAR)**

**End While**

'طباعة آخر كلمة من النص

**If inCounter < Len(stInput.Text) Then**

**Debug.WriteLine(Mid(stInput.Text, inCounter))**

**End If**

إذا قمت على سبيل المثال بوضع العبارة التالية داخل مربع النص stInput:

**Welcome To Compuscience**

فستحصل على النتيجة التالية داخل النافذة الفورية Immediate Window (انظر شكل

٢-١٠.

```

Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    Dim inCounter As Integer
    Dim inFoundPos As Integer
    Const PARSECHAR = " "
    Dim stInput As String = "البحث عن أول مسافة"
    If Len(stInput.Text) = 0 Then Exit Sub
    inCounter = 1
    While inFoundPos < 0
        inFoundPos = InStr(inCounter, stInput.Text, PARSECHAR, 0)
        inCounter = inCounter + 1
    End While
    Debug.WriteLine(Mid(stInput.Text, inCounter, inFoundPos - inCounter))
    If inCounter < Len(stInput.Text) Then
        Debug.WriteLine(Mid(stInput.Text, inCounter))
    End If
End Sub
    
```

شكل ٢-١٠ استخدام الوظيفة InStr() لتقطيع النص إلى كلمات منفصلة.

على الرغم من شيوع استخدام الوظيفة Instr() في عمليات البحث داخل النصوص، إلا أنه يعاب عليها بدء ترقيم الحروف بالرقم 1 وليس 0 وهذا على عكس غالبية الدوال إن لم يكن كلها، لذا يفضل استخدام الوظيفة IndexOf() بدلاً منها.



### استخدام الوظيفة IndexOf()

تعتبر الوظيفة IndexOf() إحدى الوظائف التي ظهرت بالإصدار السابق من Visual Basic والمستخدم في البحث داخل النصوص. وتحتوي هذه الوظيفة على العديد من المعاملات إلا أنها تبدو في أبسط صورها على الصيغة التالية:

```
variable = text1.IndexOf("text2")
```

حيث:

- يعبر Variable عن المتغير الذي يحتوى على مكان وجود النص المراد البحث عنه.
- يعبر text1 عن النص الذي نرغب في البحث فيه.

- يعبر `text2` عن النص الذي نرغب في البحث عنه.

للتعرف على استخدام الوظيفة `IndexOf()`، دعنا نرى الكود التالي:

```
Dim strSentence As String
Dim intFoundPosition As Integer
strSentence = "I will see you next Friday"
intFoundPosition = strSentence.IndexOf("you")
If intFoundPosition < 0 Then
Debug.WriteLine("I could not find you")
Else
Debug.WriteLine("I found you at position " & intFoundPosition)
End If
```

في هذا الكود يتم البحث عن كلمة "you" داخل العبارة من خلال الوظيفة `IndexOf()`. فإذا قامت الوظيفة بإرجاع قيمة أقل من الصفر وهو ما يعني عدم وجود الكلمة بالعبارة يتم تنفيذ أحد العبارات وإلا يتم تنفيذ عبارة أخرى مع بيان مكان وجود النص. فنحصل على العبارة التالية:

I found you at position 11

*معرفة طول نص*

تستخدم الوظيفة `Len()` لمعرفة طول نص ( عدد حروفه ) ويستخدم ذلك عادة للتأكد من عدم تجاوز النص لطول معين. وصيغة هذه الوظيفة كالتالي:

```
inlength=Len(stName)
```

لاحظ أن هذه الوظيفة تعطي عدد الحروف وليس الطول الفعلي الذي سيأخذه النص عند عرضه على الشاشة بأحد الخطوط مقاساً بوحدات قياس الأطوال على الشاشة مثل البوصة أو البكسل `Pixel`، إذا أردت معرفة هذا الطول الفعلي استخدم الوظيفة `TextWidth()` وإذا أردت معرفة الارتفاع الفعلي استخدم الوظيفة `TextHeight()`. يمكنك أيضاً استخدام الخاصية `Length` للحصول على طول سلسلة البيانات كما في الكود التالي:

```
intValue = txtName.Length
```

*التخلص من المسافات الزائدة*

المسافات **Spaces** الزائدة هي المسافات التي توجد في أول أو آخر النص. وهي تؤدي إلى أن عمليات المقارنة بين نصين متشابهين تنتج عدم تشابه. فمثلاً العبارات الثلاثة التالية عند تنفيذها تعطي نتائج مختلفة رغم أن النص الفعلي واحد:

```
Debug.WriteLine (Len("Computer"))
```

```
Debug.WriteLine (Len("Computer"))
```

```
Debug.WriteLine (Len("Computer "))
```

لذلك يفضل التخلص من هذه المسافات الزائدة عند إجراء المقارناتوما إلى ذلك.

يتم التخلص من المسافات الزائدة بالدوال التالية:

• الوظيفة **TrimStart()**: تمحو المسافات من بداية النص

• الوظيفة **TrimEnd()**: تمحو المسافات من نهاية النص

• الوظيفة **Trim()**: تمحو المسافات من بداية ونهاية النص

لهذه الوظائف نفس الصيغة في الاستخدام، قم بتنفيذ الأمثلة التالية لتبين تأثير هذه الوظائف:

```
Debug.WriteLine(Trim("Waleed") & " " & Trim("Mohamed"))
```

```
Dim st As String = (" 12 Salah Eddin st. ")
```

```
st.Trim 'Remove spaces from left and right
```

```
st.TrimEnd 'Remove spaces from right only
```

```
st.TrimStart 'Remove spaces from left only
```

لا يقتصر استخدام الوظائف **Trim()** و **TrimEnd()** على حذف المسافات فقط وإنما يمكنك استخدامها في حذف الحروف الأخرى شريطة أن تكون في بداية النص أو نهايته كما في الكود التالي:

```
MessageBox.Show("racecar".Trim("rac").ToCharArray)
```

وفيه يتم حذف الحروف "rac" من بداية النص ونهايته وبالتالي يبقى الحرف "e" فقط.

*اجتثاث جزء من النص*

الوظائف الآتية تقوم بتكوين نص من نص آخر، وهو ما نعينه بالاقبباس:

الوظيفة(**Left**): ترجع عدد معين من الأحرف من بداية النص  
 الوظيفة(**Right**): ترجع عدد معين من الأحرف من نهاية النص  
 الوظيفة(**Mid**): ترجع عدد معين من الأحرف من وسط النص ( بداية من حرف محدد)  
 الوظيفة(**SubString**): ترجع عدد معين من الأحرف من أى مكان فى النص ويمكنك  
 استخدام هذه الوظيفة كبديل للوظائف الثلاثة السابقة.  
 الصيغة العامة لهذه الوظائف كالاتي:

**OutStr=Microsoft.VisualBasic.Left (InpStr, NumChars)**  
**OutStr=Microsoft.VisualBasic.Right (InpStr, NumChars)**  
**OutStr=Mid (InpStr, startpos [,NumChars])**  
**OutStr=InpStr.SubString(startpos[,NumChar])**

حيث **OutStr** هو النص الناتج ، **InpStr** هو النص الأصلي ، **NumChars** عدد  
 الأحرف المراد اقتباسها، **startpos** هو ترتيب الحرف الذي يبدأ عنده الاقتباس.  
 الأمثلة الآتية توضح استخدام الوظيفة والقيمة التي ترجعها:

النص الذي تنشئه الوظيفة	الصيغة
"This"	Microsoft.VisualBasic.Left("This is My Name",4)
"Name"	Microsoft.VisualBasic.Right("This is My Name",4)
"is"	Mid("This is My Name",6,2)
"is is"	"This is My Name".SubString(2,5)
"is is My Name"	"This is My Name".SubString(2)

لاحظ أن **NumChars** إجباري في الوظيفتين الأولىين ولا بد من أن يكون عدد أكبر من  
 صفر، ولو ساوى صفر سنحصل على نص خالي (طوله صفر)، ولو زاد عن عدد الحروف في  
 النص الأصلي سينتج النص الأصلي بأكمله.  
 أيضا **startpos** إجباري في الوظيفتين الآخرتين، ولا بد أن يكون عدد أكبر من صفر، وإن

كان بعدد أكبر من طول النص فسينتج النص الخالي ( **zero length string** )، و**NumChars** في الدالتين اختياري، وعند عدم تحديده سترجع الوظيفة الحروف بدءاً من **startpos** وحتى نهاية النص.

### استبدال الأحرف داخل النص

هل استخدمت خاصية البحث والاستبدال **Find and Replace** من قبل في أي برنامج لمعالجة النصوص؟ لو قمت بذلك - وهذا شبه أكيد- فإن الوظيفة **Replace()** تصبح سهلة التناول للغاية، فهي المكافئ البرمجي لهذه الخاصية في برامج تحرير النصوص. كمثال مبدئي لنفرض أن لديك سلسلة بيانات طويلة كالتالي:

```
strDocument= "This is the first line in the document..."
```

لاحظ أن هذه السلسلة قد تكون محتويات مربع نصوص.

لو أنك تريد في برنامجك أن تستبدل أي نص جزئي مثل **"first"** بنص جزئي آخر، وليكن **"second"** في كافة السلسلة يمكنك أن تستخدم الوظيفة الجديدة **Replace()** كالتالي:

```
strResult=Replace(strDocument, "first" , "second")
```

السلسلة الناتجة بعد التغيير هي **strResult** فنحصل على العبارة التالية:

```
This is the second line in the document...
```

الصورة السابقة في الاستخدام هي الصورة البسيطة، أما الصيغة الكاملة للوظيفة **Replace()** فهي:

```
strResult=Replace(strSource, strFind, strReplace, intStart,  
intCnt, intCompare)
```

حيث المعاملات للوظيفة كالتالي:

**strSource**: النص المطلوب البحث فيه

**strFind**: النص المطلوب البحث عنه

**strReplace**: النص المطلوب الاستبدال به

**intStart**: ترتيب الحرف المراد بدء البحث منه ( في النص **strSource** )

**intCnt** : أقصى عدد لمرات الاستبدال  
**intCompare** : قيمة رقمية (0 أو 1) تحدد هل يتم الالتزام بحجم الحروف **Capital** أو **small** عند المقارنة ( في الحروف اللاتينية فقط).

وترجع الوظيفة **strResult()** وهي سلسلة حرفية تحتوي على النص بعد الاستبدال، لو استخدمت المعامل **intStart** فإن السلسلة الناتجة تبدأ من هذا الحرف المحدد داخل **intStart**.

بالنسبة للمعامل **intCompare**، فهو يحدد أسلوب المقارنة كما في جدول ١٠-٥ التالي:

جدول ١٠-٥ قيم المعامل **intCompare**.

التوضيح	القيمة	الثابت
يتم أخذ حالة الحرف <b>Case</b> ( <b>Capital</b> or <b>small</b> ) في الاعتبار، أي أن 'a' مثلاً لا تكافئ 'A' في هذه الحالة	0	<b>CompareMethod.Binary</b>
يتم تجاهل حالة الحرف <b>Case</b> ( <b>Capital</b> or <b>small</b> ) عند المقارنة، أي أن 'a' تكافئ 'A' في هذه الحالة	1	<b>CompareMethod.Text</b>

لا يقتصر استخدام الوظيفة على البحث فقط، وإنما يمكنك استخدامها في إزالة نصوص أو حروف معينة، بأن يتم تحديد نص الوظيفة **Replace()** خالي ("") في معامل **strReplace**. ونظراً لأن عملية الاستبدال عملية ليست بسيطة، فإن الوظيفة **Replace()** قد تعطي في سلسلة البيانات الناتجة رسالة خطأ أو دليل عليها وليس سلسلة حرفية صحيحة، يوضح جدول ١٠-٦ التالي هذه الحالات:

جدول ١٠-٦ الحالات الخاصة للوظيفة **Replace()**

الحالة	القيمة المرجعة <b>strResult</b>
<b>StrSource</b> نص خالي	نص خالي ""

القيمة المرجعة <code>strResult</code>	الحالة
<code>Error</code> ( يتوقف البرنامج )	<code>StrSource=NULL</code>
نسخة من <code>strSource</code>	<code>strFind</code> نص خالي
نسخة من <code>strSource</code> مع محو كل <code>strFind</code> يتم إيجاده	<code>strReplace</code> نص خالي
نص خالي ""	<code>IntStart &gt; len(strSource)</code>
نسخة من <code>strSource</code>	<code>intCnt=0</code>

في الحقيقة هذه الوظيفة بها الكثير من المعاملات، لذا ستجد في المشروع `Strings` المرفق نموذج دسم بالعناصر لتوضيح كافة معاملات هذه الوظيفة، كما يبدو عند استخدامه في شكل ١٠-٣. قم بتجربة كافة البدائل لكل اختيار، لكي تدرك تأثير المعاملات المختلفة على الوظيفة.



شكل ١٠-٣ برنامج `Strings`، عند استخدامه لاستبدال نص في سلسلة حرفية.

تستخدم الوظيفة (`Mid`) أيضاً لحذف جزء من وسط نص، نفس الاسم يستخدم لاستبدال جزء من نص بآخر، ولكن في هذه الحالة لا نستخدم الوظيفة (`Mid`) ولكن العبارة `Mid` وصيغة الاستخدام مختلفة قليلاً، فالعبارة لن ترجع نص آخر وإنما ستقوم بتغيير في نص

موجود.

الصيغة العامة لهذه العبارة كالتالي:

**Mid (sourcestr, startpos [, numchars]) = replstr**

حيث **replstr** هو النص المراد وضعه بدل آخر موجود في النص الأصلي. وبقية المعاملات سبقوا وضحناها.

مثال على العبارة السابقة:

```
strN= "My Name Is Hassan"
```

```
Mid (strN,12) = "Waleed"
```

```
Debug.WriteLine(strN) ' --- gives the string "My Name Is Waleed"
```

*عكس ترتيب الحروف في سلسلة بيانات*

تتيح لك الوظيفة الجديدة **StrReverse()** أن تعكس ترتيب الحروف في سلسلة حرفية

بالكامل، مثلا السلسلة "This is a Computer" تصبح "retupmoC a si sihT"

بعد استخدام هذه الوظيفة، صيغة استخدام الوظيفة السابقة هي:

```
strResult = StrReverse(strsource)
```

حيث:

- **strSource** هي السلسلة الأصلية.

- **strResult** هي السلسلة الناتجة بعد العكس.

لكي ترى تأثير هذه الوظيفة قم باستخدام النموذج **frmReverse** داخل

المشروع **Strings**، وأدخل جملة في المكان المخصص ثم اضغط زر التنفيذ، لترى النص

الناتج بعد تنفيذ الوظيفة، كما ترى في شكل ١٠-٤.



شكل ١٠-٤ دالة `strReverse()` عند تطبيقها على سلسلة حرفية في برنامج `Strings`.

### تجزئة سلسلة بيانات إلى مصفوفة من سلاسل البيانات

تستقبل الوظيفة (`Split()`) سلسلة بيانات ثم تقوم بتقطيعها إلى كلمات منفصلة، يعتمد هذا بالطبع على معرفة الفواصل التي تفصل بين الكلمات في السلسلة الأصلية. فلو كانت هذه السلسلة مثلاً عبارة عن جملة، فإن الفاصل عادةً ما يكون المسافة `Space`، ولو كانت سجل من قاعدة بيانات فإن الفاصل عادةً يكون الفاصلة العادية `','` `Comma`. تتيح (`Split()`) تعريف فاصل `Separator` آخر، يتم التجزئة عندها إذا وجدت، ولن ترجع الوظيفة بالطبع هذه الفواصل مع الكلمات الناتجة بعد التجزئة.

الصيغة العامة للوظيفة (`Split()`) كالتالي:

```
strResult=Split(strList,strDelimiter,intElemCnt, intCompare)
```

حيث:

- `strResult`: هي المصفوفة الناتجة.
- `strList`: هو سلسلة البيانات المراد تجزئتها
- `strDelimiter`: هو رمز يؤخذ كفاصلة تستخدم في تجزئة السلسلة، القيمة الافتراضية له هو الرمز `Space`.
- `intElemCnt`: هو العدد الأقصى للعناصر في المصفوفة الناتجة
- `intCompare`: هو عدد يحدد عملية مقارنة الفاصل `delimiter` هل تتم مع أخذ حالة الحرف (`Capital or Small`) في الاعتبار أم لا (إن كان من الحروف

الأبجدية اللاتينية ) وهي تأخذ أحد القيم التالية:

التوضيح	القيمة	الثابت
يتم أخذ حالة الـ delimiter في الاعتبار	0	CompareMethod.Binary
لا يتم أخذ حالة الـ delimiter في الاعتبار	1	CompareMethod.Text

إن هذه الوظيفة القوية (**Split()**) والتي ترجع مصفوفة من السلاسل الحرفية الجزئية، تفتح

تعمل المصفوفة **Array** بطريقة مشابهة لطريقة عمل المتغير إلا أنها تحتوي على عدد من العناصر المتماثلة بدلاً من عنصر واحد في حالة المتغير، حيث يتم استخدام الرمز ( ) لتمييز المتغير عن المصفوفة. وسوف نتناول المصفوفات بالتفصيل بالفصل الثاني عشر إن شاء الله.



الطريق إلى اثنين من الوظائف القوية أيضا وهما: (**Filter()** و(**Join()** الآتيتين.

يوضح شكل ١٠-٥ النموذج frmConversion داخل المشروع Strings، تجد مربع نص، قم بكتابة ما تشاء فيه، وحدد الفاصل، ثم اضغط زر "قم بالتجزئة"، يظهر الناتج أمامك في مربع السرد الأيمن.

The screenshot shows a Windows application window titled "استخدم خوارزميات البيانات المتقدمة". The interface includes a text input field for "أدخل السلسلة المراد تجزئتها:" with the value "السبب الأجد التي التلق الأخطاء الخمس الخمسة". Below it is a "الفاصل المستخدم:" field with a "قم بالتجزئة" button. A "المصفوفة الناتجة:" section contains a list of characters: "البريت", "الأحد", "الاثنين", "الثلاثاء", "الأربعاء", "الخميس", "الجمعة". There is also a "أوجد ما يخوف:" field and a "قم بالدمج مستخدماً الفاصل:" field. A "خروج" button is at the bottom.

شكل ١٠-٥ النموذج frmConversion عند استخدامه في تجزيء سلسلة حرفية تحوي أيام الأسبوع.

### تصفية مصفوفة من السلاسل الحرفية

بعد تجزئة السلسلة الحرفية إلى عناصر في مصفوفة حرفية **String Array**، قد تحتاج إلى اختيار عناصر معينة من هذه المصفوفة بناءً على وجود أحرف أو نص معين فيها، تقوم الوظيفة (**Filter**) بإرجاع مصفوفة جديدة تحتوي على العناصر التي يتحقق فيها الشرط الذي تحدده فقط.

الصيغة العامة لهذه الوظيفة كالتالي:

```
strResult = Filter( strList, strFind, bolInclude, intCompare)
```

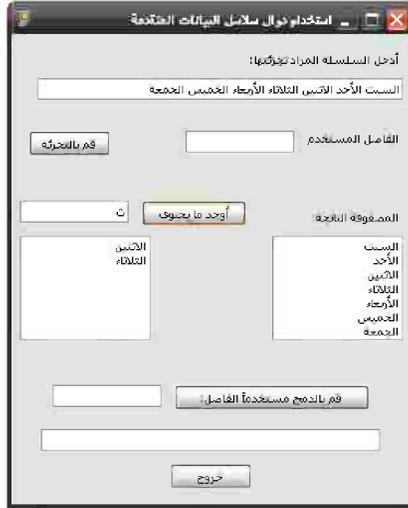
حيث:

- **strResult**: مصفوفة تحتوي على العناصر التي يوجد بها النص المحدد **strFind** من المصفوفة **strList**.
- **strList**: المصفوفة التي يتم البحث في عناصرها.
- **strFind**: النص المراد البحث عنه في مصفوفة السلاسل الحرفية **strList**.
- **bolInclude**: متغير منطقي (نعم/ لا) يحدد هل يتم تكوين المصفوفة الجديدة من العناصر التي وجد بها النص **strFind** (نعم)، أم من العناصر التي لا تحتوي على هذا النص ( لا ).
- **intCompare**: عدد يبين طريقة البحث عن **strFind** في **strList**، هل يهمل حالة الحرف (Case) أم يأخذها في الاعتبار (راجع الوظيفة **Split()** لمعرفة القيم المختلفة لهذا المعامل).

لو لم تجد الوظيفة أي عنصر يحتوي على النص **strFind** فإنها ترجع مصفوفة خالية ( لو كان **bolInclude=True**)، أما لو كانت المصفوفة **strList** نفسها خالية أو مصفوفة متعددة الأبعاد **multi-dimensional**، فإن الوظيفة تصدر خطأ.

لو اتخذنا المصفوفة التي أنشأناها في المثال السابق كمثال والتي تحتوي على أيام الأسبوع،

يمكننا أن نبحث مثلاً عن أي عنصر يحتوي على حرف "ث" مثلاً، كما ترى في شكل ١٠-٦.



شكل ١٠-٦ استخدام الوظيفة (Filter) لاختيار عناصر معينة من مصفوفة سلاسل حرفية.

### دمج عناصر مصفوفة حرفية

الوظيفة (Join) هي المقابلة في وظيفتها للوظيفة (Split)، فإن كانت (Split) تقوم بتجزئه سلسلة حرفية إلى سلاسل صغيرة وترجع مصفوفة تحتوي على هذه السلاسل، فإن (Join) تقوم بإنشاء سلسلة حرفية من عناصر مصفوفة. الصيغة العامة لهذه الوظيفة كالتالي:

**strResult = Join (strList, strDelimiter)**

حيث:

- **strList**: هو المصفوفة المحتوية على العناصر المراد دمجها.
- **strDelimiter**: هو الرمز المستخدم كفاصل يوضع بين العناصر عند تكوين السلسلة الحرفية.
- **strResult**: هي السلسلة الحرفية الناتجة بعد دمج عناصر المصفوفة.

كامتداد للمثال السابق، يمكننا أن ندمج عناصر المصفوفة ثنائية، ولكن مع استخدام فاصل مختلف ، وليكن "--" ، كما ترى في شكل ٧-١٠.



شكل ٧-١٠ استخدام الوظيفة (Join) لدمج المصفوفة من جديد مع تغيير الفاصل.

يحتوى البرنامج **Strings** على أكثر من نموذج. فإذا أردت تجربة عمل أحد النماذج، اجعل هذا النموذج هو نموذج بدء تشغيل البرنامج من خلال نافذة الخصائص الخاصة بالمشروع.



### التعامل مع الأحرف الخاصة

هناك بعض الأحرف التي لا يمكن كتابتها وإن كان لها أثر واضح في النصوص عموماً، مثل حرف البدء بسطر جديد.

وللتعبير عن هذه الأحرف التي لا تظهر ضمن الحروف الأبجدية والرموز المعروفة نستخدم الوظيفة **Chr(X)** حيث **X** كود يعبر عن الحرف المراد كتابته بالنظام **ASCII** وتراوح قيمته من ٠ إلى ٢٥٥ والتي تتضمن الأحرف والرموز الأخرى المستخدمة مثل < ، > وغيرها، و أيضاً تتضمن كود الأحرف الخاصة مثل **Backspace** والسطر الجديد وخلافه.

مثلا السطر التالي يستخدم هذه الوظيفة لإدراج رموز البدء بسطر جديد في أحد المتغيرات الحرفية:

**MessageBox.Show(Chr(13) & Chr(10))**

حيث الرمز 10 (NewLine) و13 (Carriage Return) يسببا الفصل بين السطور. يوضح الجدول ١٠-٧ التالي بعض الأحرف الخاصة التي لا يمكن إدخالها في النص من خلال لوحة المفاتيح:

جدول ١٠-٧ أكواد بعض الأحرف الخاصة

الكود	ما يمثله
٨	مفتاح التراجع Backspace
٩	مفتاح Tab
١٠	مفتاح الانتقال إلى سطر جديد
١٣	مفتاح بدء من أول السطر
٣٢	مسافة
٣٤	علامة الحذف المزدوجة ("")
٤٨	رمز الرقم (0)
٦٥	رمز حرف (A)
٩٧	رمز حرف (a)

وهناك وظيفة تقوم بعكس ما تقوم به الوظيفة Chr() وهي الوظيفة Asc ("") ويتم إعطاء هذه الوظيفة الحرف فتعطي الوظيفة الكود الخاص بهذا الحرف فمثلا عند كتابة Asc("A") تعطينا القيمة ٦٥.

### التعامل مع الحروف والأرقام

كثيراً ما نجد أرقام يتم التعامل معها على أنها مدخلات نصية وليست أعداد. فمثلاً الرقم البريدي لا يتم التعامل معه كقيمة قابلة للجمع والطرح ولكن كود يعبر عن منطقة معينة.

وكثيراً ما نحتاج إلى تحويل مدخل ما من الصورة النصية إلى الصورة العددية أو العكس. لهذا الغرض نستخدم الوظائف الآتية:

الوظيفة	الاستخدام
Str(x)	لتحويل المتغير الرقمي "x" إلى الصيغة النصية
Val(x)	لتحويل المتغير النصي (المكون من أرقام) "x" إلى الصيغة الرقمية

أمثلة لاستخدام هذه الوظائف:

الصيغة	النتائج
Str(123)	"123" (ثابت حرفي)
Val("123")	123 (عدد صحيح)
Val("123 Kg")	123 (يتم إهمال الحروف غير الرقمية)
Val("Street")	0

## التعامل مع التاريخ والوقت

التواريخ والأوقات ليست أرقام، وليست سلاسل نصية. ولكي تتعامل مع التواريخ والأوقات في Visual Basic، فأنت بحاجة لاستخدام دوال خاصة بالتاريخ. وباستخدام دوال التاريخ الخاصة، يمكنك جمع فترة زمنية (أيام، أسابيع، ساعات... الخ)، وتحديد فترة زمنية تقع بين تاريخين أو وقتيين، وكذلك معرفة الوقت أو التاريخ في أى لحظة حسب ساعة حاسبك.

### استرجاع تاريخ النظام الحالي ووقته

يوجد في Visual Basic وظيفتين لإرجاع معلومات عن الوقت والتاريخ الحاليين للنظام وهما الوظيفة (Now) التي تقوم بإرجاع التاريخ والوقت الحاليين والوظيفة (Today) التي تقوم بإرجاع التاريخ الحالي ووقت منتصف الليل كما في الكود التالي:

```
Dim dt As Date = Now()
Debug.WriteLine(Date.Now)
Debug.WriteLine(Date.Today)
```

### Debug.WriteLine(dt)

وخرج هذا الكود يكون كما في شكل ١٠-٨ التالي.



شكل ١٠-٨ الحصول على التاريخ والوقت الحاليين.

وكما ترى فإننا نحصل من الوظيفتين على التاريخ والوقت كاملين، لكن ماذا لو أردنا الحصول على جزء من التاريخ أو جزء من الوقت مثل اليوم فقط أو الساعة فقط أو الشهر فقط ؟ في هذه الحالة يمكنك استخدام الخصائص المصاحبة لكل دالة للحصول على الجزء المطلوب كما في العبارة التالية:

### Debug.WriteLine(Now.DayOfWeek)

التي تقوم بإرجاع اسم اليوم الموجود بالتاريخ الحالي. يوضح الجدول ١٠-٨ التالي الخصائص التي يمكنك استخدامها للحصول على أجزاء من التاريخ والوقت.

جدول ١٠-٨ الخصائص المستخدمة للحصول على أجزاء معينة من التاريخ والوقت

اسم الخاصية	القيمة المرجعة	نوع البيانات الناتج
Date	التاريخ فقط	Date
TimeOfDay	الوقت فقط	TimeSpan
Month	الشهر من ١ إلى ١٢	Integer
Day	اليوم من ١ إلى ٣١	Integer
Hour	الساعة من ٠ إلى ٢٣	Integer
Minute	الدقيقة من ٠ إلى ٥٩	Integer
Second	الثانية من ٠ إلى ٥٩	Integer
Millisecond	الملي ثانية من ٠ إلى ٩٩٩	Integer
DayOfWeek	اليوم في الأسبوع من ٠ إلى ٦	Integer

اسم الخاصية	القيمة المرجعة	نوع البيانات الناتج
DayOfYear	اليوم في السنة من ١ إلى ٣٦٦	Integer

### زيادة التاريخ أو الوقت

لإضافة فترة زمنية لتاريخ أو وقت محدد، استخدم الوظيفة **DateAdd()**. وهذه الوظيفة تأخذ الشكل التالي:

**DateAdd (Interval, Number, DateValue)**

حيث:

- **Interval**: هي وسيطة الفترة الزمنية (**Argument**) المطلوب إضافتها
- **Number**: عدد السنين أو الشهور أو الأيام... الخ المطلوب إضافتها للتاريخ المحدد
- **DateValue**: التاريخ المطلوب إضافة فترة زمنية إليه.

وسيطه الفترة الزمنية (**Argument**) عبارة عن ثابت أو متغير أو تعبير يمرر إلى الإجراء.



يعرض الجدول ١٠-٩ التالي القيم الممكنة للفترة الزمنية. ويجب أن تضع وسيطة الفترة الزمنية بين علامتي اقتباس أو بدلاً من ذلك يمكنك اختيار الثوابت التي تظهر أثناء كتابتك لكود الوظيفة.

جدول ١٠-٩ ضبط الفترة الزمنية للوظيفة **DateAdd()** والوظيفة **DateDiff()**

الضبط	التوصيف	الثابت
YYYY	سنة	DateInterval.Year
Q	ربع سنة	DateInterval.Quarter
M	شهر	DateInterval.Month
Y	يوم من السنة	DateInterval.DayofYear
D	يوم	DateInterval.Day

الثابت	التوصيف	الضبط
DateInterval.DayofWeek	أيام أسبوع	W
DateInterval.Week	أسبوع	WW
DateInterval.Hour	ساعة	H
DateInterval.Minite	دقيقة	N
DateInterval.Second	ثانية	S

لتوضيح استخدام هذه الوظيفة، دعنا نرى الكود التالي:

```
Dim dt As Date = Now()
dt = DateAdd(DateInterval.Day, 2, Now)
Debug.WriteLine(dt)
dt = DateAdd(DateInterval.Month, 2, Now)
Debug.WriteLine(dt)
dt = DateAdd(DateInterval.Year, 2, Now)
Debug.WriteLine(dt)
```

إذا قمت بتنفيذ هذا الكود، تحصل على نتيجة مشابهة لشكل ١٠-٩ التالي.



شكل ١٠-٩ نافذة استخدام الوظيفة DateAdd().

تحديد الفترة الزمنية بين تاريخين أو وقتين

يمكنك بسهولة حساب الفترة الزمنية بين تاريخين أو وقتين بواسطة الوظيفة DateDiff().

والشكل العام للوظيفة DateDiff() كالتالي:

DateDiff(interval, Date1, Date2)

تقبل Interval (وسيلة الفترة الزمنية) نفس القيم بالضبط كالوظيفة السابقة DateAdd()

(انظر الجدول ١٠-٨ السابق) و Date1 هو التاريخ المطروح بينما Date2 هو التاريخ

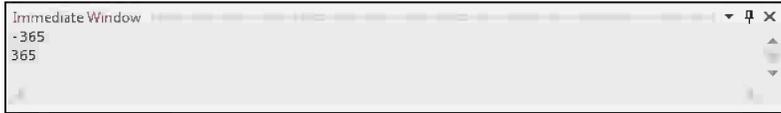
المطروح منه. الوظيفة DateDiff() ترجع رقما من النوع Long يشير إلى عدد الفترات الزمنية

المحددة بين تاريخين (بالسنة ، أو الشهر ، أو اليوم... الخ).

لتوضيح وشرح الوظيفة (`DateDiff()`) دعنا نرى الكود التالي:

```
Dim Lvar As Long
Dim dt1 As Date = "12 - 12 - 2003"
Dim dt2 As Date = "12 - 12 - 2002"
Lvar = DateDiff(DateInterval.Day, dt1, dt2)
Debug.WriteLine(Lvar)
Lvar = DateDiff(DateInterval.Day, dt2, dt1)
Debug.WriteLine(Lvar)
```

إذا قمت بتنفيذ هذا الكود، تحصل على نتيجة مشابهة لشكل ١٠-١٠ التالي.



شكل ١٠-١٠ استخدام الوظيفة (`Date Diff`)

لاحظ في الأمثلة السابقة أن عبارة الطباعة الأولى ترجع رقما سالبا، وذلك لأن التاريخ الثاني أحدث من التاريخ الأول، أي أن الفترة المحددة لم تمر بعد. وهذا الأمر سيكون مفيدا في تحديد ترتيب تاريخين لأنك بهذه الطريقة تستطيع مقارنتهما باستخدام الوظيفة (`Date Diff`) وتحديد التاريخ الأحدث (الأقرب) من خلال ما ترجعه الوظيفة (`Date Diff`) فإذا كان سالبا فإن هذا يعني أن التاريخ الأول أقدم، وإذا كان موجبا فإن هذا يعني أن التاريخ الأول أحدث.

