**الفصل الثالث**

**المتغيّرات**

* [**المتغيّرات Variables:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\03-%20المتغيرات\03-%20المتغيرات\1-%20المتغيّرات.htm)

ما هي المتغيّرات؟

* [**تعريف المتغيرات Declaring Variables:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\03-%20المتغيرات\03-%20المتغيرات\2-%20تعريف%20المتغيرات.htm)

صيغة تعريف المتغيّرات.

* [**أنواع المتغيرات Types of Variables:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\03-%20المتغيرات\03-%20المتغيرات\3-%20أنواع%20المتغيرات.htm)

نظرة شاملة لأنواع المتغيّرات، وكيفيّة التعامل مع كلّ منها.

* [**تنسيق الأرقام Formatting Numbers:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\03-%20المتغيرات\03-%20المتغيرات\4-%20تنسيق%20الأرقام.htm)

كيف تعرض الرقم في مربّع النصّ بنسقٍ معيّن، كعدد الخانات العشريّة أو وجود رمز معيّن قبل أو بعد الرقم... إلخ.

* [**الاختصارات المحدّدة للأنواع Data Type Identifiers:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\03-%20المتغيرات\03-%20المتغيرات\5-%20الاختصارات%20المحدّدة%20للأنواع.htm)

طريقة بديلة لتعريف المتغيّر بدون كتابة اسم النوع.

* [**خيار "التعريف الصريح"Option Explicit :**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\03-%20المتغيرات\03-%20المتغيرات\6-%20خيار%20التعريف%20الصريح.htm)

كيف تجبر نفسك على تعريف المتغيّر قبل استخدامه!

* [**خيار "التحويل الدقيق" Option Strict:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\03-%20المتغيرات\03-%20المتغيرات\7-%20خيار%20التحويل%20الدقيق.htm)

كيف تمنع اللغة عن التحويل التلقائي بين نوعين مختلفين من أنواع البيانات.

* [**التحويل بين أنواع المتغيّرات Converting Variable Types:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\03-%20المتغيرات\03-%20المتغيرات\8-%20التحويل%20بين%20أنواع%20المتغيّرات.htm)

كيف تقوم أنت بالتحويل بين أنواع البيانات، في حالة تفعيل خيار التحويل الدقيق.

* [**القيمة "لا شيء" Nothing:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\03-%20المتغيرات\03-%20المتغيرات\9-%20القيمة%20لا%20شيء.htm)

للكائنات فقط!

* [**اختبار أنواع المتغيّرات Examining Variable Types:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\03-%20المتغيرات\03-%20المتغيرات\10-%20اختبار%20أنواع%20المتغيّرات.htm)

كيف تتأكّد من نوع البيانات الموجودة في المتغيّر؟

* [**مجال المتغيّر Variable Scope:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\03-%20المتغيرات\03-%20المتغيرات\11-%20مجال%20المتغيّر.htm)

داخل أيّ نطاق تستطيع استخدام المتغيّر؟

* [**عمر المتغيّر Variable Lifetime:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\03-%20المتغيرات\03-%20المتغيرات\12-%20عمر%20المتغيّر.htm)

إلى متّى يظلّ المتغيّر محتفظا بقيمته؟

* [**الثوابت Constants:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\03-%20المتغيرات\03-%20المتغيرات\13-%20الثوابت.htm)

هل هناك متغيّرات لا تتغيّر قيمتها أبدا؟

* [**المصفوفات Arrays:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\03-%20المتغيرات\03-%20المتغيرات\14-%20المصفوفات.htm)

كيف يمكن التعامل مع مجموعة كبيرة من البيانات ذات النوع المشترك.

* [**الأنواع التي يعرّفها المستخدم User-Defined Data Types:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\03-%20المتغيرات\03-%20المتغيرات\15-%20الأنواع%20التي%20يُعرّفها%20المستخدم.htm)

التعرّف على المرقّمات Enums والسجلات Structures.

**المتغيّرات Variables:**

كما في أيّ لغة برمجة، تقوم المتغيّرات بتخزين القيمِ أثناء تنفيذ البرنامج.. وطبعًا سُمّيت متغيّراتٍ، لأنّك تستطيع تغيير قيمها في أيّ لحظةٍ أثناء تنفيذ البرنامج.

وللمتغيّر اسمٌ وقيمة.. فمثلا: المتغيّر "اسم المستخدم" UserName يمكن أن تُوضع به القيمة "محمد".. والمتغيّر "الخصم" Discount يمكن أن تُوضع به القيمة 0.35.

تلاحُظ هنا أنّ القيمتين "محمد" و 0.35 مختلفتان، فالأولى نص String لهذا تمّ وضعها بينَ علامتَيْ تنصيص، بينما الثانية قيمة رقميّة Numeric Value.

وكما ذكرنا سابقا، فإنّ المتغيّرات في VB.NET ليست مجرّد أسماءٍ أو مخازن للقيم.. إنّها كذلك كِيانات ذكيّةٌ لتخزين وإجراء العمليات على القيم.. باختصار: إنّها كائنات Objects، لها وسائلها وخصائصها الخاصة بها.

فمثلا: هذه الجملة تعرّفُ متغيّرا (كائنا) من النوع "تاريخ" Date:

**Dim Expiration As Date**

ويمكنك وضع تاريخ في هذا المتغير بجملةٍ كالتالية:

**Expiration = #1/1/2003#**

بل ويمكنك إجراء بعض العمليات على هذا المتغيّر مثل:

**Expiration.AddYears(3)**

حيث سيكونُ ناتج هذه العمليّة تاريخٌ جديد، يزيد بثلاث سنواتٍ عن التاريخ الأوّل.. هذا التاريخ الجديد يمكن تخزينه في أيّ متغيّر آخر كالتالي:

**Dim NewExpiration As Date**

**NewExpiration = Expiration.AddYears(3)**

لاحظ أنّ المتغير الذي يقع على يسار علامة "="، هو فقط الذي تتغيّر قيمته، بعد إجراء العملية التي تقع على يمينها.. أي أن قيم المتغيرين ستكون كما بالجدول:

|  |  |  |
| --- | --- | --- |
| المتغيّر | قيمته قبل تنفيذ العملية | قيمته بعد تنفيذ العملية |
| Expiration | #1/1/2003# | #1/1/2003# |
| NewExpiration | #12:00:00 AM# | #1/1/2006# |

كما أنّ بإمكانك قراءة المتغيّر وتغيير قيمته في جملةٍ واحدة.. مثال:

**Expiration = Expiration.AddYears(3)**

حيث يقرأ المترجم التاريخ الموجود في المتغير Expiration، ويضيف عليه ثلاث سنوات، ثم يخزّن التاريخ الجديد في نفس المتغيّر.. بعد تنفيذ هذه العملية، ستصير قيمة المتغير Expiration هي: #1/1/2006#.

الجميلُ في الأمر، هو أنّ معظم الوظائف التي تحتاجها للتعامل مع التواريخ و الأرقام والنصوص، تمنحها لك اللغة جاهزةً، لتريحك من عناء كتابتها من البداية.

**والآن، وبمنتهى البساطة:**

استخدم المتغيّرات لتخزين القيم، فإذا احتجت لمعالجة هذه القيم، فاكتب اسم المتغير متبوعا بنقطة "."، وستظهر لك قائمة تعرض لك كل خصائصه، وكل الدوال الجاهزة التي يمكن تطبيقها عليه، وفي معظم الحالات ستجد أسماءها معبّرةً عن وظائفها، بالإضافة لأنّ مجرد تحديد أيّ منها في القائمة (بضغطه مرّة واحدة بالفأرة أو الانزلاق إليه بالأسهم من لوحة المفاتيح)، يؤدّي إلى ظهور تلميح على الشاشة، يشرح لك وظيفة هذا العضو.

**تعريف المتغيرات Declaring Variables**

* + [**لماذا نعرّف المتغيّر:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\03-%20المتغيرات\2-%20تعريف%20المتغيرات\1-%20لماذا%20نعرّف%20المتغيّر.htm)
  + [**بعض الاختلافات عن VB6 في التعامل مع المتغيرات:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\03-%20المتغيرات\2-%20تعريف%20المتغيرات\2-%20بعض%20الاختلافات%20عن%20VB6%20في%20التعامل%20مع%20المتغيرات.htm)
  + [**الشروط الواجب توافرها عند تعريف المتغيرات:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\03-%20المتغيرات\2-%20تعريف%20المتغيرات\3-%20الشروط%20الواجب%20توافرها%20عند%20تعريف%20المتغيرات.htm)

**لماذا نعرّف المتغيّر؟:**

**ملحوظة:**

لو أردت تجربة أيّ كود موجود في أمثلة هذا الفصل، فأنشئ تطبيقا جديدا، وضع على النموذج زرّا، وانسخ الكود إلى حدث ضغط هذا الزرّ.

في معظم لغاتِ البرمجة، يجبُ تعريفُ المتغيّراتِ أولا قبلَ استخدامها.. إنّ هذا يجعل الأمرَ أيسرَ بالنسبة لمترجم الكود Compiler، ففي كلّ مرّة يصادف المترجم متغيّرا، عليه أن يُنشئه في الذاكرة، ونتيجة لاعتباراتٍ في تنظيم الذاكرة، فإن مثل هذه العملية تستهلك بعض الوقت، ممّا يُبطئ البرنامج.. ولكن لو كان المترجم يعرف كل متغيرات البرنامج وأنواعها سلفا قبل أن يبدأ ترجمة البرنامج، ففي هذه الحالة سيتحسّن الأداء لأقصى درجة.

لقد كانت من أشهر سمات VB، عدم إرغامه للمبرمج على تعريف كلّ المتغيرات.. لقد صارت هذه السمة منتقدةً الآن بشدّةٍ، ليس فقط للأسباب المتعلّقة بسرعة الترجمة وكفاءة الأداء، ولكن أيضا لأن تعريف المتغيّر يُمكّن المترجم من اصطياد أخطاء كثيرةٍ، سواء في وقت التصميم Design Time أو وقت الترجمة Compile Time، بدلا من أن تُفاجئكَ في وقت التشغيل Runtime.

فمثلا: عندما تعرّف متغيرا من النوع "تاريخ" Date</span:، لا يمكن أن يسمح لك المترجم بوضع عدد صحيحInteger فيه.. أيضا لن يسمح لك المترجم باستخدام خاصية "شهر" Month الخاصة بالمتغيرات من النوع "تاريخ" Date، مع متغيّر من النوع "عدد صحيح" Integer.

إن تعريف المتغيّرات يمكّن المترجم من اصطياد مثل هذين النوعين من الأخطاء، أثناء كتابتك للكود، فلا تكاد تترك السطر الذي كتبتها به، حتّى يضع تحته خطًا متعرّجا، لو حلّقت فوقه بالفأرة، فسيظهر لك تلميح على الشاشة يصف الخطأ الذي ارتكبته.

لكل تلك الأسباب، فإنّ المترجم في VB.NET في الوضع التلقائيّ، لن يسمح لك باستخدام المتغيرات بدون تعريف، إلا إذا طلبت أنت ذلك منه صراحةً ـ كما سنرى فيما بعد ـ وذلك على عكس الوضع الذي كان في VB6.

مطلوب منك أيضا، تحديد نوع المتغيّر.. وللتسهيل، يمكنك استخدام رموز الأنواع التراثية في لغة البيزيك، مثل $ التي ترمز للنصوص.. فمثلا يمكنك كتابة:

**Dim Note$**

لتعريف المتغيّر Note كمتغيّر نصّيّ.. ولو حلّقت بالفأرة فوق هذا السطر في بيئة التطوير، لظهر لك تلميح على الشاشة مكتوب فيه التالي:

**Dim Note As String**

مّما يعني أنّ الجملتين متكافئتين تمامًا.

**بعض الاختلافات عن VB6 في التعامل مع المتغيرات:**

1- لم يعد مسموحا باستخدام الدوال التي تبدأ بالمقطع Def، مثل DefInt و DefDbl، لتحديد نوع المتغير الذي يبدأ بحروف معيّنة.

2- يمكنك الآن أن تعرّف أكثر من متغيّر من نفس النوع، بدون تكرار اسم النوع بعد كل متغيّر.. ففي الجملة:

**Dim Width, Depth, Height As Integer, Area, Volume As Double**

تمّ تعريف ثلاثة متغيرات من نوع العدد الصحيح Integer، ومتغيّرين من نوع العدد المزدوج Double.

3- يمكنك الآن أن تضع قيما ابتدائية للمتغيرات، مباشرةً في جملة التعريف.. مثال:

**Dim Width As Integer = 9**

**Dim Distance As Integer = 100, Time As Single = 9.09**

فإذا لم تضع هذه القيم، فإنّ VB يعطي المتغيّرات قيما ابتدائيّة افتراضيّة، كالصفر للمتغيرات الرقميّة، و False للمتغيّرات المنطقيّة.. ولكنّ ذلك ليس هو الوضع مع النصوص والحروف، فإذا لم تستخدم معها كلمة New أو إذا لم تضع بها أيّ قيمة، فإنّ قيمتها تكون Nothing.

**الشروط الواجب توافرها عند تعريف المتغيرات:**

- ألا يكون كلمة من كلمات اللغة الأساسيّة (تلك التي تراها باللون الأزرق).. مثل Sub وFor و If وغيرها.. إنّ الجملة التالية غير مقبولة:

**Dim Sub As Integer**

ولكن لو كنت مصرّا على مثل هذا الأمر، فيمكنك أن تضع الاسم بين قوسين مضلعين [].. هذه الجملة مقبولة:

**Dim [Sub] As Integer**

لكن عليك في كلّ موضع تستخدم فيه المتغيّر أن تحيطه بالقوسين المضلعين:

**[Sub] = 5**

- ألا يزيد عن 255 حرفًا، وهو رقم كبير بالفعل بما يكفي.

- أن يتكون من كلمة واحدة لا تتخللها المسافات.. ويمكن استخدام الشرطة المنخفضة "\_" للفصل بين مقاطع الكلمة بدلا من المسافات.

- لا يبدأ بأرقام، وإن كان من الممكن أن تتوسطه أرقام، أو ينتهيَ بها.

- لا يحتوي على أيٍ من: علامات التنصيص أو الأقواس أو النقطة "."، ولا علامات العمليات الحسابية أو علامات المقارنة الحسابية أو المنطقية، فكل هذه العلامات محجوزة لوظائف أخرى.

- غير مسموح بتكرار اسم المتغيّر داخل نفس النطاق، فلا يمكن تعريف متغيّرين متماثلين في الاسم داخل نفس الإجراء، وإن كان من الممكن تكرار نفس اسم المتغيّر لكن في إجراءات مختلفة.

والمتغيرات في لغة البيزيك تتجاهل حالة الأحرف Case-insensitive، فالأسماء myAge و myage و MYAGE، كلّها متكافئة، وتشير لنفس المتغيّر.. معنى هذا أنّك لا تستطيع استخدام هذه الكلمات لتعريف ثلاثة متغيرات مختلفة، فكلّها تُعتبر اسما واحدا.

**أنواع المتغيرات Types of Variables**

تنقسم أنواع المتغيرات إلى:

* [**المتغيرات الرقميّةNumeric Variables :**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\03-%20المتغيرات\3-%20أنواع%20المتغيرات\2-%20المتغيرات%20الرقميّة.htm)
* [**المتغيرات المنطقية Boolean Variables:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\03-%20المتغيرات\3-%20أنواع%20المتغيرات\3-%20المتغيرات%20المنطقية.htm)
* [**المتغيّرات النّصّيّة String Variables:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\03-%20المتغيرات\3-%20أنواع%20المتغيرات\5-%20المتغيّرات%20النّصّيّة.htm)
* [**المتغيّرات الحرفيّة Character Variables:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\03-%20المتغيرات\3-%20أنواع%20المتغيرات\6-%20المتغيّرات%20الحرفيّة.htm)
* [**المتغيرات الزمنيّة Date Variables:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\03-%20المتغيرات\3-%20أنواع%20المتغيرات\7-%20المتغيرات%20الزمنيّة.htm)
* [**المتغيّرات الكائنات Object:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\03-%20المتغيرات\3-%20أنواع%20المتغيرات\1-%20الكائنات.htm)

**المتغيرات الرقميّة Numeric Variables**

* + - [**أنواع المتغيّرات الرقميّة:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\03-%20المتغيرات\2-%20المتغيرات%20الرقميّة\0-%20أنواع%20المتغيّرات%20الرقميّة.htm)
    - [**المتغيّرات من النوع وحدة الذاكرة Byte**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\03-%20المتغيرات\2-%20المتغيرات%20الرقميّة\1-%20المتغيرات%20من%20النوع%20وحدة%20الذاكرة.htm)
    - [**العمليّات على الأرقام:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\03-%20المتغيرات\2-%20المتغيرات%20الرقميّة\2-%20العمليّات%20على%20الأرقام.htm)
    - [**اختبار الأعداد الصحيحة:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\03-%20المتغيرات\2-%20المتغيرات%20الرقميّة\3-%20اختبار%20الأعداد%20الصحيحة.htm)
    - [**اختبار الأعداد العشريّة:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\03-%20المتغيرات\2-%20المتغيرات%20الرقميّة\4-%20اختبار%20الأعداد%20العشريّة.htm)
    - [**المتغيرات العشريّة The Decimal Data Type:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\03-%20المتغيرات\2-%20المتغيرات%20الرقميّة\5-%20المتغيرات%20العشريّة.htm)
    - [**ما لا نهاية والقيم الشاذّة الأخرى Infinity And Other Oddities:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\03-%20المتغيرات\2-%20المتغيرات%20الرقميّة\6-%20ما%20لا%20نهاية%20والقيم%20الشاذّة%20الأخرى.htm)

**أنواع المتغيّرات الرقميّة:**

إنّ للمتغيرات الرقميّة أنواعًا عديدة، تبعًا لحجم العدد ودقّته العشريّة.. وعليك أنت أن تحّدد النوع الذي يناسبك، واضعًا في الاعتبار أنّ الأعداد ذات الدقّة العشريّة الأكبر، تكون العمليات عليها أبطأ من الأعداد الصحيحة والأعداد ذات الدقّة العشريّة الأقلّ.

وفي هذا الجدول، الأنواع التي تستطيع استخدامها لتخزين القيم الرقمية في VB:

|  |  |  |
| --- | --- | --- |
| النوع | حجمه بوحدات الذاكرة Bytes | الأعداد التي يقبلها |
| وحدة ذاكرة Byte | 1 | من 0 إلى 255. |
| وحدة ذاكرة ذات إشارة SByte | 1 | من -128 إلى 127. |
| قصير Short  Int16 | 2 | من –32768 إلى 32767 |
| صحيح Integer  Int32 | 4 | من -2147483648 إلى 2147483647 |
| طويل Long  Int64 | 8 | يقبل أرقاما كبيرة جدّا. |
| مفرد Single | 4 | للأعداد ذات الدقّة العشرية العاديّة.. وهو يمثّل الأعداد السالبة: من –3.402823E38 إلى –1.401298E–45 ، والأعداد الموجبة من 1.401298E–45 إلى 3.402823E38.  وفي هذا النوع، لا يمكن تمثيل الصفر بدقّة، فهو يمثّل دائما بقيمة صغيرة جدّا جدا، ولكنها أبدا ليست صفرا! |
| مزدوج Double | 8 | للأعداد ذات الدقّة العشرية الفائقة، وهو يستخدم في الحسابات العلميّة. |
| عشري Decimal | 16 | لتمثيل الأعداد الصحيحة والعشريّة، حيث يمكّنك من تحديد الدقّة العشريّة التي تريد التعامل معها، من 0 إلى 28 خانة عشريّة. |

**ملحوظة:**

الرمز E في عدد مثل 1.401298 E –45 هو طريقة الكمبيوتر لتمثيل الأس العشري، أي أنّ هذا الرقم يساوي: 1.40129 × 10 - 45.

**المتغيرات من النوع "وحدة الذاكرة" The Byte Data Type:**

ألا تظنّ أنّ الوقت قد حان لكي نعرف ما هو الـ Byte؟

تعرفُ طبعًا أنّ للجهاز ذاكرةً مؤقّتة RAM، لا يمكنها أن تحتفظ بالبيانات، إلا أثناء تشغيل الجهاز.. وأبسط طريقةٍ تتخيّل بها تركيب هذه الذاكرة، هي أنّها عبارة عن مجموعة هائلة من الخانات Bits، كل خانة منها يمكن شحنها بشحنة كهربيّة (ونرمز لها في هذه الحالة بالرقم 1)، أو تفريغ الشحنة الكهربيّة منها (ونرمز لها بالرقم 0).. المفروض الآن عندما تريد تخزين نوعٍ من البيانات، أن تحوّله إلى أعداد ثنائيّة Binary، تتكوّن من آحاد وأصفار، حيث سيقوم الكمبيوتر بتحويلها إلى شحنات كهربيّة وتخزينها في الذاكرة.

وطبعًا قد لاحظت، أنّ تمثيل الأعداد في النظام الثنائي يحتاج لخانات أكثر بكثير من تلك التي يحتاجها النظام العشري.

ولكن كتابة كل تلك السيول من البيانات التي يتعامل معها الكمبيوتر في كلّ ثانية خانةً بخانة سيجعل العملية بطيئة جدّا.. هنا وجد المصنّعون أنّ ضمّ الخانات في وحدات أكبر، سيوفّر الوقت ويجعل الجهاز أسرع.. ولقد ساد في فترة ما، جمع كل 8 خانات 8 Bits معًا، وأُطلق عليها اسم Byte.. ولقد ظلّ هذا المصطلح مستمرًّا حتّى الآن، حتّى بعد ظهور تقنيات أخرى، مكّنت الجهاز من أن يكتب ويقرأ كل 16 خانة معا كوحدة واحدة، ثمّ تطوّر الأمر إلى كتابة وقراءة كل 32 خانة معا (وهذا هو السبب في اقتران الرقم 32 باسم الويندوز)، وما زال التطوّر مستمرّا.

يمكنك الآن أن تدرك السبب في ترجمتنا لكلمة Byte بتعبير "وحدة الذاكرة".

جميل.. ولكن بماذا سيفيدنا كل هذا؟

لقد رأينا أنّ كل الأنواع السابقة تُخزّن في أكثر من وحدة من وحدات الذاكرة Bytes، ولكنّ هناك حالات أخرى، نحتاج فيها للتعامل مع وحدات منفردة 1 Byte، مثل التعامل مع الملفات الثنائيّة Binary files وملفات الصور Image files و ملفات الصوت Sound files.. في مثل هذه الحالات يمكنك استخدام متغيّرات من النوع Byte، حيث يمكنها أن تحمل أعدادا صحيحة موجبة، من 0 إلى 255.

**ملاحظة:**

لم يعد بإمكانك استخدام وحدة ذاكرة واحدة لتحمل حرفا Character، فالحروف من النوع Unicode يتمّ تخزينها في وحدتين Two Bytes.

ولتعريف متغير من هذا النوع استخدم جملةً كالتالية:

**Dim Var As Byte = 15**

**العمليّات على الأرقام:**

لن تكون هناك فائدة إذا كنت ستضع الأرقام في متغيّرات، دون أن تستطيع أن تُجريَ عليها بعض العمليّات الحسابيّة.. وفي الجدول التالي علامات العمليّات الحسابيّة الأساسيّة:

|  |  |
| --- | --- |
| + | علامة الجمع. |
| - | علامة الطرح. |
| \* | علامة الضرب. |
| / | علامة القسمة.. ويمكن أن يكون الناتج عددا صحيحا أو به أرقام عشريّة.. فمثلا:  X = 7 / 2  ستعطي الناتج 3.5. |
| \ | علامة القسمة أيضا، ولكنّ الناتج هو العدد الصحيح فقط.. فمثلا:  X = 7 \ 2  ستعطي الناتج 3.  ويمكن أداء نفس العمليّة باستخدام الدالة Int، لو شئت ألا ترتبك بين علامتي القسمة المتشابهتين، وذلك كالتالي:  X = Int (7/2) |
| Mod | إحدى علامات القسمة أيضا، ولكنّها تعطي الباقي من القسمة فحسب.. فمثلا:  X = 7 Mod 2  سيعطي الناتج 1، الذي هو عبارة عن باقي القسمة. |
| ^ | الأسّ.. فمثلا 2 × 2 × 2 تُكتب رياضيّا بالصيغة 2 3، وتكتب في البرمجة كالتالي:  2 ^ 3  ولو أردت أن تعبّر عن الجذر التكعيبيّ مثلا، فارفع العدد للأسّ (1 ÷ 3) كالتالي:  2 ^ (1/3) |

ويجب أن ألفت انتباهك إلى أهمّيّة وضع الأقواس في العمليّات المتداخلة، وذلك حتّى تضمن صحّة إجراء العمليّة بالترتيب الذي تريدها به.. إنّ الترتيب الطبيعيّ الذي يجرى به VB العمليّات الحسابيّة يسير تبعا للقواعد التالية:

- يتمّ تنفيذ ما بين الأقواس أولا.

- إذا لم تكن هناك أقواس يتمّ تنفيذ الأسس أولا.

- ثمّ يتمّ تنفيذ الضرب والقسمة.

- ثمّ بعد ذلك يتمّ تنفيذ الجمع والطرح.

لهذا فإنّ الصيغة:

8^(1/3)

تعطي الناتج 2، وذلك لأنّ القوس ينفّذ أولا، فتصبح العمليّة هي الجذر التكعيبيّ للعدد 8.. ولكن لو أزلت الأقواس كالتالي:

8^1/3

فإنّ الناتج سيكون 2.666666، وذلك لأنّ الأس ينفّذ أولا، فتصبح العمليّة كالتالي:

8/3

فانتبه لهذا جيّدا.

بقي شيءٌ هامّ.. ماذا لو أردت أن نزيد قيمة متغيّر بمقدار 1 مثلا؟

في هذه الحالة سنقوم بالتالي:

**X = 5**

**Y = X + 1**

**X = Y ' صارت قيمة المتغيّر 6**

حيث اعتمدنا على متغيّر وسيط، جعلنا قيمته هي ناتج جمع المتغيّر الأصليّ مع الواحد، ثمّ نقلنا قيمته إلى المتغيّر الأصلي.

ولكنّ مثل هذه العمليّة تتكرّر مرارا في البرمجة، حيث تحتاج مرارا لزيادة قيم المتغيّرات أو إنقاصها، أو ضربها في رقم... إلخ.

فلو كان على المبرمج أن يكتب هذه الخطوات في كلّ مرّة، لصارت البرمجة جحيما لا يُطاق!

لا تقلق.. يمكنك أداء هذه العمليّة في سطر واحد مباشرةً كالتالي:

**X = X + 1**

لأوّل وهلةٍ ستبدو لك الصيغة غريبة، ولكن حاول أن تقرأها كالتالي: قيمة X الجديدة تساوي قيمته القديمة + 1.

ولا يشترط أن أجمع على المتغير الرقم 1 فحسب، فهذه العمليّات أيضا مباحة:

**X = X + 13**

**X = X + X**

**Y = 5**

**X = X + Y**

ولا يقتصر الأمر على الجمع فحسب، بل يمتدّ إلى باقي العمليّات الحسابيّة:

**X = X – 4 'إنقاص المتغيّر بمقدار 4**

**X = X \* 2 ' ضرب المتغيّر في 2**

**X = X / 9 ' قسمة المتغيّر على 9**

**X = X ^ 3 ' رفع المتغيّر للأس 3**

كان هذا هو ما اعتاده مبرمجو VB6.. ولكنّ هناك تسهيلا إضافيّا تقدّمه لك VB.Net، عن طريق استخدام الرموز += و -= ، \*=، /=، ^=.. والجدول التالي يريك كيفيّة استخدامها:

|  |  |
| --- | --- |
| الطريقة التقليديّة | الطريقة المختصرة المكافئة |
| **X = X + 1** | **X + = 1** |
| **X = X + Y** | **X += Y** |
| **X = X – 4** | **X -= 4** |
| **X = X \* 2** | **X \*= 2** |
| **X = X / 9** | **X /= 9** |
| **X = X ^ 3** | **X ^= 3** |

وأنت حرّ في اختيار الصيغة التي تريحك.

**ملحوظة:**

عند استخدام الأس ^ في أيّ عمليّة، تعتبر اللغة أنّ الناتج سيكون Double، وذلك لأنّ هذه العمليّة في الغالب تُعيد أرقاما كبيرة.. لهذا فإنّ الأدقّ أن تكتب الصيغ التي بها أسس كالتالي:

**Dim X As Integer = 3**

**X = CInt (X^2)**

حيث استخدمنا دالة التحويل إلى عدد صحيح CInt، وذلك حتّى لا يعترض VB إذا كان خيار التحويل الدقيق فعالا Option Strict On.. وللأسف لا توجد صيغة مختصرة للعمليّة في هذه الحالة!

**اختبار الأعداد الصحيحة:**

والآن، استخدم الكود التالي لاختبار أنواع المتغيرات.. ستجد أنّنا نستخدم خاصيّة "أصغر قيمة" MinValue، و"أكبر قيمة" MaxValue، لعرض نطاق كل نوع منها.

**Dim ShortInt As Int16**

**Dim Int As Int32**

**Dim LongInt As Int64**

**Console.WriteLine(ShortInt.MinValue)**

**Console.WriteLine(ShortInt.MaxValue)**

**Console.WriteLine(Int.MinValue)**

**Console.WriteLine(Int.MaxValue)**

**Console.WriteLine(LongInt.MinValue)**

**Console.WriteLine(LongInt.MaxValue)**

**' الجملة التالية ستتسبّب في خطإ تجاوز أقصى قيمة للمتغير**

**ٍShortInt = ShortInt.MaxValue + 1**

**' لكن الجملة التالية لن تسبّب أي خطأ**

**Int = ShortInt.MaxValue + 1**

**' الجملة التالية ستتسبّب في خطإ تجاوز الحد الأدنى للمتغيّر**

**Int = Int.MinValue – 1**

**اختبار الأعداد العشريّة:**

استخدم هذا الكود لاختبار الدقّة العشريّة للعدد المفرد Single:

**Dim A As Single**

**A = 1 / 3**

**Console.WriteLine(A)**

ستجد أن نافذة المخرجات Output window ستعرض لك النتيجة التالية: 0.3333333، حيث تلاحظ وجود 7 خانات عشريّة.

والآن عدّل الكود للتالي:

**Dim A As Single**

**A = 100000/ 3**

**Console.WriteLine(A)**

ستجد أنّ الناتج هو 33333.34، حيث يحتوي على خانتين عشريّتين فحسب، ممّا لا يمنحك الدقّة التي تريدها، بل إنّ الرقم ليس مقرّبا حتّى بطريقة صحيحة، فالمتوقّع أن يكون الناتج 33333.33 !!.. مثل هذا الخطإ يمكن أن يؤدّى إلى ما يسمّى بتراكم الخطإ Error Propagation، حيث قد تنشأ أخطاء ملموسة عند إجراء مجموعة من العمليات المتتابعة على الأعداد المفردة.

والآن دعنا نقوم بنفس العمليتين السابقتين، ولكن مع استخدام المتغير المزدوج Double:

**Dim B As Double**

**B = 1 / 3**

**Console.WriteLine(B)**

**B = B \* 100000**

**Console.WriteLine(B)**

ستجد أنّ شاشة المخرجات ستعرض النتيجتين كما يلي:

0.333333333333333

33333.3333333333

واضح طبعا أنّ النتيجتين أكثر دقّة.

وإليك مثال آخر، لتوضيح الفرق في دقّة تمثيل الأعداد العشريّة، بين النوعين المفرد والمزدوج:

**Dim A As Single, B As Double**

**A = 0.03007**

**B = 0.03007**

**Console.WriteLine(A - B)**

ستفاجأ بأن الناتج لن يكون صفرا، وبدلا من ذلك ستعرض نافذة المخرجات الرقم:

6.03199004634014E-10

الذي يكافئ 0.000000000603199004634014 .. واضح أنّه رقم صغير جدا، بما يكفي لاعتباره صفرا، ولكنّه ليس صفرا!

إنّ ذلك الخطأ يرجع لاختلاف الطريقة التي يتمّ بها تمثيل المفرد والمزدوج في الذاكرة، ممّا يجعلهما غير متماثلين، حتّى ولو حملا نفس الرقم!

لهذا يجب أن تحتاط للتالي:

- لا تُجرِ عملياتك الحسابيّة على متغيّرين من نوعين مختلفين في الدقّة العشريّة.

- لا تحاول مقارنة عددين من نوعين مختلفين في الدقّة العشريّة بجملة كالتالية:

**If A = B Then**

لأنّك لن تضمن أداء البرنامج في هذه الحالة.. والأفضل أن تستخدم جملةً كالتالية:

**If (A - B) < 0.000001 Then**

فلو كان الفارق بين العددين أصغر من حدّ معيّن، يمكنك اختياره على حسب برنامجك، ففي هذه الحالة يمكن اعتبار العددين متساويين.

**المتغيرات العشريّة Decimal Data Type:**

لو أردت أن تضرب رقمين عشريين مثل: 328.558 × 12.4051، فإنّ أوّل خطوة، هي تحويلهما إلى عددين صحيحين، مع تذكّر عدد الخانات العشريّة.. أيّ أنّ العملية تتحوّل إلى: 328558 × 124051، والتي ستعطي النتاج: 40757948458.. بعد ذلك نضع العلامة العشرية بعد سبع خانات، فيكون الناتج النهائيّ هو 4075.7948458.

هذه هي الطريقة التي يتعامل بها VB مع الأعداد العشريّة Decimal.. تعال نجرّب ذلك:

**Dim A As Decimal = 328.558**

**Dim B As Decimal = 12.4051**

**Dim C As Decimal**

**C = A \* B**

**Console.WriteLine(C)**

ستجد أنّك ستحصل على الناتج الحقيقيّ، بدون أيّ أخطاء في التقريب، بينما لو كنت استخدمت متغيرات مفردة Single كنت ستحصل على الناتج 4075.795، وهو ناتج مقرّب ذو دقّة عشريّة صغيرة.

ولكن بالرغم من دقّة الأرقام العشريّة، إلا إنّك يجب أن تستخدم الأعداد المزدوجة في الحسابات العلميّة.. أمّا لو كنت تجري حساباتٍ ماليّة، فاستخدم الأعداد العشريّة، وقرّب الناتج لأقرب خانتين عشريتين في النهاية.

**ما لا نهاية والقيم الشاذّة الأخرى Infinity And Other Oddities:**

يستخدم VB في الحسابات الرقميّة القيمتين "ليس رقمًا" NaN (التي هي اختصار لتعبير **N**ot **a** **N**umber)، و "ما لا نهاية" Infinity، وذلك لإعلامك بأنّ شيئا ليس على ما يرام قد تمّ في حساباتك، لتتّخذ التصرّف المناسب، بدلا من أن يعرض لك VB رسالة خطإ كما كان يحدث في الماضي.

**ما لا نهاية Infinity:**

إنّ بعض العمليات الحسابيّة ـ كالقسمة على صفر ـ تعطي ما لا نهاية.. ولو استدعيت الدالة "حوّل إلى نصّ" ToString على هذه القيمة، لكان الناتج هو النص "Infinity".. فلْنَرَ هذا المثال:

**Dim DblVar As Double = 999**

**Dim InfVar As Double**

**InfVar = DblVar / 0**

**MsgBox(InfVar.ToString)**

سيظهر لك النصّ "Infinity" في الرسالة.

سببٌ آخر للما لا نهاية، هو قسمة عدد كبير جدًّا على عدد صغير جدا جدا، لدرجة تجعل الناتج يتجاوز حدود المتغيّر المزدوج.. جرّب هذا المثال:

**Dim LargeVar As Double = 1E299**

**Dim SmallVar As Double = 1E-299**

**Dim Result As Double**

**Result = LargeVar / SmallVar**

**MsgBox(Result)**

أيضا سيظهر لك النصّ "Infinity" في الرسالة.

وإذا ما عكست هذه العمليّة، بقسمة العدد الصغير جدا على العدد الكبير جدا، فسيكون الناتج صفرا.. الواقع أنّ الناتج سيكون عددا صغيرا صغيرا جدا جدا، ولكنّ المتغير المزدوج لا يستطيع التعامل مع مثل هذه الأعداد القريبة جدا من الصفر.

**ليس رقما NaN:**

هذه القيمة توضّح أن ناتج عمليّة ما ليس معرّفا: ليس رقما، ولا صفرا، ولا حتّى ما لا نهاية.

فمثلا، ستنتج لك هذه القيمة، لو حاولت أن تحسب اللوغاريتم لعدد سالب، أو لو حاولت أن تقسم صفرا على صفر (0/0)، فهما عمليّتان غير معرّفتان رياضيّا، وليس لهما معني.. جرّب هذا المثال:

**Dim Var1, Var2 As Double**

**Dim Result As Double**

**Var1 = 0**

**Var2 = 0**

**Result = Var1 / Var2**

**MsgBox(Result)**

ستظهر لك رسالة تعرض النص: "NaN".. لاحظ أنّك لو جعلت قيمة Var2 صغيرة جدّا مثل 1E-299، فإنّ الناتج سيكون صفرا.. ولو جعلت قيمة Var1 صغيرة جدّا، فسيكون الناتج ما لا نهاية.

وطبعا لو استخدمت متغيّرا يحمل القيمة NaN، في أيّ عملية حسابية، فإنّ الناتج سيكون أيضا NaN.. أضف هذا الكود للمثال السابق:

**Result = Result + Result**

**Result = 10 / Result**

**Result = Result + 1E299**

**MsgBox(Result)**

سيكون الناتج هو الرسالة "NaN" أيضا.

**اختبار وجود "ما لا نهاية" و" ليس رقما" Testing For Infinity And NaN:**

الأمر بسيط: للتحقّق من هاتين القيمتين، استخدم الدالتين: "إنّه ليس رقما" IsNaN و"إنّها ما لا نهاية" IsInfinity، اللتين تجدهما من أعضاء المتغيرات المفردة والمزدوجة.

ولمعرفة إشارة الما لا نهاية، استخدم الدالتين: "إنها ما لا نهاية سالبة"IsNegativeInfinity و" إنها ما لا نهاية موجبة" IsPositiveInfinity.

وإليك مثال توضيحيّ:

**Dim Var1, Var2 As Double**

**Dim Result As Double**

**Var1 = 0**

**Var2 = 0**

**Result = Var1 / Var2**

**If Result.IsInfinity(Result) Then**

**If Result.IsPositiveInfinity(Result) Then**

**MsgBox("هذا رقم كبير جدا.. لا يمكن الاستمرار")**

**Else ' ما لا نهاية سالبة**

**MsgBox("هذا رقم صغير جدا.. لا يمكن الاستمرار")**

**End If**

**ElseIf Result.IsNaN(Result) Then**

**MsgBox("خطأ غير متوقع في الحسابات")**

**Else 'لم يحدث أي خطأ**

**MsgBox("النتيجة هي: " & Result.ToString)**

**End If**

هذا الكود سيعرض الرسالة "ليس برقم" NaN.. ولكن لو غيّرت قيمة Var1 إلى 1، فستنتج ما لا نهاية موجبة، ولو غيرتها إلى -1، فستنتج ما لا نهاية سالبة.

**ملاحظة:**

رغم أن الدوال IsInfinity و IsPositiveInfinity وIsNegativeInfinity و IsNaN هي من أعضاء المتغيّر Result، إلا إنّها لا تنطبق عليه، لهذا اضطررنا لإرسال المتغير Result كمعامل لها، ليتم إجراء الدالة عليه.. هذا الأمر يقتضي أنّ الصيغة التالية هي الأكثر منطقيّة للتعامل مع هذه الدوالّ:

**System.Double.IsInfinity (Result)**

وذلك بالتعامل مع أيّ من هذه الدوال من خلال الخليّة الأساسيّة Base Class للمتغيّر.

ولكن الصيغة السابقة قد تبدو طويلة.. لا تقلق.. يمكنك أن تكتبها مختصرة وأنت مطمئنّ، كالتالي:

**Double.IsInfinity (Result)**

**تعريف:**

الدوال التي لا تُطبّق على الكائن الذي تنتمي إليه (بدون إرساله كمعامل بين قوسيها)، هي دوال مشتركة "Shared"، يمكن استخدامها كذلك من الخليّة الأساسيّة Base Class، وذلك ما فعلناه مع الدالة IsInfinity.

**المتغيرات المنطقية Boolean Variables:**

المتغيرات المنطقيّة تخزّن واحدةً فقط من القيمتين: "صواب" True و"خطأ" False، وهي في الأساس أعداد صحيحة، فالقيمة "صواب" تعادل -1، والقيمة "خطأ" تعادل صفرا.. وفي الواقع، أي قيمة غير صفرية، تعتبر True.

ويمكنك تعريف المتغيرات من هذا النوع، بجملة كالتالية:

**Dim Failure As Boolean**

ولو لم تحدّد قيمةً ابتدائيّة، فستكون قيمة المتغيّر المبدئيّة False.

ولكن كيف نضع القيم في المتغيّر Failure؟.. إنّ كلّ التعبيرات التالية متاحة:

**Dim Failure As Boolean = True**

**Failure = False**

**Failure = CheckFailure() ' دالة تعيد قيمة منطقيّة**

ولا يتوقّف الأمر عند هذا الحدّ، بل يمكن أن تضع في المتغيّر Failure نتيجة أيّ عمليّة مقارنة، كالتالي:

**Failure = TextBox1.Text = ""**

أعرف أنّ هذا التعبير قد يكون غريبا عليك.. تعال نعيد كتابته مرّة أخرى مع وضع قوسين، حتّى يسهل علينا فهمه:

**Failure = (TextBox1.Text = "")**

الآن سيفعل VB ما يأتي: سيتحقّق أولا من صحّة العلاقة الموجودة بين القوسين، فإذا كانت صحيحة، يضع القيمة True في المتغيّر Failure، وإن كانت خاطئة، يضع القيمة False في المتغيّر Failure.

ولكي تدرك مدى أهمّيّة هذه الصيغة، تعال نكتبها بالطريقة التقليديّة المكافئة لها:

**If TextBox1.Text = "" Then**

**Failure = True**

**Else**

**Failure = False**

**End If**

واضح جدّا أنّ الصيغة الأولى تختصر خمسة أسطر في سطر واحد فقط!

وعلى هذا، لو أردت أن تفعّل أو تمنع تفعيل أحد الأزرار، تبعا لما إذا كان أحد مربّعات النصّ به كلمات أم فارغا، فاكتب السطر التالي في حدث تغيّر حروف مربّع النصّ:

**Private Sub TextBox1\_TextChanged(ByVal sender As Object, ByVal \_**

**e As System.EventArgs) Handles TextBox1.TextChanged**

**Button1.Enabled = TxtElem.Text <> ""**

**End Sub**

وتُستخدم المتغيرات المنطقيّة في اختبار الشروط، كالتالي:

**If Failure Then MsgBox("لا يمكن إكمال العمليّة")**

ويمكن تكوين علاقة بين متغيرين منطقيّين أو أكثر، باستخدام العمليات المنطقيّة "و" AND، "أو" OR ، "ليس" NOT، "أو الحصريّة" XOR:

**If Failure = True And TextBox1.Text <> "تجاهل الخطأ" Then Exit Sub**

ولا تستخدم المعاملات And و Or و Not مع المتغيّرات المنطقيّة فقط، بل تستخدم كذلك في العمليّات الثنائيّة Bit Wise Operations.. هنا لا بدّ لك من وقفة، لتتعرّف على نظام العدّ الثنائي Binary.. عليك بقراءته في ملحق 2.

**المتغيّرات النّصّيّة String Variables**

* + - [**تعريف المتغيّرات النّصّيّة:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\03-%20المتغيرات\5-%20المتغيّرات%20النّصّيّة\1-%20تعريف%20المتغيّرات%20النّصّيّة.htm)
    - [**تشبيك النصوص Concatenation:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\03-%20المتغيرات\5-%20المتغيّرات%20النّصّيّة\2-%20تشبيك%20النصوص.htm)
    - [**مقارنة النصوص:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\03-%20المتغيرات\5-%20المتغيّرات%20النّصّيّة\3-%20مقارنة%20النصوص.htm)

**تعريف المتغيّرات النّصّيّة:**

يتم تعريفها كالتالي:

**Dim SomeText As String**

الآن يصبح بإمكانك أن تضع أيّ نصّ، مها كانت مكوناته (حروف، رموز، أرقام، علامات تنسيق... إلخ)، ومهما كان طوله (يمكن أن يصل إلى 2 جيجا بايت، أي 2 مليار حرف!).

وهذه الجمل تُريك كيف تستخدم المتغيّر النصّيّ:

**Dim AString As String**

**AString = "اكتب ما تريد في هذا المتغيّر"**

**AString = "" ' متغير نصّي فارغ**

**AString = "ولكن بإمكانك أن تكتب بهذا المتغيّر نصا جديدا أطول من القديم"**

**' هذا نص يحتوي على قيمة رقميّة، ويمكن استخدام دوال التحويل**

**' للتعامل معها كرقم وليس كنص**

**AString = "25000"**

طبعا لاحظت أنّ النصّ يوضع بين علامتي تنصيص.. وهذا بالتأكيد سيدفعك للتساؤل: ماذا لو أردت أن أضع علامة التنصيص نفسها في متغيّر؟

في هذه الحالة يجب أن تكتب أربع علامات تنصيص كالتالي:

**Dim AString As String = """"**

وهناك حلّ آخر، هو استخدام الدالة Chr للتعبير عن علامة التنصيص كالتالي:

**Dim AString As String = Chr(34)**

سؤال آخر سيراودك: ما الفارق بين التعبيرين التاليين:

**Dim ANumber As Integer = 25000**

**Dim AString As String = "25,000"**

إن كلا المتغيرين يحمل قيمة مختلفة عن الآخر:

1- فالمتغير النصّيّAString يحتوي على ستة حروف، وهي "2" و "5" و "," و "0" و "0" و "0"، بينما العدد الصحيح ANumber يحتوي على رقم واحد هو 25000.

2- بما أن كل حرف يتم تخزينه في وحدتي ذاكرة 2 Bytes، فإن المتغيّر النصّيّ يخزن حروفه السّتّة في 12 وحدة ذاكرة، بينما يخزّن العدد الصحيح العدد 25000 في 4 وحدات فقط.

ولكن برغم هذه الخلافات، فإنّ بإمكانك أن تستخدم المتغيّر النصّيّ AString في العمليات الحسابيّة، وتستخدم المتغيّر الرقميّ ANumber في العمليّات النّصّيّة، حيث يقوم VB بالعمليات اللازمة للتحويل بين النوعين (ما لم تمنعه من القيام بذلك، باستخدام جملة Option Strict On).

**ملحوظة لمبرمجي VB6:**

لم يعد بإمكانك تعريف متغيّر نصّيّ ذي طول ثابت Fixed-length String.

**تشبيك النصوص Concatenation:**

ماذا لو أردت أن تلحم نصيّن، بحيث ينتج نصّ جديد يجمع الاثنين معا؟

بسيطة.. استخدم علامة الجمع "+" كما اعتدت في استخدامها بين المتغيّرات الرقميّة:

**Dim X As String = "محمد "**

**X = X + "حمدي"**

**Dim Y As String = " غانم"**

**Y = "أعدّ هذا الكتاب " + X + Y**

وحتّى لا تشعر النصوص بالغيرة من الأرقام، فإنّ بإمكانك استخدام الصيغة المختصرة التالية:

**X += "حمدي"**

ولو كنت تخشى من الارتباك في قراءة الكود بين الجمع الحسابيّ وتشبيك النصوص، فاستخدم العلامة & بدلا من العلامة +:

**Dim X As String = "محمد "**

**X = X & "حمدي"**

**Dim Y As String = " غانم"**

**Y = X & Y**

**Y = "أعدّ هذا الكتاب " & X & Y**

وأيضا هناك صيغة مختصرة تدخل فيها العلامة & كالتالي:

**X &= "حمدي"**

**مقارنة النصوص:**

يمكنك مقارنة تساوي نصّين عن طريق علامة "=" كالتالي:

**Dim X As String = "محمد", Y As String = "محمد"**

**If X = Y Then**

**MsgBox ("النصّان متساويان")**

**End If**

كما يمكن استخدام علامة عدم التساوي "<>" (وهي عبارة عن علامة < تليها علامة >) للتأكّد من اختلاف نصّين، كالتالي:

**Y = "أحمد"**

**If X <> Y Then MsgBox ("النصّان مختلفان")**

والطريف أنّك تستطيع مقارنة النصّين باستخدام علامتى "<" و ">" وذلك لتعرف من منهما يسبق الآخر في الترتيب الأبجديّ، حيث يكونُ الأسبق منهما في الترتيب الهجائيّ هو النصّ الأصغر:

**If** Y < X **Then** MsgBox("أحمد يسبق محمد أبجديّا")

كما يمكنك استخدام المعامل "يشبه" Like لمعرفة إذا ما كان النصّ يحتوي على صيغة Pattern معيّنة أم لا.. انظر للمثال التالي:

**Dim S As String = "هذا نصّ للتجريب"**

**If S Like "\*نص\*" Then MsgBox ("الكلمة موجودة")**

لاحظ استخدامنا للعلامة "\*"، وهي تقوم بديلا عن أيّ عدد من الحروفِ مهما كانت هذه الحروف، ومهما كان عددها (يمكن أن يكون هذا العدد صفرا!).

ويمكن استخدام علامة الاستفهام الإنجليزيّة "?" كبديل لحرفٍ واحدٍ فقط مهما كان هذا الحرف، والعلامة # كبديل للأرقام (من 0 إلى 9).

انظر للأمثلة التالية:

**Dim S As String = "هذا نصّ للتجريب"**

**' عدم وجود علامات خاصّة يكافئ عمليّة التساوي**

**If S Like "هذا" Then MsgBox ("الكلمة موجودة") ' لن تظهر الرسالة**

**' هل كلمة "هذا" تبدأ من الحرف الثاني في الصّ**

**If S Like "!هذا" Then MsgBox ("الكلمة موجودة") ' لن تظهر الرسالة**

**' هل كلمة هذا هي آخر كلمة في النصّ**

**If S Like "\*هذا" Then MsgBox ("الكلمة موجودة") ' لن تظهر الرسالة**

**' هل يبدأ النصّ بكلمة هذا وينتهي بكلم للتجريب**

**If S Like "هذا\*للتجريب" Then MsgBox ("الكلمة موجودة") ' ستظهر الرسالة**

**' هل يحتوى النصّ على حرف نون يليه حرفٌ مشدّد**

**If S Like "\*ن!ّ\*" Then MsgBox ("الكلمة موجودة") ' ستظهر الرسالة**

كما يمكن استخدام قوسين مضلّعين [ ] لنعبّر عن حرفٍ واحد من الحروف الموجودة بين القوسين.. هذه الحروف إمّا أن تكتب متتالية مثل [ACdF]، وإمّا أن تكتب على صورة مجال، مثل [g-y] حيث تعبّر هذه الصيغة عن الحروف من g إلى y.. فمثلا المقارنة التالية صحيحة، لأنّ النصّ يبدأ بحرف الهاء، وهي محصورة بين حرفي الميم والياء:

**If S Like "[م-ي]\*" Then MsgBox ("الكلمة موجودة") ' ستظهر الرسالة**

ولو أردت أن تتأكّد أنّ النصّ لا يحتوي على أيّ حرف من تلك الموجودة بين القوسين، فضع في بداية القوسين علامة ! .. وفي المثال التالي ستكون المقارنة خاطئة، لأنّ النصّ يبدأ بحرف الهاء، وهي محصورة بين حرفي الميم والياء، وهو عكس المطلوب:

**If S Like "[!م-ي]\*" Then MsgBox ("الكلمة موجودة") ' لن تظهر الرسالة**

**ملحوظة:**

إذا أردت البحث عن أيّ علامة من العلامات الخاصّة بالمعامل Like في النصّ، فضعها بين قوسين مضلّعين، مثل استخدام الصيغة '\*[?]\*' للبحث عن علامة الاستفهام الإنجليزية في أيّ موضع من النصّ.

**المتغيّرات الحرفيّة Character Variables:**

هي متغيّرات طولها وحدتا ذاكرة 2 Bytes، يمكن أن تخزّن فيها حرفا واحدا.. ونظرا لأنّ الكمبيوتر لا يعرف شيئا غير الأرقام، وأي شيء يُحفظ به يجب أن يتمّ تمثيله بأرقام تدلّ عليه، فإن الكمبيوتر يمثّل الحروف بأرقام من النوع "عدد قصير بدون إشارة" Unsigned Short Integers (UInt16).. فمثلا الحرف "a" يمثّله الرقم 65.

لهذا فإنّ بإمكانك تحويل الأعداد الصحيحة إلى حروف، باستخدام الدالة Chr()، وتحويل الحروف إلى أرقام، باستخدام الدالة CInt().. جرّب ما يلي:

**Console.WriteLine(CInt("a"))**

ستجد أن نافذة المخرجات تعرض الرقم 65.

ويمكنك استخدام جملة كالتالية لتعريف متغيرات حرفيّة:

**Dim Char1, Char2 As Char**

**ملاحظة:**

**لا تستطيع مقارنة حرف بعدد مباشرة.. فجملة كالتالية هي جملة مرفوضة:**

**If Char1 = 5 Then Char1 = 40**

**والصحيح أن تستخدم الجملة التالية:**

**If Char1 = Chr(5) Then Char1 = Chr(40)**

وبإمكانك أن تضع قيمةً ابتدائيّةً في المتغيّر الحرفيّ عند تعريفه، سواء أكانت حرفا أم نصّا (في الحالة الأخيرة سيتم وضع أول حرف فقط في المتغيّر).. جرّب ما يلي:

**Dim Char1 As Char = "a", Char2 As Char = "ABC"**

**Console.WriteLine(Char1)**

**Console.WriteLine(Char2)**

ستجد أن شاشة المخرجات ستعرض ما يلي:

a

A

ولكن الطريقة السابقة لن تعمل إذا كان اختيار "التحويل الدقيق" فعالا Option Strict On، لهذا يجب استخدام الطريقة الآتية للتعبير عن الحروف:

**Dim Char1 As Char = "a"c**

حيث يخبر الحرف c، VB أنّ ما بين علامتي التنصيص حرف وليس نصا.. وفي هذه الحالة يجب أن يكون ما بينَ علامتي التنصيص حرف واحد بالضبط، حيث لن يقبل VB جملةً كالتالية:

**Dim Char1 As Char = "abc"c**

إن وضع حرف c بجوار علامتي التنصيص هو طريقة VB لكي يرمز للحروف، ويمكن استخدامها في أيّ موضع كالتالي:

**Dim Char1 As Char = "a"c**

**If Char1 = "b"c Then exit sub**

والمتغيّر الحرفيّ يمتلك بعض الوسائل الشيّقة، مثل "إنّه حرف" IsLetter و"إنّه رقم" IsDigit و"إنّه علامة ترقيم" IsPunctuation... إلخ.

فلو جرّبت الجملة التالية:

**Console.WriteLine(Char.IsDigit("2"))**

لظهر لك في نافذة المخرجات كلمة "True".

جرّب أيضا ما يلي:

**Dim A As Char = "2"**

**Console.WriteLine(A.IsDigit(a))**

ستحصل على نفس النتيجة.. طبعا يمكنك أن تلاحظ أن الدالة IsDigit هي دالة مشتركة Shared، يمكن استخدامها من المتغير الحرفي، أو من خليّة النوع الأساسيّة Char Class.. بل حتّى يمكن استخدامها من أيّة دالة ناتجها حرف.. جرّب المثال التالي:

**Dim X As String = "M2f"**

**Console.WriteLine(X.Chars(1).IsDigit(X, 1))**

ستحصل على نفس النتيجة، وهي True.. ولكننا نحتاج لبعض الإيضاح:

فالدالة "حروف" Chars تأخذ رقم الحرف الذي تريد معرفته في النص، وتُرجع لك هذا الحرف.. هذا مع ملاحظة أن الحرف الأول في النص يوجد في الموضع رقم 0.. إلى هنا يمكن التوقّف، حيث يمكن استخدام جملة كالتالية:

**Console.WriteLine(X.Chars(1))**

حيث سيظهر لك في نافذة المخرجات ما يلي:

2

حيث إنّ "2" هو الحرف الموجود في الموضع 1 (الحرف الثاني) في النصّ.

ولكن بما أنّ الدالة Chars() تُرجع حرفا، إذن فيمكن تطبيق وسائل الحروف عليها، كالتالي:

**Console.WriteLine(X.Chars(1).IsDigit(X, 1))**

وتلاحظ أنّنا لم نرسل حرفا لدالة "إنّه رقم" IsDigit كما فعلنا من قبل، ولكن أرسلنا لها المتغيّر النصّيّ X، وموضع الحرف فيه، الذي نريد اختبار كونه رقما أم لا.. ولا بأس في هذا، فالدالة IsDigit لها تعريفان، يختلفان فقط في نوعية المعاملات التي تقبلها الدالة: أحدهما يمكن أن ترسل فيه معاملا واحدا للدالة، هو الحرف الذي تريد اختباره، والآخر ترسل فيه معاملين للدالة: أحد النصوص وموضع الحرف المراد اختباره فيه.

**ملاحظة:**

القدرة على تعريف أكثر من صورة للدالة تختلف في المعاملات (إما في عدد المعاملات أو في نوعها أو في كليهما) تسمّى "زيادة التحميل" Overloading، وسنترجمها بعد ذلك "تعدد التعريفات".

أي أنّنا يمكن أن نكتب صيغة أخرى للجملة السابقة كالتالي:

**Console.WriteLine(X.Chars(1).IsDigit(X.Chars(1)))**

حيث أرسلنا الحرف للدالة.. هذا الحرف لم نرسله مباشرة، بل حصلنا عليه من الدالة Chars.. طبعا ستبدو لك الجملة السابقة غبيّة، لتكرار الدالة Chars مرتين، وهو ما لا ينصح به.. معك حق، والأفضل أن تستخدم الجملة التالية:

**Console.WriteLine(Char.IsDigit(X.Chars(1)))**

ولكننا فقط كنا نريد توضيح قدرتك على استخدام وسائل المتغيّر الحرفيّ من أماكن أخرى غير المتغيّرات الحرفيّة نفسها.

**قاعدة هامّة:**

كل دالة لها قيمة معادة Return Value، هذه القيمة لها نوع ما، رقميّ أو نصّيّ أو حرفيّ أو منطقيّ أو كائن من أي نوع.. أبشر إذن: بمجرد كتابة نقطة بعد قوس الدالة، سيظهر لك قائمة بكل خصائص ووسائل نوع القيمة المعادة، حيث يمكنك استخدامها مباشرة.. مثل هذه الطريقة تمكّنك من كتابة جمل معقّدة جدّا، فلو لديك متغيّر نصّيّ مثلا، يمكنك استدعاء إحدى دواله، ولتكن قيمتها المعادة حرفا، حيث يمكنك استدعاء إحدى دوال الحروف منها، ولتكن قيمتها المعادة منطقيّة (صواب أو خطأ)، حيث يمكنك استدعاء إحدى دوال المتغيرات المنطقيّة منها، .... وهكذا..

انظر إلى الجملة التالية:

**Console.WriteLine(X.Chars(1).IsDigit(X, 0). \_**

**ToString.Chars(1).IsDigit(""))**

تلك الجملة المكلكعة متاحة الآن في VB، وهي تطبع كلمة "False" في شاشة المخرجات.. هل تريد أن تفهمها؟.. أنت الجاني على نفسك!

في البداية نستدعي دالة "حروف" Chars الخاصة بالمتغيّر النصّي X، للحصول على الحرف الثاني في النصّ (وهو "2").. هذا الحرف سنستدعى منه الدالة "إنّه رقميّ" IsDigit، حيث ستنتج عنها القيمة "صواب" True، التي سنستدعي منها الدالة "حوّل إلى نصّ" ToString، لتحويلها إلى النصّ "True".. هذا النصّ سنستدعي منه الدالة "حروف" Chars، لنحصل على الحرف الثاني (وهو "r").. هذا الحرف سنستدعي منه الدالة "إنّه رقميّ" IsDigit، حيث سنختبر بها هل النص الفارغ "" هو رقم أم لا.. طبعا لا.. لهذا سيكون الناتج "خطأ" False!

طبعا هذا نوع من العبث على غرار "من أين تصل لأذنك يا جحا؟"، ولكنّه عبثٌ يوضّح الإمكانيات الجديدة للغة.. أمّا لو شئت الجدّ، فإنّ كل الكلكعة السابقة يمكن اختصارها إلى الجملة التالية:

**Console.WriteLine(Char.IsDigit(""))**

والتي ستؤدّي نفس وظيفة الجملة المكلكعة السابقة بمنتهى الدّقّة.. ومن أقصر الطرق للوصول إلى أذن جحا!

**المتغيرات الزمنيّة Date Variables:**

يتم تخزين متغيّرات التاريخ والوقت بتنسيق خاص كأعداد مزدوجة، بحيث يمثّل العدد الصحيح التاريخ، ويمثّل الجزء العشريّ الزمن.

ويتم تعريف هذا النوع من المتغيّرات والكتابة فيه بطرق كالتالية:

**Dim Expiration As Date**

**Expiration = #01/01/2004#**

**Expiration = #8/27/2001 6:29:11 PM#**

**Expiration = "July 2, 2002"**

**Expiration = Now() ' دالة "الآن" تعطيك التاريخ والوقت الحاليّين.**

تلاحظ أنّنا نكتب التاريخ بطريقتين: إما كتاريخ رقميّ مكتوب بين علامتي ##، أو كتاريخ نصّيّ مكتوب بين علامتي تنصيص ""، حيث سيتم تحويله ضمنيّا في الحالة الأخيرة، إلى تنسيق التاريخ والوقت المناسب.

ولتجعل الأمر في غاية السهولة، استخدم الأعضاء الخاصة بمتغيرات التاريخ والوقت، والتي سنناقشها في [الفصل رقم 8](file:///C:\Users\Administrator\Desktop\02-%20Classes%20الخلايا\08-%20خلايا%20النصوص%20والتواريخ\2-%20التعامل%20مع%20التواريخ.htm) إن شاء الله.

كما أنّ بإمكانك استخدام الدوال الجاهزة التي يمنحها لك VB للتعامل مع التاريخ والوقت.. فمثلا: يمكنك استخدام الدالة "فرق التاريخ" DateDIff()، لحساب الفرق بين تاريخين مختلفين، وبالوحدة الزمنيّة التي تختارها (أيام، ساعات، ...).. إليك هذا المثال، لحساب عدد الأيّام التي عشناها في هذه الألفيّة:

**Dim Days As Long**

**Days = DateDIff(DateInterval.Day, #12/31/2000#, Now())**

**ملحوظة لمبرمجي VB6:**

لم يعد بالإمكان التعامل مع متغيرات التاريخ والوقت كأرقام، ولم يعد ممكنا أن تستخدمها في العمليات الحسابيّة كالطرح والجمع، كما كان في الماضي، ولكن بإمكانك استخدام [الدوال الزمنيّة](file:///C:\Users\Administrator\Desktop\02-%20Classes%20الخلايا\08-%20خلايا%20النصوص%20والتواريخ\2-%20التعامل%20مع%20التواريخ.htm) في المقابل.

**المتغيّرات الكائنات Object:**

الطريف في هذا النوع، أنّه يستطيع تخزين أيّ نوعٍ من البيانات، سواء من الأنواع السابقة، أو من أيّ كائن آخر تعرّفه اللغة، أو تعرّفه أنت.

ستتهلّل فرحا، وتهتف: رائع.. فلأعرّف كل المتغيّرات إذن من هذا النوع العامّ، حيثُ يمكنني أن أضع فيه أيّ نوع من البيانات دونما قلق!

للأسف: لا تبدو الأمور بهذه البساطة، ففعلُ ذلك سيكون له آثار جانبيّة سيّئة، فقبل أن يستخدم VB هذا المتغيّر الكائن، لا بد له أن يعرِف نوع البيانات التي به أولا، ليقوم بعمليات التحويل المناسبة، للتعامل مع هذا النوع من البيانات.. فمثلا، لو كان المتغيّر الكائن يحتوي على عدد صحيح، فعلى VB أن يحوّله إلى نصّ قبل لصقه بنصّ آخر.. مثل هذه التحويلات تمثّل عبئا على سرعة البرنامج.. لذا فعليك استخدام المتغيّر الكائن Object في حالات الضرورة فحسب.

**تنسيق الأرقام Formatting Numbers:**

الدالة "حوّل إلى نصّ" ToString هي إحدى وسائل كل أنواع البيانات، ما عدا البيانات النصّيّة String بالطبع،وهي تقوم بتحويل الأرقام إلى نصوص، بأيّ تنسيق تريده.

وكما رأيت سابقا، يمكنك استدعاء هذه الدالة بدون أي معاملات، لتحويل العدد إلى نص كما هو بدون تنسيق.

ولكنّ بإمكانك أيضا أن ترسل لهذه الدالة معاملا، تبيّن به الصيغة التي يجب أن يُنسّق العدد على شاكلتها.

وفي الجدول التالي، ستجد كل الرموز الجاهزة التي يمكن أن تستخدمها في التنسيق:

|  |  |  |  |
| --- | --- | --- | --- |
| الرمز | المعنى | مثال | الناتج |
| C أو c | عملة Currency | 12345.67.ToString("C") | سيتوقف الناتج على لغة نسخة الويندوز والمنطقة التي ضبطت عمله عليها، فلو كانت أمريكا، فسيكون الناتج $12,345.67، أمّا لو كانت مصر، فسيكون الناتج 12.345,67 ج.م. |
| E أو e | علمي Scientific | 12345.67.ToString("E") | 1.234567E+004 |
| F أو f | مقربة لأقرب 3 علامات عشريّة Fixed-point | 12345.6715.ToString("F") | 12345.672 |
| G أو g | عامّ General، وهي ترجع الرقم إمّا مقربا لأقرب ثلاث علامات عشرية، أو في تنسيق محدّد. |  |  |
| N أو n | تنسيق رقم Number | 12345.67.ToString("N") | 12,345.67 |
| X أو x | عدد سداسي عشر HexaDecimal | 250.ToString("X") | FA |

**ملاحظتان:**

1- يمكن أن يُتبع رمز التنسيق برقم، ليدل على عدد الخانات العشريّة في النصّ المنسّق، فالتعبير 5596.ToString("c") سوف ينتج عنه"$5,596.00"، أمّا التعبير5596.4499.ToString("c3") فسوف ينتج عنه "$5,596.450"، والتعبير (134.5).ToString("f3") سينتج عنه "134.500".

2- ليست كل التنسيقات قابلة للتطبيق على كل أنواع الأعداد، فمثلا، لا يمكن تنسيق عدد بتنسيق سداسي عشري إلا إذا كان عددا صحيحا.

ولكنّك قد لا تجد التنسيق الذي تريده في الرموز السابقة، فماذا تفعل؟

في هذه الحالة يمكنك تكوين صيغة عامة للتنسيق الذي تريده، باستخدام الرموز التالية:

|  |  |
| --- | --- |
| الرمز | معناه |
| 0 | خانة عشريّة في هذا الموضع، فإذا كانت غير موجودة، يتم وضع صفر مكانها، وأنت تعرف أنّ الصفر على أقصى يمين العلامة العشريّة لا يؤثّر في قيمة العدد، فالعدد 0.1 هو نفسه العدد 0.10. |
| # | رقم يؤثّر في قيمة العدد في هذا الموضع (مثل الجزء الصحيح من العدد). |
| . | علامة عشرية في هذا الموضع. |
| , | فاصلة في هذا الموضع. |
| % | علامة مئويّة في هذا الموضع. |
| E+0  E-0  e+0  e-0 | لتنسيق الرقم في صيغة أسّيّة.. ولقد ذكرنا سابقا أنّ العدد 233 × 10 12يمثّله الكمبيوتر كالتالي: 233E+12. |
| \n | سطر جديد، بمعني أنّ الجزء التالي من النص سيعرض في سطر جديد. |
| " " | لعرض نص بين علامتي تنصيص. |

**مثال:**

**MsgBox (123.4890.ToString("#.00"))**

**' ستعرض الرسالة الرقم مقربا إلى أقرب خانتين عشريتين: 123.49**

ما زالت لدينا نقطة هامّة، هي الرموز المستخدمة لتنسيق التواريخ، وسيوضّحها لك الأمثلة التالية، حيث وضعنا الناتج كتعليق بجوار كل جملة:

**Console.WriteLine(birthDate.ToString("d"))**

**'1/1/2000**

**Console.WriteLine(birthDate.ToString("D"))**

**'Saturday, January 01, 2000**

**Console.WriteLine(birthDate.ToString("f"))**

**'Saturday, January 01, 2000 12:00 AM**

**Console.WriteLine(birthDate.ToString("s"))**

**'2000-01-01T00:00:00**

**Console.WriteLine(birthDate.ToString("U"))**

**'Saturday, January 01, 2000 12:00:00 AM**

**ملحوظة1:**

لاستعادة الرقم الأصليّ من النصّ المنسّق، يمكنك استخدام الوسيلة Parse، التي تمتلكها كلّ أنواع البيانات الرقميّة.

**ملحوظة2:**

هناك مجموعة من دوال التنسيق الخاصّة بـ VB6 ما زال بإمكانك استخدامها حتّى الآن.. هذه الدوالّ هي:

تنسيق **Format** ـ تنسيق العملة **FormatCurrency** ـ تنسيق التاريخ والوقت **FormatDateTime** ـ تنسيق الأرقام **FormatNumber** ـ تنسيق مئويّ **FormatPercent**.. وسأترك لك مهمّة استكشاف هذه الدوالّ، وإن كنت لا أرى أنّك ستحتاج إليها أصلا!

**الاختصارات المحدّدة للأنواع Data Type Identifiers:**

كما ذكرنا من قبل، يمكنك ألا تذكر نوع المتغيّر، إذا لصقت الرمز الخاص به في نهاية اسمه، مثل:

**Dim MyText$**

فعلامة الدولار $ تدل على متغيّر نصّيّ String.. ويمكنك بعد ذلك أن تستخدم المتغيّر بعلامة الدولار MyText$ أو بدونها MyText، عند كتابة الكود.

والجدول التالي يوضّح الاختصارات المستخدمة مع كل نوع من المتغيرات:

|  |  |
| --- | --- |
| نص String | $ |
| عدد صحيح Integer (Int32) | % |
| عدد طويل Long (Int64) | & |
| عدد مفرد Single | ! |
| عدد مزدوج Double | # |
| عدد عشري Decimal | @ |

ولا أرى أنّ هناك دافعا حقيقيا يدفعك لاستخدام مثل هذه الاختصارات، إلا إذا كنت من هواة إرباك نفسك!!

**الخيار "التعريف الصريح"Option Explicit :**

يمكنك أن تكتب الجملة التالية في بداية أي ملف، قبل تعريف النموذج أو الخليّة أو قالب الكود Module:

**Option Explicit Off**

وذلك لإخبار مترجم الكود أنّك ستستخدم المتغيّرات في هذا الملفّ، بدون أن تحتاج لتعريفها أولا.. في هذه الحالة سيفترض المترجم أنّ هذه المتغيّرات يمكنها أن تستوعب أي نوع من أنواع البيانات، لهذا سيعتبرها من النوع "كائن" Object، حيث سيقوم بتعديل نوع كل متغيّر فيما بعد، تبعا لنوع البيانات التي تخزّنها فيه.

**ملحوظة لمبرمجي VB6:**

لم يعد بالإمكان استخدام المتغيرات من النوع "متنوّع" Variant، وفي المقابل يمكنك استخدام المتغير من النوع "كائن" Object لتخزين أي نوعٍ من أنواع البيانات والكائنات.

**الخيار "التحويل الدقيق" Option Strict:**

هذا الخيار مغلق في الوضع التلقائيّ، ليسمح للغة بنقل البيانات بين أنواع مختلفة دون استخدام دوال التحويل، ففي هذه الحالة، تقوم اللغة بإجراء التحويلات المناسبة تلقائيّا.

انظر هذا المثال:

**Dim A As String = "25000"**

**' انظر كيف سنستخدم المتغير النصّي في عمليّة القسمة**

**Console.WriteLine(A / 2)**

وانظر أيضا هذا المثال، لترى كيف نجمع نصا على رقم:

**Dim A As Double = 31.03**

**A = A + "1"**

أمّا هذا المثال فسيوضّح لك الأمر تماما:

**Dim I As Integer, S As String**

**I = 10**

**S = "11"**

**Console.WriteLine(I + S) ' ستظهر القيمة 21 في شاشة المخرجات (جمع حسابي)**

**Console.WriteLine(I & S) ' ستظهر القيمة 1011 في شاشة المخرجات (جمع نصّي)**

ويمكنك أن توقف هذه الخاصّيّة، بكتابة الجملة التالية في بداية الملفّ:

**Option Strict On**

وبهذا لن يمكنك تنفيذ المثالين السابقين، حيث ستجد أنّ خطوطا متعرجة قد ظهرت ، لتدل على وجود خطإ كالتالي: "خيار التحويل الدقيق يمنع التحويل التلقائي من نص إلى مزدوج" Option strict disallows implicit conversions from String To Double.

ولكن هذا لا يعني أنّ كل أنواع التحويلات الضمنيّة ستصير مستحيلة، فما زال التحويل بين أنواع المتغيرات الرقميّة يتم آليّا، ولكن في اتجاه واحد فقط: حينما تضع المتغير الأصغر في المتغيّر الأكبر، كأن تضع "عددا صحيحا"Integer في "عدد مزدوج" Double.. ولكن العكس ليس مسموحا به، كأن تحاول وضع "عدد مزدوج" Doubleفي "عدد صحيح" Integer، لأنّ هناك احتمالا كبيرا ألا يستوعب العدد الصحيح قيمة العدد المزدوج.

وهناك شيء آخر سيمنعه خيار التحويل الدقيق، ذلك هو "الربط المتأخّر" Late-bind.. أتذكر عندما كنا نبرمج أزرار الأرقام في تطبيق الآلة الحاسبة في الفصل السابق؟.. لقد كان تعريف الإجراء المستجيب لضغط الحدث كالتالي:

**Private Sub Digit\_Click(ByVal sender As System.Object, \_**

**ByVal e As System.EventArgs) HAndles Bt1.Click, Bt2.Click, \_**

**Bt3.Click, Bt4.Click, Bt5.Click, Bt6.Click, \_**

**Bt7.Click, Bt8.Click, Bt9.Click, Bt0.Click**

**End Sub**

وقلنا إنّ المتغيّر sender هو متغير عام من النوع "كائن" Object، يمكن أن يستقبل أي نوع من أنواع البيانات أو الكائنات.. ونظرا لأنّنا نعرف أنّه سيستقبل الزر المضغوط، فيمكننا كتابة جملة من نوع:

**sender.Text = "1"**

هذا هو "الربط المتأخّر"، فبينما تعامل أنت المتغيّر sender على أنّه زر، وله الخاصيّة Text، لا يستطيع مترجم الكود أن يستوعب هذه المعلومة إلا عند محاولة تنفيذ هذه الجملة عند تشغيل البرنامج، لأنّ المترجم لا يقرأ الغيب، ولا يعرف ما نوع الكائن أو المتغيّر الذي سيستقبله المتغير العام sender قبل أن يستقبله بالفعل!

لهذا عندما تكتب sender.، وتظهر لك قائمة الأعضاء، لن تجد فيها خاصية Text.. بل لن تجد فيها إلا دالة واحدة هي "اقرأ نوع الكائن" GetType.. لهذا فإن استخدام التعبير sender.Text يعتبر استخدامًا خطيرًا، لأنّ المترجم لن يستطيع أن يتحقّق من قدرة الكائن على تنفيذ الجملة التي كتبتها، إلا أثناء تنفيذها، ممّا يهدّد بحدوث أخطاء أثناء تشغيل البرنامج وإنهاء تشغيله.

لهذا فإن استخدام خيار "التحويل الدقيق" لن يسمح بمثل هذا الاستخدام، ويجب عليك ساعتها أن تستخدم "الربط المبكّر" Early-bind، بتحويل نوع الكائن sender إلى زر، كالتالي:

**CType(sender, Button).Text = "1"**

ففي هذه الحالة يعرف المترجم مع أيّ شيء يتعامل، ويمكنه أن يتحقّق من صحّة الكود قبل تشغيل البرنامج.

**ملحوظة:**

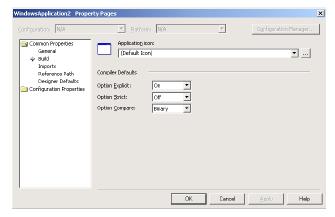
قلنا إن كتابة الخيارين:

**Option Explicit Off**

**Option Strict On**

في بداية أيّ ملف من ملفات المشروع، يؤثّر فقط على هذا الملفّ.. فماذا لو أردت أن تؤّثّر على كل ملفات المشروع؟

في هذه الحالة، يمكنك أن تختار هذين الخيارين من صفحات الخواص للمشروع الحالي، وذلك بضغط زر الفأرة الأيمن على اسم المشروع في متصفّح المشاريع Solution Explorer، وضغط "خصائص" Properties من القائمة الموضعيّة.. وفي هذه الحالة، ستكون الإعدادات التي تختارها لهما عامّة لكل المشروع.



**التحويل بين أنواع المتغيّرات Converting Variable Types:**

في الجدول التالي، ستجد الدوال التي تسمح لك بالتحويل من نوع متغيّر لآخر:

|  |  |
| --- | --- |
| الدالة | تحوّل إلى |
| CBool | منطقي Boolean |
| CByte | وحدة ذاكرة Byte |
| CChar | حرف Unicode character |
| CDate | تاريخ Date |
| CDbl | عدد مزدوج Double |
| CDec | عدد عشري Decimal |
| CInt | عدد صحيح Integer (Int32) |
| CLng | عدد طويل Long (Int64) |
| CObj | كائن Object |
| CShort | عدد قصير Short (Int16) |
| CSng | عدد مفرد Single |
| CStr | نصّ String |

مثال:

لتحويل عدد صحيح إلى عدد مزدوج، جرّب ما يلي:

**Dim A As Integer**

**Dim B As Double**

**B = CDbl(A)**

ولكن فيمَ نحتاج ذلك؟

افترض أنّك عرّفت عددين صحيحين كالتالي:

**Dim A As Integer, B As Integer**

**A = 23**

**B = 7**

لو حاولت قسمة العددين 23/7، فسيكون الناتج عددا مزدوجا.. هنا سينتج خطأ عن جملةٍ كالتالية:

**A = A / B**

ولكن استخدام دالة التحويل، يمكن أن يُخلصك من هذا الخطإ:

**A = CInt(A / B)**

وإن كان الثمن هو أنّك ستفقد الدّقّة العشرية للناتج، ولو لم تكن تريد ذلك، فعليك أن تعرف متغيرا مزدوجا تستقبل فيه الناتج:

**Dim C As Double**

**C = A / B**

كما أن بإمكانك استخدام الدالة العامّة "تحويل النوع" CType، للتحويل إلى أي نوع تحدّده لها.. إليك هذا المثال:

**Dim A As String = "34.56"**

**Dim B As Double**

**B = CType(A, Double) / 1.14**

وكما أوضحنا من قبل، فإنّ استخدام التحويل الصريح إجباريّ فقط في حالة عمل الخيار "التحويل الدقيق" Option Strict On ، ولكنّ الأفضل دائما أن تراعي دقّة التحويل أثناء كتابتك للكود، وذلك لأنّ التحويل الضمنيّ قد يُفسد بعض البيانات، مثل ما يحدث عندما تحاول وضع عدد مزدوج في عدد صحيح، ففي هذه الحالة ستخسر الجزء العشريّ من الرقم، وهو ما قد لا تريده.

**القيمة "لا شيء" The Nothing Value:**

تستخدم هذه القيمة مع المتغيرات من النوع "كائن" Object، لتوضّح أنّ المتغيّر ما زال فارغا لم يرتبط بكائن ما بالفعل.. فإذا أردت أن تدمّر ارتباط متغيّر بكائن، فضع فيه هذه القيمة:

**Dim B As New Button**

**{بعض الكود الذي يستخدم الزر}**

**B = Nothing**

**ملحوظة لمبرمجي VB6:**

لم يعد مطلوبا استخدام كلمة Set لوضع القيم في الكائنات، ففي الماضي كان حتميا أن تكتب:

**Set B = Nothing**

ولكن الآن يتم معاملة الكائنات كالمتغيرات العاديّة، ويمكن وضع القيم بها باستخدام عملية التساوي العاديّة:

**B = Nothing**

ولو أردت أن تعرف إذا ما كان المتغير مرتبطا بكائن أم لا، استخدم جملةً كالتالية:

**If B Is Nothing Then Exit Sub**

ولو أردت إعادة ربط هذا المتغيّر بنسخة جديدة من الكائن مرّة أخرى، فاستخدم جملة كالتالية:

**B = New Button**

**ملحوظة:**

عند تعريف متغير من أيّ نوع من أنواع البيانات الأساسيّة ذات القيمة الرقميّة (العدد الصحيح Integer، والمفرد Single، والمزدوج Double، والقيم المنطقية Boolean.. إلخ) فإنه يأخذ القيمة الافتراضيّة (صفر للأرقام و False للقيمة المنطقيّة).. ولكنّ هذا لا ينطبق على المتغيّرات النصّيّة String، والمتغيّرات الحرفيّة Char، فهي ليس أنواعا رقميّة، ولكنّها أنواع مرجعيّة Reference Type، لهذا فإنّها تأخذ القيمة الابتدائيّة Nothing، ممّا يهدّد بحدوث خطإ عند محاولة استخدام أيّ وسيلة من وسائلها أو عند محاولة إرسالها كمعامل لأيّ دالة.. ولتنّجب أيّ أخطاء محتملة استخدم واحدا من الحلول التالية:

- بعد تعريف المتغيّر النصّي أو الحرفي ضع فيه أيّ قيمة، حتّى لو كانت نصّا فارغا "":

**Dim X As String**

**X = ""**

- أو ضع للمتغيّر قيمة ابتدائيّة في سطر تعريفه:

**Dim X As String = ""**

- أو اربط المتغيّر بنسخة جديدة بعد تعريفه:

**Dim X As Char**

**X = New Char**

- أو استخدم كلمة New في تعريف المتغيّر منذ البداية، حتّى يرتبط بنسخة جديدة من المتغيّر فارغة القيمة "":

**Dim X As New Char**

**اختبار أنواع المتغيّرات Examining Variable Types:**

يمتلك كلّ متغيّر وسيلتين تخبرانك بنوعه:

- "اقرأ النوع" GetType():

وهي تُرجع نوعا Type عبارة عن كائن يمثّل نوع المتغيّر.

- "اقرأ كود النوع" GetTypeCode():

وهي تُرجع قيمةً ترمز للنوع.. فالعدد المزدوج مثلا رقمه 14.

انظر للمثال التالي:

**Dim Var As Double**

**Console.WriteLine("نوع المتغيّر هو " & Var.GetType.ToString)**

**If Var.GetType() Is GetType(Double) Then Exit Sub**

ويمكنك كذلك استخدام التعبير TypeOf ….. IS، لمعرفة نوع المتغيّر، كما في المثال التالي:

**Dim Var As String = ""**

**If TypeOf Var Is String Then MsgBox("نوع المتغير نصّ")**

**ملحوظة:**

لا يمكن استخدام التعبير TypeOf ….. IS مع أنواع المتغيّرات الرقميّة، مثل الأعداد الصحيحة والمفردة والمزدوجة والقيم المنطقيّة.. يمكن استخدامها فقط مع الأنواع المرجعيّة Reference Type مثل النصوص والحروف والكائنات Objects والأنواع المشتقّة من الخلايا Classes.. وقبلَ أن تتعجّبَ من هذا الاختلاف، يجبُ أن أخبرَك أنّ المتغيّرات الرقميّة ـ على خلاف الحروف والنصوص ـ ليست في الواقع خلايا، بل هي عبارة عن سجلات Structures.. ماذا؟.. لا تعرفُ ما هي السجلات؟.. لا يهمّنّك.. سنتعرّف عليها في نهاية هذا الفصل.

ولو أردت أن تعرف اسم نوع المتغيّر، فيمكنك أن تستخدم الدالة TypeName، التي تُعيد نصّا يمثّل اسم النوع الذي ينتمي إليه المتغيّر.. انظر للجمل التالية:

**Dim name As Integer**

**Dim a As Object**

**Console.WriteLine(TypeName(name)) ' Integer**

**Console.WriteLine(TypeName(a)) ' Nothing**

**a = "نصّ ما"**

**Console.WriteLine(TypeName(a)) ' String**

وإذا كان التعامل مع اسم النوع لا يعجبك، فيمكنك التعامل معك من خلال المرقّم VariantType، وذلك عن طريق استخدام الدالّة VarType.. انظر للمثال التالي:

**Dim X As Object**

**If VarType(X) = VariantType.Empty Then** **X = "111"**

كما أنّ لديك نوعيّة أخرى من الدوال، لمعرفة نوعيّة البيانات: هل هي رقميّة أم نصّيّة ... إلخ.. انظر للجدول التالي:

|  |  |
| --- | --- |
| "إنّه رقميّ" IsNumeric() | تُرجع "صواب" True، إذا كان معاملها رقما من أيّ نوع، حتّى لو كان نصّا يحتوي على رقم. |
| إنّه تاريخ" IsDate() | تُرجع "صواب" True، إذا كان معاملها تاريخا أو وقتا. |
| إنّه لا شيء" IsNothing | تُرجع "صواب" True، إذا كان معاملها غير مرتبط بأيّ كائن. |
| "إنّه فارغ" IsDBNull() | تُرجع "صواب" True، إذا كان معاملها غير مرتبط بأيّ كائن.. وهذه الدالة مماثلة لدالة VB6 "إنّه فارغ" IsNull(). |
| "إنّه مرجع" IsReference() | تُرجع "صواب" True، إذا كان معاملها كائنا، وهي مماثلة لدالة VB6 "إنّه كائن" IsObject(). |
| IsArray | تُرجع "صواب" True، إذا كان معاملها مصفوفة.. وسنتعرّف على المصفوفات لاحقا في هذا الفصل. |

**مجال المتغيّر Variable Scope:**

مجال المتغيّر هو مقطع الكود الذي يمكن خلالَه رؤيةُ المتغيّر واستخدامه.

وهذا المجال ببساطة، هو أصغر مقطع يوجد فيه المتغيّر، من المقاطع التالية:

- Sub.. End Sub.

- Function.. End Function.

- If.. End If.

- For.. Next.

- Do.. Loop.

- While.. End While.

وهذا يعني أنّ أيّ متغيّر يتم تعريفه في أحد هذه المقاطع، لا يمكن التعامل معه إلا داخل هذا المقطع، هو وما ينتمي إليه من مقاطع فرعيّة.. وبهذا يمكن أن يحتوى البرنامج على عشرات المتغيرات بنفس الاسم (مثل المتغير sender الذي تجده في معامل كل الإجراءات المستجيبة للأحداث)، ولكنّها معزولة عن بعضها، كل في نطاقه ومجاله، بحيث تكون لكلّ منهما قيمته المستقلّة.

هذا، ويُسمّى المتغيّر المعرّف على مستوى الإجراء أو الدالة متغيّرا موضعيّا Local Variable، بينما يسمّى المتغيّر الذي نعرّفه داخل أحد المقاطع متغيرا مقطعيا Block Variable.. أمّا المتغيّرات المعرّفة على مستوى الخليّة أو القالب، فتسمّى متغيرات مستوى القالب Module-level Variables.

انظر لهذا المثال، ولاحظ كم لدينا من متغيّر يحمل الاسم X:

**Sub VarDomain()**

**Dim I As Integer 'متغير خاص بهذا الإجراء فقط**

**If I = 3 Then**

**Dim X As Integer = 3 ' متغير خاص بجملة الشرط فقط**

**End If**

**For I = 1 To 3**

**Dim X As String ' متغير خاص بجملة التكرار "من إلى" فقط**

**Next I**

**Do While I < 5**

**Dim X As Long ' متغير خاص بجملة التكرار :نفذ بينما" فقط**

**Loop**

**End Sub**

ورغم هذه المرونة، إلا أنّ VB لن يسمح لك بتعريف متغيّر اسمه I داخل أي مقطع من مقاطع الإجراء السابق، لأنّ I هو متغيّر موضعيّ سبق تعريفه على مستوى الإجراء كلّه، مما سيمنعك من استخدامه مع المتغيّر الذي يحمل نفس اسمه داخل نفس المقطع.. تسمّى هذا الأمر "التظليل" Shadowing، بمعنى أنّ المتغيّر الأصغر يحجب المتغيّر الأكبر، لأنّهما يحملان نفس الاسم، فلا يمكن استخدام أكثر من واحد منهما في نفس المقطع.

ولكنّ حدوث هذا مقبول عندما يكون أحد المتغيّرين الحاملين لنفس الاسم قد تم تعريفه على مستوى النموذج، والآخر على مستوى الإجراء، ففي هذه الحالة يمكن التفريق بينهما كما في المثال التالي:

**Public Class Form1**

**Inherits System.Windows.Forms.Form**

**Dim I As Int16 = 3**

**Private Sub Button1\_Click(ByVal sender As System.Object, ByVal e \_**

**As System.EventArgs) Handles Button1.Click**

**Dim I As Int16 = 5**

**'هذه هي طريقة استخدام متغير مستوى النموذج**

**MsgBox(Me.I) ' الناتج 3**

**'هذه هي طريقة استخدام متغير مستوى الإجراء**

**MsgBox(I) ' الناتج 5**

**End Sub**

**End Class**

وبرغم أنّ المتغيّر المقطعيّ لا يُرى إلا في مقطعه، إلا إنّه لا يدمّر فور مغادرة المقطع، ولكن يدمّر وتزول قيمته عند مغادرة الإجراء الذي يحتويه، لهذا عند دخول المقطع مرة أخرى (قبل مغادرة الإجراء)، ستكون للمتغيّر المقطعيّ نفس قيمته التي يحتفظ بها منذ مغادرة المقطع آخر مرّة.. لهذا يجب مراعاة ذلك، حتّى لا تنتج نتائج غير متوقّعة.

والمتغيرات المعرّفة على مستوى القالب، يمكن التعامل معها من جميع الإجراءات والدوال المنتمية لهذا القالب، وهو ما يوجب عليك أن تقلّل من تعريف مثل هذه المتغيّرات بقدر الإمكان، لأنّها ترفع قابلية الخطإ في البرنامج، نظرا لتعدد الأماكن التي يتم تغييرها فيها.

ما زال هناك نوع من مجال المتغيّرات، ذلك هو مجال المشروع ككل، حيث يمكن تعريف بعض المتغيرات على مستوى القالب أو النموذج على أنّها عامّة Public لكل المشروع:

**Public GlobalVar As Integer**

**ملاحظة:**

لا يمكن تعريف مثل هذا النوع من المتغيرات على مستوى المقطع ولا على مستوى الإجراء أو الدالة.

ولا تقتصر كلمة Public على المتغيرات، بل تمتدّ كذلك للأدوات، فلو لديك أداة على نموذج، وتريد التعامل معها من نموذج آخر، فيمكنك تعديل خاصّيّة "المجال" Modifier الخاصّة بهذه الأداة إلى "عام" Public.

ولكنّ هناك مشكلة: استخدام المجال العام للمتغيرات، يجعلها قابلة للاستخدام من خارج المشروع نفسه، من أي مشروع آخر يُضيف مرجعا لمشروعك.. فماذا لو لم تكن تريد حدوث ذلك؟

في هذه الحالة استخدم كلمة "صديق" Friend بدلا من Public، فهي تجعل المتغيّر مرئيّا في كل أجزاء المشروع، ولكن لا يمكن استخدامه من خارجه من أي مشروع آخر:

**Friend Var As Integer**

**عمر المتغيّر The Lifetime of a Variable:**

عمر المتغيّر هو الفترة التي يحتفظ خلالها بقيمته.

- المتغيرات المعرّفة على مستوى المشروع تحتفظ بقيمتها طوال تشغيل البرنامج.

- والمتغيرات المعرّفة على مستوى النموذج، توجد عند تحميل النموذج Load، ولو حاولت استخدامها قبل تحميل النموذج، فسيتم تحميل النموذج أولا.. وعند إنهاء تحميل النموذج UnLoad، تنتهي هذه المتغيرات، فلو أعيد تحميل النموذج، توضع بها قيمها الابتدائيّّة كأنّما أعيد حجزها في الذاكرة من جديد.

- والمتغيّرات المعرّفة داخل الإجراءات، تحتفظ بقيمتها إلى أن ينتهي الإجراء، حيث يتم تحرير الذاكرة منها، فإذا عاد البرنامج إلى الإجراء من جديد، يتم حجزها من جديد، ومنحها قيمها الابتدائية.

**ملاحظة:**

ينتهي الإجراء بإحدى ثلاث جمل:

1- End Sub: وهي النهاية الطبيعية للإجراء.

2- Exit Sub أو Return: وهي نهاية طارئة، تنهي الإجراء عند حدوث سبب من الأسباب.

3- End: وهي تنهي البرنامج كلّه.

ولكن من الممكن أن يغادر VB الإجراء مؤقّتا، نتيجة استدعاء إجراء آخر من داخله.. في هذه الحالة يتم الاحتفاظ بقيم جميع المتغيرات كما هي، حتّى ينتهي الإجراء الفرعيّ، ويعود VB لتنفيذ الإجراء الأوّل بطريقة طبيعيّة.

سيدور بيننا هذا الحوار:

- ولكن ماذا لو أردت أن أحتفظ بقيمة المتغيّر بعد انتهاء الإجراء؟

- يمكنك أن تعرّفه خارج الإجراء.

- ولكنّني لن أستخدم المتغيّر إلا في هذا الإجراء، وأنت حذّرتني من تعريف المتغيّر بمجال أكبر من المطلوب، حتّى لا يتسبّب في أخطاء غير متوقّعة.

- آه: عندي لك حلّ مناسب.

- أسعفني به.

- إنّ بإمكانك أن تعرّف المتغيّر على أنّه "ثابت القيمة" Static، وبهذا سيحتفظ بقيمته حتّى بعد انتهاء الإجراء، رغم أنّه ما زال موضعيّا، لا يمكن التعامل معه من خارج الإجراء.

- يا بن الإيه!

- هذه هي صيغة التعريف:

**Static X As Integer = 3**

- ولكن: ألن توضع به القيمة الابتدائيّة 3، في كلّ مرّة يتمّ فيها استدعاء الإجراء، ممّا يدمّر قيمته القديمة؟

- لا، فهذه القيمة الابتدائيّة لا توضع في المتغيّر، إلا في أوّل مرّة يتم تعريفه فيها فقط، بعد ذلك سيحتفظ بقيمته التي كانت به قبل مغادرة الإجراء، ليمكنك استخدامها عندما تعود للإجراء مرّة أخرى.

- جميل.. ولكن ماذا لو أردت أن أجعله يعود لقيمته الابتدائيّة من جديد، لسبب أو آخر؟

- سؤالٌ وجيه.. للأسف.. لا توجد وسيلة مباشرةٌ لفعل هذا من خارج الإجراء.. ولكن يمكن التحايل على ذلك، بتعريف معامل للإجراء، مهمّته إخبار الإجراء بإعادة المتغيّرات ثابتة القيمة Static إلى قيمها الأصليّة.. انظر لهذا المثال.. أضف الإجراء التالي لخليّة النموذج، قبل جملة End Class مباشرةً:

**Sub ShowMessage(InitializeStatics As Boolean)**

**Static X As Integer = 3, Y As String = "رقم "**

**If InitializeStatics = True Then**

**X = 3**

**Y = "علي"**

**End If**

**X + = 1**

**Y += Str(I)**

**MsgBox(Y)**

**End Sub**

والآن ضع زرّا على النموذج واكتب فيه ما يلي:

**ShowMessage (False) ' ستظهر الرسالة "رقم 1"**

**ShowMessage (False) ' ستظهر الرسالة "رقم 12"**

**ShowMessage (False) ' ستظهر الرسالة "رقم 123"**

**ShowMessage (False) ' ستظهر الرسالة "رقم 1234"**

تلاحظ تصاعد رقم الرسالة، وذلك لأنّ كلا من المتغيّرين يحتفظ بقيمته، التي يتمّ زيادتها في كلّ استدعاء.. ولكن لو جرّبت الاستدعاءات التالية بعد الاستدعاءات السابقة:

**ShowMessage (True) ' ستظهر الرسالة "رقم 1"**

**ShowMessage (True) ' ستظهر الرسالة "رقم 1" أيضا**

**ShowMessage (True) ' ستظهر الرسالة "رقم 1" أيضا**

فستلاحظ ثبات نتيجة الرسالة، وذلك لأنّنا في كلّ مرّة نطلب من الإجراء إعادة المتغيّرين ثابتي القيمة إلى قيمتيهما الابتدائيّة.

وستجد المزيد التفاصيل حول تعريف الإجراءات ومعاملاتها، في للفصل التالي.

**الثوابت Constants:**

عرَفنا وعرّفنا المتغيّرات، فلماذا لا نعرِفُ ونُعرّف الثوابت؟

يبدو أنّك قد فهمت ما سيُقال هنا: هذا نوع من مخازن الذاكرة، توضع فيه قيمة، تماما كالمتغيّرات، ولكنّ هذه القيمة لا يمكن تغييرها بعد ذلك، وستظلّ ثابتة طوال البرنامج.. مرحى: لقد فهمتَ بالضبط ما هي الثوابت!

وهذه طريق تعريفها:

**Const MyValue = 10**

ولكن لماذا نعرّف مثل هذه الثوابت؟

إنّ لذلك أهمّيّة كبيرة.. فبخلاف أنّها تُنفّذ أسرع من المتغيّرات، فإنّها تمكّنك من تغيير الكثير من الحسابات بمجرّد تغيير جملة واحدة.. افترض أنّك تستخدم المتغيّر MyValue في ثلاثين عمليّة حسابيّة في مواضع متفرّقة من التطبيق.. الآن لو اكتشفت أنّ القيمة 10 غير مناسبة، فكل ما عليك هو تغييرها في جملة التعريف، ليتم تعديل كل العمليات الحسابيّة الثلاثين دفعة واحدة:

**Const MyValue = 15**

ولكن حاول أن تتخيّل حجم المأساة، لو كنت تستخدم العدد 10 مباشرةً في حسابات العمليات الثلاثين!!.. في هذه الحالة كلّما اكتشفت حاجتك لتغيير هذه القيمة، فعليك البحث عن هذه الجمل واستبدالها.. يا لها من عمليّة شاقّة!

ويمكن أن تحدّد نوع الثابت لمترجم الكود، كما يلي:

**Const MyValue As String = "اسمي هو محمد"**

كما يمكن أن تحدّد مجال الثابت، بكلمة من الكلمات: Public أو Private أو Friend:

**Public Const MyValue As Integer = 15**

ومسموح لك باستخدام الثوابت في تعريف ثوابت أخرى، فمثلا: يمكن استخدام التعبير التالي لتعريف ثابت يحتوي على النسبة التقريبيّة "ط":

**Const Pi = Math.Pi**

**Const Pi2 = 2 \* Pi**

ولكن ليس مسموحا لك باستخدام الدوال في تعريف الثوابت، فمثلا: التعريف التالي غير مسموح به:

**Const Pi = Math.Log(Math.Pi)**

**ملحوظة:**

حاول ألا تقع في التعريفات الدائريّةCircular Declaration عند تعريف ثوابت بثوابت أخرى.. فمثلا عند تعريف:

**Const Pi = Pi2 / 2**

**Const Pi2 = 2 \* Pi**

ستجد أن كلا من التعريفين معتمد على الآخر، بطريقة ليس لها معنى، فأيّ منهما لن تكون له قيمة بدون الآخر، ممّا سيجعل المترجم يعرض رسالة خطإ.

**المصفوفات Arrays**

* + [**ما هي المصفوفة:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\03-%20المتغيرات\14-%20المصفوفات\1-%20ما%20هي%20المصفوفة.htm)
  + [**وضع القيم الابتدائيّة للمصفوفات Initializing Arrays:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\03-%20المتغيرات\14-%20المصفوفات\2-%20وضع%20القيم%20الابتدائيّة%20للمصفوفات.htm)
  + [**حدود المصفوفة Array Limits:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\03-%20المتغيرات\14-%20المصفوفات\3-%20حدود%20المصفوفة.htm)
  + [**المصفوفة متعدّدة الأبعاد Multidimensional Arrays:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\03-%20المتغيرات\14-%20المصفوفات\4-%20المصفوفة%20متعدّدة%20الأبعاد.htm)
  + [**المصفوفات المرنة Dynamic Arrays:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\03-%20المتغيرات\14-%20المصفوفات\5-%20المصفوفات%20المرنة.htm)
  + [**مصفوفة المصفوفات Arrays of Arrays:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\03-%20المتغيرات\14-%20المصفوفات\6-%20مصفوفة%20المصفوفات.htm)
  + [**مشروع المصفوفات:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\03-%20المتغيرات\14-%20المصفوفات\7-%20مشروع%20المصفوفات.htm)

**ما هي المصفوفة:**

البرمجة الحقيقيّة تبدأ من هذه النقطة، فلقد صُنع الكمبيوتر أساسا، ليقوم بالعمليات الرتيبة المتكرّرة لآلاف أو ملايين المرّات، بسرعةٍ وبدون ملل.

افترض مثلا أنّك تريد حساب متوسط العمر لخمسين طالبا.. أوّل ما ستفكّر فيه، هو أن تعرّف خمسين متغيّرا وتجمعها معا وتقسم الناتج على 50.. إنّ مثل هذه الطريقة ستستهلك منك شهرا على الأقلّ لكتابتها، وهي كفيلة بجعلك تكره البرمجة على بكرة أبيها!

مع أنّك تستطيع أن تكتب هذا البرنامج في خمس سطور لا غير.. تخيّل!

والفكرة كلّها تعتمد على تخزين أعمار الطلبة في "تركيب ما"، يمكن للكمبيوتر أن يتعامل معه بطريقة آليّة رتيبة متكرّرة، لينفّذ عليه العمليّات التي نريدها.

هذا التركيب هو المصفوفة Array، التي تتكوّن من مجموعة من الخانات، كل خانة منها تحتفظ بقيمة معيّنة، بحيث يمكن الوصول لهذه القيمة عن طريق رقم خانتها Index.

ألا يبدو لك هذا الكلام مألوفا؟

إنّ النصّ هو عبارة عن مصفوفة من الحروف، ولقد استخدمنا بالفعل الدالة "حروف" Chars للحصول على أيّ حرف من أيّ موضع في مصفوفة النصّ:

**Dim X As String = "أنا مصفوفة حروف"**

**MsgBox ( X.Chars(2)) ' ستعرض الرسالة الحرف ا**

ولكن كيف نعرّف المصفوفة؟

تماما كما تعرّف المتغيّر العاديّ، ولكنّنا سنُتبع اسمه بقوسين يحتويان على عدد خانات المصفوفة:

**Dim StudentAge(49) As Integer**

بهذا صار لدينا مصفوفة تحتوي على 50 خانة (من 0 إلى 49)، كل خانة منها هي متغيّر من النوع "عدد صحيح".

**ملحوظة لمبرمجي VB6:**

لم يعد بإمكانك استخدام جملة Option Base 1 لجعل ترقيم خانات المصفوفة يبدأ من 1 بدلا من 0.. أنت مجبر على أن تتعامل مع مصفوفات تبدأ بالخانة رقم صفر، سواء أعجبك هذا أم لا!

ليس هذا فحسب، بل لم يعد بإمكانك تحديد بداية ونهاية الترقيم بجملة كالتالية:

**Dim StudentAge(20 To 70) As Integer**

فمثل هذه الجملة مرفوضةٌ الآن تماما.

ولكن كيف نتعامل مع المصفوفة؟

الأمر أبسيط ممّا تتخيّل، فبمجرد كتابة رقم الخانة التي تريدها، يبدو الأمر كأنّك تتعامل مع متغيّر عاديّ تماما:

**StudentAge(0) = 10**

**MsgBox (StudentAge(0).ToString)**

فلو اعتبرت أنّ الجزء (0) غير موجود في الجملتين السابقتين، فستجد أنّك تتعامل مع متغير تقليديّ من النوع عدد صحيح، حتّى إنّك تستخدم الدالة ToString الخاصّة به.

سيدور بيننا هذا الحوار.. ستبادرني قائلا:

- جميل.. هذه هي المصفوفة، ولكنني ما زلت لا أفهم كيف ستساعدنا في تبسيط الحسابات.

- فكّر قليلا في فائدة وجود رقم الخانة Index.

- لا أرى فيه ما يلفت النظر.. ما زالت العمليّة تشبه طريقة التعامل مع خمسين متغيّرا، اللهم إلا إنّنا اختصرنا جمل تعريفهم في جملة واحدة واسم واحد!

- حتّى لو قلت لك إنّك تستطيع كتابة رقم الخانة بطريقة غير مباشرة، بوضعه في متغيّر؟

- كيف هذا؟

- كالتالي:

**Dim StudentAge(49) As Integer**

**Dim I As Integer = 3**

**StudentAge (I) = 10**

فأنت في الجملة الثالثة تضع القيمة 10 في الخانة الرابعة (الخانة رقم 3) من المصفوفة، نظرا لأنّ المتغيّر I = 3.

- يبدو هذا مثيرا.. ولكنّي ما زلت لا أفهم كيف سنستفيد منه!

- فكّر كالتالي: هل هناك طريقة أخبر بها اللغة، أنّني أريد أن أنفّذ عمليّة ما على المصفوفة، على الخانة رقم I، حيث I تبدأ من 0، ثم تزيد في كل مرّة، لتصبح 1، ثم 2، ثم 3، ...، حتّى تصل إلى 49، حيث تتوقّف العمليّة؟

- إم.. تبدو فكرةً جذّابة.. ولكن هل هناك حقًّا ما يفعل مثل هذه الألاعيب الجهنّميّة؟

- امتلئ بهجة، فلدينا هذه الطريقة بالفعل!

- أسعفني بها أرجوك.

كل ما نحتاجه، هو مقطع التكرار "من إلى" For Loop، كالتالي:

**For I = 0 To 49 Step 1**

**' { StudentAge (I) أي عمليات على الخانة}**

**Next**

وأعتقد أنّ هذا المقطع واضح، فهو يقول ببساطة: من (I) تساوي 0 إلى أن تصبح (I) تساوي 49، بزيادة 1 في كل دورة، نفّذ المقطع.

معنى هذا أنّ I ستساوي صفرا في بداية الجملة التكراريّة، وكلّما وصل التنفيذ إلى كلمة "التالي" Next، يتمّ زيادة 1 على قيمة المتغيّر (I)، حيث يتمّ التأكّد هل ما زال لم يتجاوز الرقم 49:

- فإن كان كذلك، يتمّ تنفيذ المقطع مرّة أأخرى، لكن بالقيمة الجديدة للمتغيّر I.

- وإن كان I أكبر من 49، ينتهي التكرار، وينتقل التنفيذ إلى السطر التالي لكلمة Next.. معنى هذا أن قيمة (I) ستكون 50 بعد انتهاء تنفيذ التكرار.

الآن، كلّ ما نريد فعله لحساب متوسّط أعمار الطلبة، هو جمع كل أعمار الطلاب (كل خانات المصفوفة) داخل مقطع التكرار، وبعد انتهائنا من ذلك، نقسم الناتج على 50.. هذا هو البرنامج:

**Dim StudentAge(49) As Integer, I As Integer, Sum As Integer**

**'{بعض الكود لكتابة أعمار الطلاب في المصفوفة}**

**For I = 0 To 49 Step 1**

**Sum += AstudentAge (I) 'جمع محتوى الخانة الحاليّة على المجموع السابق**

**Next**

**MsgBox ( Sum / 50)**

وحتّى لا تظنّ أن المصفوفة مخصّصة للأرقام، انظر إلى المصفوفة التالية، والتي نخزّن بها ثلاثة ألوان:

**Dim Colors(2) As Color**

**Colors (0) = Color.Red**

**Colors (1) = Color.Blue**

**Colors (2) = Color.Green**

إنّ كائن الألوان Color، يمنحك من بين خصائصه، أسماء 128 لونا، يمكنك اختيار ما تشاء منها.

**ملحوظة:**

كل خانات المصفوفة لها نفس النوع، ولكن لو عرّفت مصفوفة من النوع "كائن" Object، فيمكنك أن تضع في كل خانة قيمة مختلفة في النوع عن الأخرى.

**وضع القيم الابتدائيّة للمصفوفات Initializing Arrays:**

يمكنك أن تضع القيم الابتدائيّة للمصفوفة في نفس جملة التعريف بالصيغة التالية:

**Dim اسم\_المصفوفة() As نوع\_البيانات = {قيمة\_0, قيمة\_1, … قيمة\_ن}**

وكمثال:

**Dim Names() As String = {"فكري", "حسّان"}**

وهو نفس ما يمكن أن تفعله بالجمل الثلاث التالية:

**Dim Names(1) As String**

**Names(0) = " فكري "**

**Names(1) = "حسّان"**

أو بالجملتين التاليتين:

**Dim Names() As String**

**Names = New String() {"فكري", "حسّان"}**

ما يجب أن تعرفه هنا، هو أنّ اللغة لن تقبل منك جملة كالتالية:

**Dim Names(1) As String = {"فكري", "حسّان"}**

لأنّك ارتكبت خطأ قاتلا، وحاولت أن تضع قيما ابتدائيّة لمصفوفة تم تحديد عدد الخانات لها!!

ألا يبدو لك هذا غباء من اللغة؟!

يحدث هذا بينما تبدو اللغة في منتهى المرونة في نواحيَ عديدة.. فمثلا يمكنك أن تملأ خانات المصفوفة بطريقة أخرى، بوضع مصفوفة أخرى فيها مباشرة:

**Dim X(2) As Integer**

**Dim Y() As Integer = {1,3,4,5}**

**X = Y**

**MsgBox X(3) ' ستعرض الرسالة الرقم 5**

تلاحظّ أنّه بمجرّد كتابة X = Y، قد صارت المصفوفة X نسخة طبق الأصل من المصفوفة Y، لدرجة أن عدد خانات X قد زاد بواحد عمّا عرّفناه سابقا، بالإضافة إلى أنّ هذه الخانات قد صارت تحتوي على قيم خانات المصفوفة Y.. منتهى المرونة والسهولة.

قيد واحد فقط هو المفروض عليك في هذه الحالة: أن تكون المصفوفتان من نفس النوع: لهما نفس عدد الأبعاد، ولهما نفس نوع البيانات.. فمثلا، سيحدث خطأ لو كانت إحداهما ثنائية البعد بينما الأخرى أحاديّة البعد، أو كانت إحداهما نصّيّة والأخرى رقميّة، أو حتّى كانت إحداهما تستوعب أعدادا صحيحة Integers، والأخرى تستوعب أعدادا مزدوجة Double.

**حدود المصفوفة Array Limits:**

يمكنك أن تعرف رقم الخانة الأخيرة في المصفوفة، باستخدام الدالة "الحدّ الأعلى" UBound()، التي جاء اسمها اختصارا لتعبير Upper Bound، وهي تأخذ اسم المصفوفة كمعامل، وتُرجع رقم آخر خانة بها.. ويمكنك أن تحصل على نفس النتيجة، باستخدام الدالة "اقرإ الحدّ الأعلى" GetUpperBound، التي تنتمي لوسائل المصفوفة.. طبعا لم يدهشك الأمر: نعم.. حتّى المصفوفة هي نوع من الكائنات، ولها خصائصها ووسائلها:

**Dim MyArray(19) As Integer**

**MsgBox (UBound(MyArray)) ' ستعرض الرسالة الرقم 19**

**MsgBox (MyArray.GetUpperBound(0)) ' ستعرض الرسالة الرقم 19**

تلاحظ أنّنا أرسلنا الرقم صفر كمعامل للوسيلة GetUpperBound.. إنّ هذه الوسيلة لا تحتاج لاسم المصفوفة، لأنّها تُنفّذُ على المصفوفة التي تنتمي إليها، ولكنّها في المقابل تحتاج لرقم البعد Dimension الذي ستحسب خانته الأخيرة.. ما هو البعد؟.. أسمعك تتساءل.. حسنا: سنشرح ذلك بعد قليلٍ فاصبر!

ومن المواضع الهامّة لاستخدام هذه الدالة، الجمل التكراريّة، فبدلا من تحديد رقم الخانة الأخيرة رقميّا يمكنك استخدام كود كالتالي:

**Dim I As Integer, MyArray(19) As Integer**

**For I = 0 To MyArray.GetUpperBound(0)**

**MyArray(I) = I \* 1000**

**Next**

ويمكنك استخدام الخاصيّة "طول" Length لمعرفة طول المصفوفة:

**Console.WriteLine(MyArray.Length)**

الطريف أيضًا، أنّك تستطيع استخدام الوسائل الخاصة بنوع خانات المصفوفة، فلو كانت المصفوفة من النوع عدد صحيحInteger مثلا، يمكن استخدام وسائل الأعداد الصحيحة لكل خانة، كالتالي:

**MyArray(0) = 3**

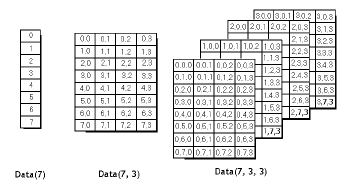
**I = MyArray(0).MinValue**

**MsgBox (MyArray(0).ToString)**

وبالمثل، لو كانت المصفوفة نصّيّة String، يمكن استخدام وسائل النصوص لكل خانة، ولو كانت المصفوفة زمنيّة Date، يمكن استخدام وسائل النصوص لكل خانة.... وهكذا.

**المصفوفة متعدّدة الأبعاد Multidimensional Arrays:**

كل ما تعاملنا مع من المصفوفات حتّى الآن، هو المصفوفات أحاديّة البعد.. ولكنّك كثيرا ما ستحتاج لاستخدام مصفوفات متعدّدة البعد، ثنائيّة وثلاثيّة وأكثر.



فمثلا، ستحتاج لمصفوفة ثنائيّة لتمثيل عشرة مدن ودرجات حرارتها في أسبوع.. إنّ المصفوفة ثنائيّة البعد تمثّل جدولا: صفوفا وأعمدة.. وفي حالتنا هذه، سيكون الصف هو رقم المدينة، والعمود هو رقم اليوم في الأسبوع.

طبعا يمكنك أن تستنتج أن المصفوفة الثنائيّة تحتاج لرقمين للوصول لأي خانة: رقم الصف ورقم العمود.

**Dim City(9, 6) As Integer ' عشر مدن × سبع أيام**

**City(0,0) = 23 ' درجة حرارة المدينة الأولى في اليوم الأوّل 23 درجة**

كما تلاحظ، الخانة الأولى تمثّل الصفّ، والخانة الثانية تمثّل العمود.

يمكن أيضا أن تكون لديك مصفوفة ثلاثيّة، مثل:

**Dim City(9, 6, 3) As Integer**

طبعا سترهق ذهنك في محاولة تخيّل الأبعاد الثلاثيّة ثمّ الرباعيّة ... إلخ.. لا تحاول.. إنّها طرق رياضيّة لإيضاح ارتباط مجموعة معيّنة من المعاملات والبيانات.. ففي المثال السابق، يمكن أن يمثّل البعد الأوّل 10 مدن، والثاني 7 أيام، والثالث 3 قراءات في اليوم (في الفجر = 0، في الظهيرة = 1، في الغروب = 2).

فلكي تحدّد أن درجة الحرارة في المدينة الثانية (الإحداثي الأوّل = 1) في اليوم الثالث (الإحداثي الثاني = 2) في وقت الظهيرة (الإحداثي الثالث = 1) كانت 40 درجة (قيمة الخانة = 40)، يمكنك أن تكتب:

**City(1,2,1) = 40**

ولك أن تتوقّع، أنّ هناك وسيلة ما لوضع القيم الابتدائيّة في المصفوفات متعدّدة البعد.. انظر لما يلي:

**Dim A(,) As Integer = { {10, 20, 30}, {11, 21, 31}, {12, 22, 32} }**

**Console.WriteLine(A(0, 1)) ' الناتج 20**

**Console.WriteLine(A(2, 2)) ' الناتج 32**

لاحظ أن أكبر خانة في المصفوفة السابقة هي (2،2)، لأنّ المصفوفة 3 صفوف × 3 أعمدة.

هنا ستحتاج لخاصيّة هامّة من خواصّ المصفوفة، هي خاصّيّة "الرتبة" Rank، التي يمكنك استخدامها لمعرفة عدد أبعاد المصفوفة:

**Dim Salaries(1,99) As Object**

**MsgBox(Salaries.Rank) ' ستعرض الرسالة 2**

وأحب أن ألفت نظرك إلى شيء: عندما تستخدم خاصّيّة "الطول" Length في مصفوفة متعددة الأبعاد، فإنّ الناتج سيكون هو الطول الكلّيّ للمصفوفة، الناتج من ضرب أطوال كل أبعادها:

**MsgBox (Salaries.Length) ' ستعرض الرسالة "200" (2×100)**

ولكن ماذا لو أحببت أن تعرف طول بعد بعينه؟.. في هذه الحالة استخدم الدالة "اقرإ الطول" GetLength، وابعث لها البعد الذي تقصده كمعامل:

**Console.WriteLine(Salaries.GetLength(0)) ' الناتج 2**

**المصفوفات المرنة Dynamic Arrays:**

كثيرا ما تجد نفسك عاجزا عن أن تعرف بالضبط حجم مصفوفتك قبل تشغيل التطبيق، أو قد يحتاج المستخدم لأن يُدخل قيمًا جديدة باستمرار، ممّا يجعلك في حاجة لزيادة طول المصفوفة.. في هذه الحالة تحتاج إلى مصفوفة مرنة، قابلة لزيادة حجمها أو إنقاصه في أيّ وقت، لتتلاءم مع عدد الخانات التي لديك.

ولفعل ذلك، ابدأ بتعريف المصفوفة بقوسين فارغين، بمعنى ألا تحدّد لها طولا معيّنا:

**Dim DynArray() As Integer**

ويمكن وضع القوسين الفارغين بعد نوع البيانات:

**Dim DynArray As Integer()**

**ملحوظة:**

قد يصيبك بعض الارتباك من الصيغة الأخيرة، خاصّة وأنّها تتشابه مع صيغة تعريف الكائنات (باستخدام كلمة New).. إنّ الجملة التالية تعرّف عددا صحيحا منفردا:

**Dim StudentAge As New Integer()**

بينما لو أزلت كلمة New وتركت القوسين فستؤدّي الجملة لتعريف مصفوفة أعداد صحيحة.. انتبه لهذا جيّدا، ولا تنسَ أن تحذف القوسين عند حذف كلمة New حتّى لا يختلف معنى الجملة عن الذي تريده.

الجدير بالذكر، أنّك لا تستطيع استخدام كلمة New في تعريف مصفوفة كائنات.. فالجملة التالية مرفوضة تماما:

**Dim X() As New Form1()**

فماذا تفعل لو أردت أن تنشئ نسخة جديدة من كلّ كائن في خانات المصفوفة؟

في هذه الحالة استخدم هذه الصيغة:

**Dim X() As Form1 = {New Form1()}**

وبهذا تحصل على مصفوفة، تحتوي كلّ خانة منها على نسخة جديدة من النموذج (أو أيّ كائن تريده).

ثم حينما تجد نفسك أثناء البرنامج محتاجا لحجز عدد من الخانات، استخدم تعبير"أعد تعريف" ReDim كالتالي:

**ReDim DynArray(UserCount)**

حيث UserCount هو متغيّر رقميّ، يحتوي على عدد أدخله المستخدم، وليكن 20.

وجملة "أعد تعريف" ReDim لا تُستخدم فقط إلا في إجراء أو دالة أو خاصّيّة، فهي ليست مثل Dim، ولا تظهر أبدا على مستوى النموذج.

كما يمكن تحديد أبعاد المصفوفة بجملة ReDim:

**Dim Matrix( , , ) As Double**

**ReDim Matrix(9, 9, 9)**

ولكن لا يمكنك تغيير عدد هذه الأبعاد فيما بعد، فجملة كالتالية لن تكون مقبولة:

**ReDim Matrix(9, 9)**

وإن كان من الممكن تغيير طول أي بعد من الأبعاد:

**ReDim Matrix(9, 2, 3)**

**ملاحظات:**

1- لا يمكن تغيير نوع المصفوفة بجملة ReDim.. هذا هو السبب في غياب الجزء الخاص بتعبير As منها.

2- يمكن استخدام جملة ReDim لتغيير حجم المصفوفات ثابتة الحجم كذلك!.. انظر المثال التالي:

**Dim X(3) As Integer**

**ReDim X(4)**

3- في الواقع، إنّ الملاحظة السابقة تقولُ بوضوح: إنّ اللغة تتعامل مع كلّ المصفوفات على أنّها مرنة، لدرجة أنّك صرت تستطيع كتابة شيء كان ممنوعا في النسخ السابقة وفي باقي لغات البرمجة، ألا وهو تعريف عدد خانات المصفوفة بمتغيّر، شريطةَ أن يكون قد تم تعريفه في أي سطر يسبق تعريف المصفوفة، وذلك حتّى تكون له قيمة مبدئيّة، سواء القيمة الافتراضيّة 0، أو أي قيمة ابتدائيّة تضعها فيه أثناء تعريفه، أو أي قيمة أخرى تضعها فيه بعد أن تتلقاها من مربع نص أو أي وسيلة إدخال يستخدمها المستخدم.. إنّ كل الجمل التالية مباحة:

* **Dim I As Integer**

**Dim X(I) As Integer ' I = 0**

* **Dim I As Integer = 3**

**Dim X(I) As Integer**

* **Dim I As Integer**

**I = Val( Text1.Text)**

**Dim X(I) As Integer**

ولكنّك ما زلت قلقا، تقضم أظافرك وتتساءل:

- ولكن ماذا عن البيانات الموجودة في المصفوفة؟.. ماذا سيحدث لها بعد تغيير حجم المصفوفة؟

- ستُمحى تماما ونهائيًّا!

- ماذا؟!!.. ولكنّني أريد الاحتفاظ بالبيانات السابقة.. هذا سيجعل هذه الوسيلة عقيمة بالنسبة لي!

- ليس في كلّ الأحوال، فهذه الطريقة نافعة جدا حينما تريد حجز مصفوفة فارغة بالحجم الذي تتطلّبه بيانات المستخدم، كعدد الطلبة في فصل مثلا، وهي تتغيّر من فصل لفصل.

- ولكنّ الحالة الأهمّ، هي عندما أريد زيادة حجم المصفوفة لاستيعاب بيانات جديدة على البيانات الحاليّة دون أن أفقدها.

- حسنا حسنا.. لا تبتئس هكذا.. هناك حل!

- حقًّّا؟!.. إليّ به.

- يمكنك استخدام كلمة "احتفاظ" Preserve، للاحتفاظ بالبيانات السابقة في المصفوفة.. فلو أزدت عدد الخانات، تصاف إلى المصفوفة خانات جديدة مع الاحتفاظ بقيم خاناتها القديمة، ولو أنقصت عدد الخانات، ضاعت البيانات الموجودة في الخانات التي نقصت فقط، واحتفظت ببيانات باقي الخانات.

انظر لهذا المثال:

**ReDim Preserve DynArray(UBound(DynArray) + 1)**

لقد أضفنا خانة جديدة للمصفوفة، بزيادة 1 على حدّها الأعلى.. هذا مع عدم ضياع القيم القديمة من المصفوفة.

وهناك طريقة أخرى لتغيير حجم المصفوفة المرنة، ولكن دون حفظ محتوياتها السابقة، وذلك باستخدام كلمة New كالتالي:

**DynArray = New Integer(30) {}**

لاحظ حتميّة وجود القوسين المتعرّجين.. وحتّى لا تظنّ أنّهما بلا فائدة، فإنّ بإمكانك أن تستخدمهما في وضع القيم الابتدائيّة للمصفوفة:

**DynArray = New Integer(3) {1,2,4,5}**

وهناك عمليّة عكسيّة لحجز المصفوفة، هي محوها، وذلك إذا أردت أن تمحو كلّ خانات المصفوفة، لتوفير مساحة الذاكرة بعد الانتهاء من استخدامها.. والأمر بسيط.. كلّ ما عليك هو أن تستخدم جملة المحو Erase كالتالي:

**Erase DynArray**

ويمكنك أن تمحو بجملة واحدة أكثر من مصفوفة، وذلك بوضع فواصل بين أسمائها:

**Erase DynArray1, DynArray2, DynArray3**

**مصفوفة المصفوفات Arrays of Arrays:**

لو عرّفت مصفوفة من النوع "كائن" Object، فإن بإمكانك أن تضع في خاناتها أيّ شيء، حتّى لو كان مصفوفة!.. نعم: يمكنك أن تضع مصفوفة في خانة من مصفوفة من النوع "كائن" بعملية تساوٍ عاديّة.. انظر لهذا المثال:

**Dim X(2) As Object, Y(2) As Object, Z(3) As Int16**

**Z(0) = 1**

**Y(0) = Z**

لقد وضعنا المصفوفة Z بكاملها في الخانة الأولى من المصفوفة Y.. الآن يمكن استخدام التعبير Y(0) كأنّه هو المصفوفة Z:

**MsgBox(Y(0)(0))**

هذه الرسالة ستعرض 1، لأنّ Y(0)(0) هو تعبير مكافئ لـ Z(0).. تعال نعقّد العمليّة أكثر.. سنضع المصفوفة Y في إحدى خاناتها هي نفسها!!

**Y(1) = 2**

**Y(2) = Y**

الآن يصير التعبير Y(2) مكافئا للمصفوفة Y نفسها، وبالتالي يصير التعبير Y(2)(0) مكافئا للتعبير Y(0)، الذي يكافئ بدوره المصفوفة Z:

**MsgBox(Y(2)(0)(0) + Y(2)(1))**

سيكون الناتج هو 3، نتيجة لجمع الخانة الأولى من المصفوفة Z مع الخانة الثانية من المصفوفة Y.. ويمكنك تعقيد العمليّة لأيّ درجة تريدها:

**X(0) = Z**

**X(1) = Y**

**MsgBox(X(1)(2)(0)(0)) ' الناتج 1**

**X(2) = X**

**MsgBox(X(2)(1)(2)(0)(0)) ' الناتج 1**

**ملاحظة:**

لن تعمل هذه الطريقة في حالة اختيار "التحويل الدقيق" Option Strict On، لأنّه نوع من الربط المتأخّر Late Binding، فالكود الذي تكتبه يشير إلى أن إحدى خانات المصفوفة تحتوي على مصفوفة أخرى، وهو ما لا يمكن لمترجم الكود أن يتحقّق من صحّته، إلا عند تنفيذ البرنامج بالفعل.

تلك كانت طريقة غير مباشرة، ولكن بإمكانك أن تعرّف مصفوفة مصفوفات مباشرةً كالتالي:

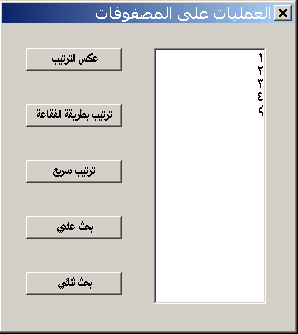
**Dim X() () As Integer**

والتعامل معها سيبدو معقّدا قليلا، لهذا لن نناقشها هنا.

**مشروع المصفوفات:**

آن الأوان لكي نتدرّب على استخدام بعض ما تعلّمناه حتّى الآن عن المتغيّرات والمصفوفات.. وسنفعل ذلك عن طريق إنشاء تطبيق يأخذ القيم التي يدخلها المستخدم في مربّع نصّ، ويضعها في مصفوفة، مانحا للمستخدم القدرة على عكس هذه المصفوفة أو ترتيبها أو البحث فيها.. ورغم أنّ VB.Net تمنحك الآن وسائل جاهزة عكس المصفوفة وترتيبها والبحث فيها (سنتعرّف عليها بالتفصيل في فصل المجموعات Collections)، إلا إنّه سيكون تدريبا مفيدا أن نبني هذه الوسائل بأنفسنا، حتّى نتعرّف على بعض المهارات البرمجيّة الأساسيّة.. ستجد هذا التطبيق في مجلّد هذا الفصل باسم Arrays.

ابدأ بتصميم واجهة التطبيق كما في الصورة التالية:



وحتّى يستطيع المستخدم كتابة أكثر من سطر في مربّع النصّ، لا تنسَ أن تغيّر خاصيّة "متعدّد الأسطر" MultiLine إلى صواب True.

**قراءة المدخلات:**

نريد الآن أن نخزّن كلّ سطر في مربّع النصّ في خانة في مصفوفة.. أبسط طريقة لفعل هذا هو استخدام الخاصيّة Lines الخاصّة بمربّع النصّ.. هذه الخاصيّة تعيد لك مصفوفة نصيّة، تحتوي كلّ خانة منها على سطر من سطور مربّع النصّ.. إذن فكلّ ما نحتاجه هو السطران التاليان:

**Dim Arr() As String**

**Arr = TextBox1.Lines**

ولكن قبل أن نفعل ذلك، يجب علينا أولا أن نتأكّد من أنّ المستخدم قد أدخل بعض العناصر في مربّع النصّ، حتّى لا يحدث أيّ خطأ غير متوقّع نتيجةً للتعامل مع مربّع نصّ فارغ.

لهذا سنتستخدم الكود التالي:

**If TextBox1.Text = "" Then**

**MsgBox("لا تحاول التذاكي علينا.. أدخل أيّ نصّ في مربّع النصّ أولا")**

**TextBox1.Focus() 'ضع مؤشّر الكتابة في مربّع النصّ**

**Exit Sub ' عدم إكمال العمليّة**

**End If**

**البحث في المصفوفة:**

أبسط طريقة يمكن أن نستخدمها للبحث في المصفوفة هي في نفس الوقت أسوأ وأبطأ طريقة، وذلك بالمرور عبر كلّ خانات المصفوفة خانة خانة، ومقارنة نصّ البحث بنصّ الخانة.

وتوجد طرق أفضل للبحث، مثل البحث الثنائي Binary Search، الذي يعدّ أسرع بكثير من الطريقة الأولى.

ولكّنا سنقتصر على كتابة الطريقة الأولى في هذا الفصل، نظرًا لأنّ البحث الثنائيّ يتطلّب معرفة جيّدةً بالبرمجة الارتداديّة Recursive Programming، وسنتعرّف عليها بالتفصيل في الفصل القادم بإذن الله.

تعالَ نرَ البحثَ التقليديّ:

في البداية يجب أن نطلب من المستخدم أن يُدخل النصّ الذي يريد البحث عنه:

**Dim S As String = InputBox("")**

فإذا ترك المستخدم مربّع الإدخال فارغا، فعلينا ألا نقوم بالبحث:

**If S = "" Then**

**MsgBox ("نصّ البحث فارغ.. لا تحاول التذاكي علينا!")**

**Exit Sub**

**End If**

فإذا كان هناك نصّ بحث، فهذا هو الكود الذي يبحث عنه:

**Dim Found As Boolean, I As Integer**

**For I = 0 To Arr.GetUpperBound(0)**

**If S = Arr(I) Then**

**MsgBox("نصّ البحث موجود في السطر رقم " + Str(I + 1))**

**Found = True**

**' يمكنك استخدام هذه الجملة لو أردت الاكتفاء بالعثور على أوّل نسخة من النص**

**' Exit For**

**' ومن دونها ستستمرّ جملة التكرار لنهاية المصفوفة،**

**' وقد تعثر على نفس النصّ أكثر من مرّة**

**End If**

**Next**

**' لو لم نجد النصّ فيجب أن نخبر المستخدم بهذا**

**If Found = False Then MsgBox("لم يتمّ العثور على نصّ البحث")**

**عكس خانات المصفوفة:**

تعالَ نفكّر معا بصوتٍ مسموع:

كيف نعكس خانات المصفوفة؟

سنضع في الخانة الأولي محتويات الخانة الأخيرة، ونضع في الخانة الأخيرة محتويات الأولى.

ثمّ سنفعل المثل في الخانتين الثانية وقبل الأخيرة.

ثمّ الخانة الثالثة وقبل قبل الأخيرة (الثالثة من آخر المصفوفة).

وهكذا... .

فكأنّنا وضعنا مرآةً عند منتصف المصفوفة، لنعكس كلّ خانة مع صورتها في المرآة.

ولكن كيف تعرف صورة الخانة في هذه المرآة؟

لاحظ ما يلي:

الخانة رقم . تقابلها الخانة الأخيرة.

والخانة رقم 1 تقابلها تلك التي تسبق الأخيرة بـ 1.

والخانة رقم 2 تقابلها تلك التي تسبق الأخيرة بـ 2.

والخانة رقم I تقابلها تلك التي تسبق الأخيرة بـ I.

إذن، لو كان رقم الخانة الحاليّة I، فإنّ رقم صورتها في المرآة هو J، حيث:

**J = Arr.GetUperBound(0) – I**

وأنت بالطبع تعلم أن Arr.GetUperBound(0) هو رقم الخانة الأخيرة في المصفوفة.

ولكن متى تتوقّف هذه العمليّة؟

تتوقّف بالطبع عند الوصول إلى منتصف المصفوفة (المرآة)، ففي هذه الحالة سيكون النصف الأيمن قد انتقل للنصف الأيسر والعكس.

أعتقد الآن أنّك تستطيع فهم هذا الكود، الذي يقوم بعكس المصفوفة:

**Dim Temp As String, I As Integer, J As Integer**

**Dim Up As Integer = Arr.GetUpperBound(0)**

**If CheckTextBox() = False Then Exit Sub**

**For I = 0 To Up \ 2**

**J = Up - I**

**' Swap تبادل قيمتي الخانتين**

**' يجب حفظ قيمة الخانة الحاليّة في متغيّر احتياطي،**

**' قبل الكتابة في هذه الخانة، وذلك لأنّنا نحتاجها لكي توضع في الخانة المناظرة**

**Temp = Arr(I)**

**' نقل قيمة الخانة المناظرة إلى الخانة الحاليّة**

**Arr(I) = Arr(J)**

**' نقل قيمة الخانة الأولى من المتغيّر الاحتياطي إلى الخانة المناظرة**

**Arr(J) = Temp**

**Next**

الآن تحتوي المصفوفة Arr على نسخة معكوسة من سطور مربّع النصّ.. لم تتبقّ إذن إلا خطوةٌ واحدة، هي وضع المصفوفة المعكوسة في مصفوفة سطور مربّع النصّ، ليرى المستخدم التغيير الذي حدث:

**TextBox1.Lines = Arr**

**ترتيب المصفوفة:**

هناك طرق عديدة لترتيب المصفوفات، سنكتفي هنا بشرح إحداها، وهي طريقة الفقّاعة Bubble Sort، وسنحيل الطريقة الأخرى الأفضل والأسرع ـ وهي طريقة الترتيب السريع Quick Sort ـ إلى الفصل القادم إن شاء الله.

أعرف أنّك ستسألني في لماضة:

- ما هو ترتيب الفقاقيع هذا؟.. وهل لهذا علاقة بالصابون والمنظّفات؟

- يبدو أنّك على حقّ!.. فهذه الطريقة في الترتيب تستخدم نظريّة: الأخفّ يطفو!

- كمن فسّر الماءَ بعدَ العُسرِ بالماءِ!.. هل يبدو لك أنّك أضفت لي جديدا؟

- اصبر قليلا وستفهم.

- أطل اللهمّ الروح.. تفضّل يا سيّدي!

- الحكايةُ وما فيها أنّنا سنقارن قيمة كلّ خانة في الصفوفة بما يليها، فإذا كانت الخانة التالية أصغر، يتمّ تبادل قيمتها مع الخانة الحاليّة.. إنّ هذا في النهاية سيجعل العناصر الأصغر تطفو إلى بداية المصفوفة.. تماما كما تفعل الفقاقيع حينما تتصاعد إلى سطح الماء.

- أعتقد أنّني فهمت، ولكنّني ما زلت في حاجةٍ لبعض الإيضاح.

- إذن فلنأخذ مثالا عمليّا.

افترض أنّ لدينا المصفوفة التالية:

|  |
| --- |
| 5 |
| 2 |
| 6 |
| 4 |
| 3 |

سنبدأ بمقارنة الخانة الأولى بما يليها.. سنكتشف أنّ بها الرقم 5 وهو أكبر من الرقم 2.. إذن فلا بدّ من مبادلتهما:

|  |
| --- |
| 2 |
| 5 |
| 6 |
| 4 |
| 3 |

لاحظ أنّنا سنستمرّ في مقارنة الخانة الأولى بما يليها وكأنّ شيئا لم يكن.. الفارق الوحيد أنّها صارت تحتوي على الرقم 2.. لحسن الحظّ أنّه أصغر من الرقم 6 والرقم 4 والرقم 3.

ننتقل الآن للخانة الثانية.. سنجد أن الرقم 5 أقلّ من الرقم 6، ولكنّه أكبر من الرقم 4.. إذن فلا بدّ من "تعويم" الرقم 4 إلى أعلى:

|  |
| --- |
| 2 |
| 4 |
| 6 |
| 5 |
| 3 |

وكما اعتدنا سنواصل باقي عمليّة المقارنة، ولكن بالأخذ في الاعتبار أنّ الخانة الثانية صارت تحتوي على الرقم 4.. هنا سنجد أنّه أكبر من الرقم 3، ممّا يعني حتميّة مبادلتهما:

|  |
| --- |
| 2 |
| 3 |
| 6 |
| 5 |
| 4 |

الآن نصل للخانة الثالثة، وفيها الرقم 6.. للأسف سنجده أكبر من الرقم 5 (لاحظ أنّ هذه هي المرّة الثانية التي نقارن فيها هذين الرقمين.. إنّ هذا من عيوب هذه الطريقة، حيث يكون عدد المقارنات كبيرا).. الآن سنبادلهما:

|  |
| --- |
| 2 |
| 3 |
| 5 |
| 6 |
| 4 |

وسنواصل المقارنة على الخانة الثالثة وفيها الرقم 5.. سنجد أنّه أقلّ من الرقم 4.. فلنقُم بعمل اللازم:

|  |
| --- |
| 2 |
| 3 |
| 4 |
| 6 |
| 5 |

نصل الآن لمقارنة الخانة الرابعة بما يليها.. سنجد فيها الرقم 6.. آه.. هذه هي المرّة الثالثة التي نقارنه فيها بالرقم 5!!.. أليس شيئا مملا؟

طبعا يجب تعويم الرقم 5.

|  |
| --- |
| 2 |
| 3 |
| 4 |
| 5 |
| 6 |

نصل الآن للخانة الأخيرة.. لم يعد هناك ما نقارنها به.. إذن فقد انتهت العمليّة.. وطبعا المصفوفة الآن مرتّبة!.. لاحظ أنّنا نستطيع الاستغناء عن مقارنة الخانة الأخيرة مع بعدها، لأنّها بالفعل بلا فائدة، فلا توجد أيّ خانات بعدها!!

وقبلَ أن تطلقَ أيّ اعتراض على هذه الطريقة وما بها من عيوب، يجب أن أُحيطك علمًا بأن كتابة الكود لهذه الطريقة يعدُ أسهل ما يمكن، وهي الميزة الوحيدة في الموضوع!!

تعال نرَ هذا الكود:

**Dim Temp As String, I, J As Integer**

**Dim Up As Integer = Arr.GetUpperBound(0)**

**For I = 0 To Up -1 ' جملة تكراريّة لكلّ خانة من خانات المصفوفة**

**For J = I + 1 To Up ' جملة تكراريّة لكلّ الخانات التالية للخانة الحاليّة**

**If Arr(I) > Arr(J) Then**

**' Swap تبادل قيمتي الخانتين**

**Temp = Arr(I)**

**Arr(I) = Arr(J)**

**Arr(J) = Temp**

**End If**

**Next**

**Next**

لاحظ أنّ هذا ترتيب هجائيّ، فالمصفوفة نصّيةّ.. فماذا لو أردنا أن نكتب في مربّع النصّ مجموعة من الأرقام ونرتّبها؟

في هذه الحالة سيتمّ ترتيبها هجائيّا، بمعنى أنّ 11 ستوضع قبل 2 و 34 ستوضع قبل 5!!

فهل هناك طريقة لإجراء ترتيب رقميّ؟

بسيطة جدّا.. كلّ ما سنفعله هو تغيير سطر مقارنة خانتي المصفوفة في الإجراء السابق إلى السطر التالي:

**If Val(Arr(I)) > Val(Arr(J)) Then**

إنّ دالة القيمة Val() تحوّل النصّ إلى قيمة رقميّة ـ إذا كان قابلا للتحويل (إذا كان نصّا معبّرا عن رقم).. وبهذا سيتمّ إجراء الترتيب بين الأرقام الموجودة في المصفوفة باعتبارها أرقاما هذه المرّة لا نصوصا.

وستجد الكود الكامل لهذا التطبيق في مجلّد برامج هذا الفصل.

**الأنواع التي يُعرّفها المستخدم User-Defined Data Types**

* + [**المُرقَّمَات Enumerators:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\03-%20المتغيرات\15-%20الأنواع%20التي%20يُعرّفها%20المستخدم\1-%20المُرقَّمَات.htm)
  + [**السجلات Structures:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\03-%20المتغيرات\15-%20الأنواع%20التي%20يُعرّفها%20المستخدم\2-%20السجلات.htm)

**المُرقَّمَات Enumerators**

* + - [**تعريف المُرقَّمَات:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\03-%20المتغيرات\1-%20المُرقَّمَات\1-%20تعريف%20المُرقَّمَات.htm)
    - [**المرقّمات كمؤشّرات Flags:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\03-%20المتغيرات\1-%20المُرقَّمَات\2-%20المرقّمات%20كمؤشّرات.htm)
    - [**بعض وسائل خليّة المرقّم Enum Class:**](file:///C:\Users\Administrator\Desktop\vb.net\B-%20أساسيّات%20اللغة\03-%20المتغيرات\1-%20المُرقَّمَات\3-%20بعض%20وسائل%20خليّة%20المرقّم.htm)

**تعريف المُرقَّمَات:**

من الأنواع التي تستطيع تعريفها، المُرقَّمات: وهي أنواع تحتوى على تعبيرات نصّيّة مناظرة لمجموعة من الأرقام، لاستخدامها بدلا من هذه الأرقام عند كتابة الكود، ليصبح أكثر سهولة ووضوحا عند قراءته.

وتُعرّف المُرقَّمات في بداية النموذج أو القالب أو الخليّة كالتالي:

**Private Enum اسم\_المرقم**

**تعبير1 = قيمة1**

**…….**

**تعبير\_ن = قيمة\_ن**

**End Enum**

هذا مع إمكانيّة استخدام مجال عامّ Public أو "صديق" Friend للمرقّم بدلا من Private.

مثال: مرقّم أيام الأسبوع:

**Private Enum WeekDays**

**السبت = 0**

**Saturday = 0**

**الأحد = 1**

**Sunday = 1**

**الاثنين = 2**

**Monday = 2**

**الثلاثاء = 3**

**Tuesday = 3**

**الأربعاء = 4**

**Wednesday = 4**

**الخميس = 5**

**Thursday = 5**

**الجمعة = 6**

**Friday = 6**

**End Enum**

طبعا تلاحظ أنّنا أسمينا الأيّام بالعربيّة والإنجليزيّة، ومنحنا اسمي نفس اليوم في اللغتين نفس الرقم، حتّى يمكن استخدام أحدهما مكان الآخر بدون قيود.. لقد تعمّدتُ فعل هذا حتّى أريك أن المرقّم يمكن أن يحتوي على أكثر من اسم لنفس القيمة الرقميّة بدون مشاكل.

والآن تعال نستخدم هذا النوع:

**Dim X As WeekDays**

**X = WeekDays.السبت**

**If X = WeekDays.Saturday Then MsgBox ("القيمتان متكافئتان")**

ولا يوجد ما يمنعك من استخدام الرقم المناظر للتعبير الرقميّ مباشرة.. فمثلا الكود التالي مماثل تماما لذلك الذي في المثال السابق:

**Dim X As WeekDays**

**X = 0**

**If X = WeekDays.Saturday Then MsgBox ("القيمتان متكافئتان")**

ويمكن تعريف المرقّم على الصيغة المختصرة التالية:

**Private Enum DigitNames**

**Zero**

**One**

**Two**

**Three**

**Four**

**Five**

**Six**

**Seven**

**Eight**

**Nine**

**End Enum**

حيث في هذه الحالة سيُعتبر أوّل عضو هو القيمة صفر، والثاني هو القيمة 1،.. وهكذا.. أيّ أنّ التعريف السابق هو اختصار لهذا التعريف:

**Private Enum DigitNames**

**Zero = 0**

**One = 1**

**Two = 2**

**Three = 3**

**Four = 4**

**Five = 5**

**Six = 6**

**Seven = 7**

**Eight = 8**

**Nine = 9**

**End Enum**

كما يمكن تحديد نوع قيم المرقّم.. انظر للمثال التالي:

**Private Enum DigitNames As Byte**

**Zero**

**One**

**Two**

**Three**

**Four**

**Five**

**Six**

**Seven**

**Eight**

**Nine**

**End Enum**

**ملحوظة لمبرمجي VB6:**

نظرًا للكم الهائل من التعبيرات الرقمية المعرّفة في اللغة، بالإضافة لما يمكن أن تُعرّفه أنت، فإن هناك احتمالا كبيرا في حدوث تشابه بين أسمائها، وهو ما ترفضه قواعد اللغة، هذا بالإضافة لصعوبة حفظك لها، وهو نقيضُ للهدف الذي وُضعت من أجله.. لهذا كان من الأفضل عزل كل مجموعة من التعبيرات الرقمية داخل المُرَقّم الذي تنتمي إليه، وبهذا يمكن السماح بتكرار أسماء التعبيرات الرقمية ومنحها قيمًا مختلفة، ما دام لكل منها مدى Domain مختلف.. هذا هو السبب في أن التعبيرات الرقمية أصبحت تكتب كالتالي:

**X = WeekDays.Saturday**

بكتابة اسم المرقّم قبل التعبير الرقميّ، على خلاف ما كنت تكتبه في النسخ السابقة، حيث كانت التعبيرات الرقمية تستخدم مباشرة كالتالي:

**X = Saturday**

معنى هذا أنّ مرقَّما اسمه Enum1 يمكن أن يحتوي على التعبير Value1 وقيمته 1، في نفس اللحظة التي يحتوى فيها مرقم اسمه Enum2 على التعبير Value1 وقيمته 2، لأنّ مترجم الكود Compiler لن يرتبك بينهما، فنحن نستخدمهما كالتالي:

**X = Enum1.Value1**

**'بعد تنفيذ هذه الجملة، ستصبح قيمة X تساوي 1**

**X = Enum2.Value1**

**'بعد تنفيذ هذه الجملة، ستصبح قيمة X تساوي 2**

**المرقّمات كمؤشّرات Flags:**

ماذا لو أردت أن تضع في متغيّر قيمتين أو أكثر معا؟

إنّ أهمّيّة ذلك لا تتّضح في مرقّم أيّام الأسبوع، ولكنّها ستتّضح بالتأكيد في المرقّم التالي:

**Enum FileAttribute**

**عادي**

**للقراءة\_فقط**

**خفيّ**

**مضغوط**

**End Enum**

هذا المرقّم يعبّر عن سمات الملفّ.. المشكلة أنّ الملفّ يمكن أن يمتلك أكثر من سمة في نفس الوقت، إذ يمكن أن يكون خفيّا وللقراءة فقط.. أو مضغوطا وخفيّا.. أو مضغوطا وخفيّا وللقراءة فقط.. إنّ أيّ متغيّر نعرّفه من هذا المرقّم لن يحمل أكثر من قيمة، فكيف يمكن أن نعبّر إذن عن مثل هذه الحالات؟

بسيطة.. أعد تعرّف هذا المرقّم ليصبح كالتالي:

**<Flags()> Enum FileAttribute**

**عادي = 1**

**للقراءة\_فقط = 2**

**خفيّ = 4**

**مضغوط = 8**

**End Enum**

حيث قمنا بخطوتين أساسيّتين:

- استخدمنا السمة <Flags()> في تعريف المرقّم.

- كما رقّمنا التعبيرات بأرقام ثنائيّة (2 0، 2 1، 2 2، 2 3)، وذلك حتّى نضمن أنّ جمع أيّ رقمين أو أكثر ينتج عنه عدد متفرّد غير قابلٍ للتكرار.. بمعنى أنّ الرقم 6 مثلا، لا يمكن أن ينتج إلا من جمع 2 + 4، وبهذا فهو يدلّ على ملفّ للقراءة فقط وخفيّ.

جرّب ما يلي:

**Dim MyFileAttr As FileAttribute**

**MyFileAttr = FileAttribute.للقراءة\_فقط Or FileAttribute.خفيّ**

**MsgBox(MyFileAttr.ToString)**

ستظهر الرسالة تحمل العبارة التالية: "للقراءة\_فقط، خفيّ".

لاحظ أنّنا استخدمنا المعامل Or لدمج أكثر من سمة.. أمّا لو شئت التحقّق من وجود سمة معيّنة، فعليك استخدام المعامل And كالتالي:

**If MyFileAttr And FileAttribute.للقراءة\_فقط Then ……**

ولمزيدٍ من التفاصيل، اقرأ موضوع الأقنعة Masks في ملحق رقم 2.

**بعض وسائل خليّة المرقّم Enum Class:**

تمنحك خليّة المرقّم ـ التي يرثها أيّ مرقّم تعرّفه ـ بعض الوسائل، سنذكر هنا أهمّها:

**اقرإ الاسم GetName:**

استخدم هذه الوسيلة لمعرفة الاسم الذي يمثّل رقما معيّنا في أحد المرقّمات.. فمثلا، الجملة التالية تعرض رسالة "للقراة\_فقط":

**MsgBox([Enum].GetName(GetType(FileAttribute), 2))**

لاحظ وضعنا لاسم الخليّة Enum بين قوسين مضلّعين، وذلك لأنّ هذه كلمة أساسيّة ولن يسمح لك VB باستخدامها لتشير لاسم الخليّة، إلا بوضعها بين هذين القوسين.. إنّ بإمكانك أن تفعل هذا مع باقي الخلايا الأساسيّة، مثل [Integer]، و [String].... إلخ.

لاحظ كذلك أنّ المعامل الأوّل للوسيلة لا يقبل اسم المرقّم مباشرة، بل يقبل نوعه.. ويمكن الحصول على هذا النوع باستخدام الدالة GetType.

**اقرإ الأسماء GetName:**

استخدم هذه الوسيلة للحصول على مصفوفة بها كلّ الأسماء التي يعرّفها مرقّم معيّن.. انظر للمثال التالي:

**Dim N() As String = ([Enum].GetNames(GetType(FileAttribute)))**

**اقرإ القيم GetValues:**

مماثلةٌ للوسيلةِ السابقة، إلا إنّها تعيد مصفوفة تحتوي على القيم المعرّفة في المرقّم.

**إنّها معرّفة IsDefined:**

تعيد هذه الوسيلة True، إذا كانت القيمة المرسلة في المعامل الثاني معرّفة في المرقّم المرسل في المعامل الأوّل.

**السجلات Structures:**

كثيرا ما ستحتاج للتعامل مع مجموعات من المتغيرات مختلفة الأنواع، لتخزين سجلّ واحد من البيانات، كأن تريد تخزين سجلات عن الطلبة، كل سجل يحتوي على اسم الطالب ورقمه وفصله... إلخ.

ويمكنك أن تُعرّف مثل هذا السجلّ Structure كالتالي:

**Structure Student**

**Dim Name As String**

**Dim ID As Integer**

**Dim StudentClass As Integer**

**End Structure**

هذا التعريف لا يمكن أن يكونَ داخل إيّ إجراء أو دالة، وإنما على مستوى النموذج أو القالب.

**ملحوظة هامّة:**

في أيّ موضع في اللغة، يكون المتغيّر المعرّف باستخدام كلمة Dim موضعيّا Local للمقطع الذي يوجد به، فكأنّ كلمة Dim مناظرة لكلمة Private.. إلا في مقطع السجلّ، حيث تكون كلمة Dim مناظرة لكلمة Public، وبذلك يكون المتغيّر عامّا قابلا للاستخدام عبر اسم السجلّ في باقي البرنامج.. أمّا لو شئت أن تجعل المتغيّر خاصّا، لا يمكن استخدامه إلا داخل مقطع السجلّ فقط، ولا يظهر اسمه في قائمة أعضاء السجلّ، فعرّفه باستخدام كلمة Private.

الآن قد صار لديك نوع جديد من أنواع المتغيّرات، يمكنك استخدامه في برنامجك، كالتالي:

**Dim Ahmad As Student**

**Ahmad.Name= "أحمد شاكر"**

**Ahmad.StudentClass = 5**

**Ahmad.ID = 40**

ستلاحظ أن قائمة الأعضاء ستظهر لك بعد كتابة نقطة "." بعد اسم المتغير Ahmad، تحتوي على العناصر التي عرّفتها في السجلّ، بالإضافة لبعض الدوال الجاهزة التي يمنحها لك VB، مثل "يساوي" Equals، و"اقرإ النوع" GetType و "حوّل إلى نصّ" ToString.. وسيكون من المفيد هنا أن نتعرّف على الدالة "يساوي" Equals.

إنّ هذه الدالة موجودة في كلّ أنواع البيانات، ولكنّ أهميتها لا تتّضحُ في الأنواع البسيطة، كالأرقام والنصوص، لأنّك تستطيع التحقّق من تساوي قيمتين، باستخدام علامة التساوي "=" مباشرة.. أمّا في الأنواع المعقّدة كالسجلات، التي تحتوي على العديد من الحقول الداخليّة، فإنّك تحتاج للتأكّد من تساوي كل حقل (متغيّر) في أحد السجلات، مع الحقل الذي يناظره في السجلّ الآخر، ممّا يجعلها عمليّة شاقّة.. هنا تبرز أهمّيّة دالة التساوي Equals، فهي تقوم بمقارنة كل حقول السجلين، فإن تطابق السجلان فهما متساويان، وإن اختلف أحد الحقول عن مناظره، فالسجلان غير متساويين.. انظر هذا المثال:

**Dim S1, S2 As Student**

**{ املأ حقول السجلين ببعض البيانات }**

**If S1.Equals(S2) Then**

**MsgBox "السجلان متماثلان"**

**Else**

**MsgBox "السجلان مختلفان"**

**End If**

وهو البديل السهل، للطريقة التقليديّة العقيمة التالية:

**Dim S1, S2 As Student**

**{ املأ حقول السجلين ببعض البيانات }**

**If S1.Name = S2.Name AND S1.ID = S2.ID AND \_**

**S1.StudentClass = S2. StudentClass Then**

**MsgBox "السجلان متماثلان"**

**Else**

**MsgBox "السجلان مختلفان"**

**End If**

فإذا كان لدينا هنا 3 حقول في سجل الطالب، فيمكن أن يكون لديك في سجل آخر 10 حقول أو 20 حقلا أو أكثر، ممّا سيجعل المقارنة حقلا بحقل عمليّة مملّة ومضيّعة للوقت والجهد.

وكأيّ متغيّر آخر، يمكنك أن تعرّف مصفوفة Array من أيّ سجلّ:

**Dim Students(10) As Student**

**Students(0).Name ="Ahmad"**

**Students(1).ID = 5**

**If Students(3).Equals(Students(4)) Then Exit Sub**

بل يمكن أن يكون أحد حقول السجلّ نفسه مصفوفة، كما يلي:

**Structure Person**

**Dim First As String**

**Dim Last As String**

**Dim Address As String**

**Dim Phone As String**

**Dim EMail(10) As String**

**End Structure**

تلاحظ أننا نحاول أن نجعل الحقل EMail مصفوفة من 11 خانة (من 0 إلى 10)، بحيث يمكنك أن تخزّن 11 عنوانا بريديا لكل شخص.. للأسف لن تنجح هذه المحاولة، فسيظهر لك خط متعرج تحت هذا السطر، ولو حلّقت فوقه بالفأرة، لظهر لك تلميح على الشاشة، يخبرك أن المصفوفة التي تنتمي للسجلّ لا يمكن منحها حجما ابتدائيّا.. لهذا سنعدّل التعريف إلى:

**Structure Person**

**Dim First As String**

**Dim Last As String**

**Dim Address As String**

**Dim Phone As String**

**Dim EMail() As String**

**End Structure**

ولاستخدام هذا الحقل في الكود، إليك هذا المثال:

**Dim APerson As Person**

**ReDim APerson.EMail(10)**

**APerson.EMail(0) = "JDoe@tex.com"**

وطبعا يمكنك أن تعرّف مصفوفة من هذا السجلّ، ثابتة أو مرنة، كالتالي:

**Dim AllPeople(1000) As Person**

**ReDim AllPeople(3).EMail(2)**

**ReDim AllPeople(4).EMail(5)**

**If AllPeople(3).EMail(0) = AllPeople(4).EMail(1) Then Exit Sub**

وإن كانت المشكلة التي ستواجهك هي حاجتك لإعادة تعريف أطوال المصفوفة Email لكلّ عنصر في من عناصر المصفوفة AllPeople!.. لا مشكلة.. استخدم الجملة التكراريّة التالية:

**Dim C As Integer**

**For C = 0 To 1000**

**ReDim AllPeople(C).EMail(10)**

**Next**

وهناك المزيد ممّا يقال عن السجّلات، سنتعرّف عليه في الفصل القادم، عندما نتعرّف على علاقة السجّلات بالدوالّ.

بقي نوع هامّ جدا من أنواع المتغيرات التي يمكنك أن تُعرّفها، وهي الخلايا Classes، ولكنّ موضعها ليس هنا، فهي تحتاج لفصلٍ مستقلّ بذاته.