الفصل الثالث

أساسيات لغة VB .NET

VB .NET PROGRAMMING FUNDAMENTALS

أهداف الفصل:

- التعرف على لغة VB .NET.
- كتابة وحدات برمجية (Modules) بلغة VB .NET.
- استخدام متغيرات VB .NET وأنواع بياناتها.
- كتابة مصطلحات حسابية بلغة VB .NET.
- كتابة أوامر شرطية Decision-Making Statements.
- كتابة أوامر تكرارية (Loops).
- توصيف المصفوفات واستخدامها.

قدم الفصل الأول كلاً من المفاهيم الرئيسة المتعلقة بتطوير نظم المعلومات مستخدمين التقنية التي تعتمد على الكائنات (Object-Oriented) والتعريف بأهم مفردات هذه التقنية. كما قدم الفصل الثاني البيئة المتكاملة لتطوير البرامج (IDE) مستخدماً لغة VB .NET حيث تم التعرف على كيفية كتابة هذه البرامج وتنفيذها. أما في هذا الفصل فسوف يتم التعرف على أساسيات لغة VB .NET مشتملاً على أسلوب أوامر هذه اللغة وتراكيبها.

نفترض في هذا الكتاب أن القارئ لديه فكرة عن أساسيات البرمجة وسوف نلاحظ أن العديد من أوامر لغة VB .NET ستكون مألوفة لدى القارئ مثل أمر (If-Then-Else) والأوامر التكرارية "While" و"do" والتي تتشابه في العديد من لغات البرمجة، هذا بالإضافة إلى تعلم أوامر جديدة تتعلق بلغة VB .NET.

سوف نتعلم خلال هذا الفصل كيفية تعريف وحدات برمجية (Modules) وتوصيف متغيرات مصطلحات حسابية وكتابتها وكتابة كل من الأوامر الشرطية والأوامر التكرارية والتعامل مع المصفوفات، أو بمعنى آخر عند

الانتهاء من الفصل الحالي سوف يكتسب القارئ المهارات الأساسية لتصميم برامج بسيطة وكتابتها بلغة VB .NET حيث يركز هذا الفصل على عرض أساسيات البرمجة بلغة VB .NET، أما الفصل الرابع فسوف يكشف عن مزيد من التفاصيل الخاصة بلغة VB .NET باعتبارها إحدى لغات البرمجة الموجهة للكائنات.

تقديم لغة VB .NET
Introducing VB .NET

مع أن لغة VB .NET لغة حديثة (تم إصدارها في بداية عام ٢٠٠٢م) ولكنها اكتسبت شهرة واسعة وذلك بسبب العديد من المميزات التي تحتويها مثل دعم تقنية الكائنات وسهولة تعلمها واستخدامها. والأهم من ذلك فإنها لغة تدعم تطوير برامج تعمل على شبكات الحاسب. تأتي قوة لغة VB .NET من وجود عدد هائل من الأصناف سابقة التعريف (Predefined classes) والتي بدورها تحتوي على أساليب (Methods) تُستخدم في إنجاز مهام مختلفة بداية من مهمة تنسيق الأرقام إلى إنجاز الاتصال بالشبكات والتعامل مع قواعد البيانات، وسيتم شرح هذه الأصناف وأساليبها في هذا الفصل والفصول القادمة.

إن لغة VB .NET تدعم مفاهيم الكائنات التي تم التعرف عليها خلال الفصل الأول: الصنف Class، والنسخة Instance، والأسلوب Method، والصفة Attribute، والكبسلة Encapsulation، والتوارث Inheritance، وتعدد الأشكال Polymorphism.، وباختيارك أسلوب التطوير بتقنية OO التي تدعمها لغة VB .NET فإن ذلك يسهل عملية تصحيح الأخطاء وصيانة البرامج بعد التطوير. وسوف نناقش في الفصل الرابع مظاهر تقنية OO بواسطة VB .NET بشيء من التفصيل.

كتابة وحدات برنامج VB .NET
Writing a VB .NET Module Definition

تم التعرف في الفصل الثاني على بيئة التطوير المتكاملة للغة VB .NET من خلال كتابة وحدة برمجية بسيطة باسم HelloWorldWideWebModule.vb وتعريف نموذج باسم HelloWorldWideWebForm.vb، وفي هذه الفقرة سوف نقوم بوصف تلك الوحدة البرمجية بشيء من التفصيل، ويوضح الشكل رقم (٣.١) نص الوحدة البرمجية، بينما الشكل رقم (٣.٢) فيوضح مخرجات البرنامج.

يتكون هذا البرنامج من وحدة برمجية والتي تبدأ بكلمة Module وتنتهي بكلمة End Module، ويمكن أيضاً كتابة أوامر لغة VB .NET داخل تعريف النموذج الذي يُستخدم عند إنشاء واجهة الاستخدام الرسومية للنظام (كما سيتم وصفه في الفصل العاشر) أو داخل تعريف الصنف Class الذي يمثل كائنات النظام (كما سيتم شرحه في الفصل السادس).

```
' Chapter 2 Module definition HelloWorldWideWebModule
Module HelloWorldWideWebForm
   Sub Main()
      Console.WriteLine("Hello World Wide Web")
   End Sub
End Module
```

الشكل رقم (٣,١). شفرة البرنامج HelloWorldWideWebModule.vb.

```
Hello World Wide Web
```

الشكل رقم (٣,٢). مخرجات البرنامج HelloWorldWideWebModule.

تتكون أوامر لغة VB .NET من كلمات أساسية (Keywords) مثل Module و Sub و End Sub ومعرفات (Identifiers) مثل HelloWorldWideWeb، والكلمات الأساسية لها مدلول خاص في لغة VB .NET ويوضح الجدول رقم (٣,١) الكلمات الأساسية التي يتم استخدامها في هذا الكتاب.

الجدول رقم (٣,١). بعض الكلمات الأساسية.

And	As	Boolean	ByRef
Byte	ByVal	Case	Catch
Char	Const	Decimal	Default
Delegate	Dim	Do	Double
Each	Else	ElseIf	End
False	Finally	For	Friend
Function	Handles	If	Implements
Imports	In	Inherits	Integer
Interface	Is	Long	Loop
Me	Mod	Module	MustInherit
MustOverride	MyBase	MyClass	Namespace
New	Next	Not	Nothing
NotInheritable	NotOverridable	Object	On
Option	Optional	Or	Overloads
Overridable	Overrides	Private	Property
Protected	Public	Return	Select
Set	Shadows	Shared	Short
Single	Static	Step	Structure
Sub	Then	Throw	To
True	Try	Until	When
While	With	WithEvents	Xor

تمثل معرفات لغة VB .NET أسماء تسند بواسطة المبرمج للأشياء مثل اسم الوحدة البرمجية، واسم الإجراء، واسم المتغير، وهكذا. وتوجد مجموعة من القواعد البسيطة التي يجب أن تتبع عند تسمية المعرفات والتي يمكن تلخيصها كما يلي:

- يمكن أن يحتوي المعرف على أي عدد من الحروف والأرقام.
- يمكن أن يشتمل المعرف على أي حرف أو رقم عدا المسافات.
- يجب أن يبدأ المعرف بحرف.

لغة VB .NET لا تفرق بين حالتي الحروف (Small or Capital)، وبهذا فإن كلمة Module تكافئ كلمة module. أيضاً وبالرغم من أن ترجمة VB .NET لا يتطلب تنسيق شفرة البرنامج ولكن البرمجة الجيدة تشجع على ذلك كما هو واضح في أمثلة هذا الكتاب. وقد تعلمت في الفصل الثاني كيفية ضغط الاختيار Pretty Listing والذي يجعل محرر لغة VB .NET يقوم بتنسيق شفرة البرنامج وذلك بتحويل أوائل حروف الكلمات الأساسية إلى الحالة Capital والتنسيق الآلي لبدايات الأوامر، ولاحظ أن البرنامج HelloWorldWideWeb يحتوي على ستة أسطر حيث يمثل أول سطر تعليق Comment والذي يهمل بواسطة الكمبيوتر عند تنفيذ البرنامج حيث يتم استخدم التعليقات لإضافة شرح لشفرة البرنامج يمكن الرجوع إليه لتذكر كيفية عمل البرنامج ويضاف التعليق في لغة VB .NET باستخدام علامة التنصيص الفردية (') والذي يطلق عليه Single quote وما يتم كتابته خلف هذا الرمز يمثل تعليقاً.

```
' Chapter 2 Module definition HelloWorldWideWebModule
```

والسطر الثاني يمثل رأس الوحدة البرمجية والذي يبدأ بالكلمة الأساسية Module متبوعاً باسم الوحدة البرمجية HelloWorldWideWeb.

```
Module HelloWorldWideWeb
```

بينما تمثل شفرة البرنامج من السطر الثالث إلى السطر الخامس الإجراء Main وتبدأ الإجراءات (Procedures) برأس الإجراء ويحتوي رأس الإجراء على الكلمة الأساسية Sub متبوعاً باسم الإجراء.

```
Sub Main( )
```

سوف نتعلم الكثير عن الإجراءات لاحقاً في هذا الكتاب، ولكن بشكل عام يتكون الإجراء من عدة أوامر برمجية Statements التي تنفذ عند استدعائها وتنقسم الإجراءات في لغة VB .NET إلى دوال Function وإجراءات فرعية Sub والفرق بينهما أن الدالة تعيد قيمة بعد الانتهاء من تنفيذ أوامرها، أما الإجراء فلا يعيد قيمة.

وإذا احتوى الملف البرمجي على الإجراء Main فعند تحميل هذا الملف في الذاكرة فإن هذا الإجراء يتم استدعاءه مباشرة وتنفيذ أوامره وهذا يعني أن ملف البرنامج HelloWorldWideWeb عند تحميله في الذاكرة سيبدأ تنفيذ الإجراء Main الذي يحتوي على أمر واحد وهو المسؤول عن عرض رسالة ترحيبية.

```
Console.WriteLine("Hello World Wide Web")
```

إن هذا الأمر يشبه العديد من أوامر لغة VB .NET والذي يستدعي إجراء Method للقيام بالمهمة وتنفيذ أوامره (عرض الرسالة). ويوفر الصنف Console العديد من الإجراءات (Methods)، وأحد هذه الإجراءات الإجراء WriteLine والمسؤول عن عرض الرسالة التي يتم تمريرها إليه على الشاشة. والرسالة (البيانات المرسلة بين الأقواس) المرسلة إلى الإجراء والتي تحاط بعلامتي التنصيص الثنائية (" ") تسمى Argument.

استخدام متغيرات VB .NET وأنواع البيانات
Using VB .NET Variables and Data Types

المتغير (Variable) هو مكان في الذاكرة (Memory) يحتوي على قيمة. فعلى سبيل المثال يمكنك كتابة الأمر التالي لجمع عددين معاً:

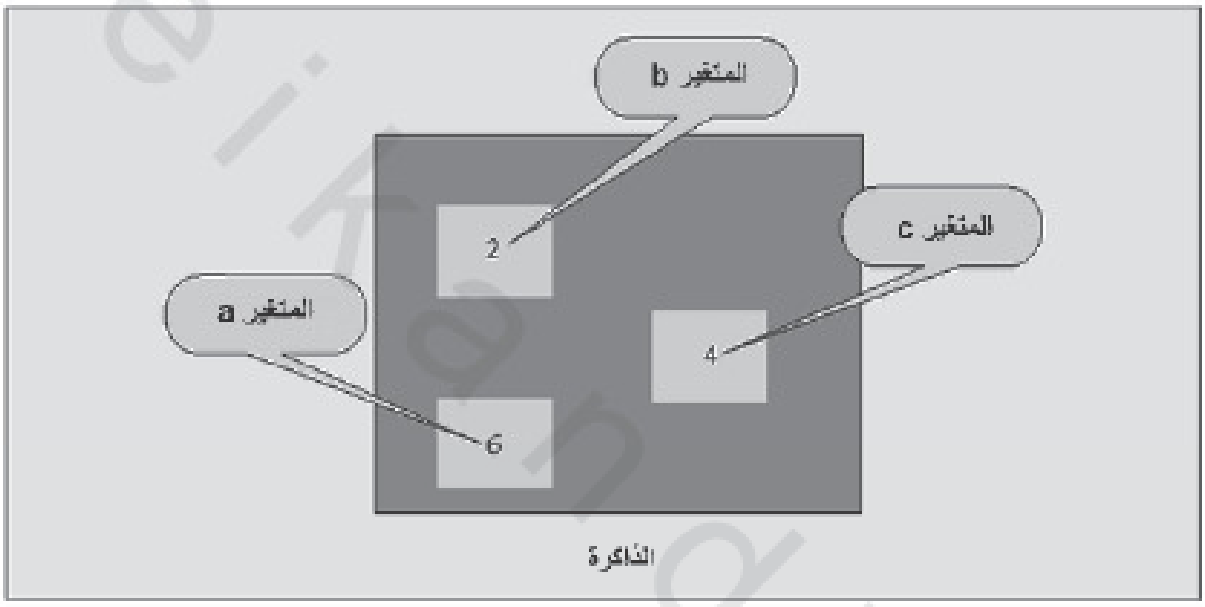
```
a = b + c
```

في هذا المثال فإن كلاً من a وb و c يطلق عليها متغيرات (Variables)، أي أنها أسماء لأماكن في الذاكرة تحتوي على بيانات. فإذا افترضنا أن المتغير b يحتوي على قيمة ٢ والمتغير c يحتوي على قيمة ٤، فبعد تنفيذ هذا الأمر سيحتوي المتغير a على القيمة ٦. ويوضح الشكل رقم (٣.٣) حالة المتغيرات الثلاث بعد تنفيذ هذا الأمر.

ولكي تستخدم متغيراً يجب عليك توصيفه أولاً حيث يحدد لكل متغير اسمه ونوع بياناته (Data Type) وقيمته:

- نوع البيانات: يوضح نوع البيانات ما هو نوع القيم التي يمكن أن يحتويها متغير معين (هل هي أرقام صحيحة أو أرقام كسرية)، وتعتبر لغة VB .NET لغة صارمة تجاه استخدام نوع البيانات، بمعنى أنك إذا قمت بتعريف متغير من النوع الرقمي الصحيح (Integer) فلا يمكنك تخزين رقم كسري بداخله. وسيتم شرح أنواع البيانات بالتفصيل في الفقرة القادمة.

- الاسم: يمثل اسم المتغير معرفاً لهذا المتغير لكي يستخدم للإشارة إليه عند التعامل معه.
- القيمة: كما ذكرنا سابقاً أن المتغير ما هو إلا مكان في الذاكرة يحتوي على قيمة، وإذا تم توصيف متغير بدون إسناد قيمة له فإن لغة VB .NET ستسند قيمة افتراضية له بناء على نوع بياناته حيث تأخذ المتغيرات الرقمية قيمة أولية صفر، بينما المتغير الحرفي (Characters) فيأخذ القيمة الأولية Nothing (هي إحدى الكلمات الأساسية التي تعني لا قيمة)، والمتغير المنطقي (Boolean) سيأخذ القيمة الأولية False.



الشكل رقم (٣,٣). متغيرات تحتوي على بيانات.

توصيف المتغيرات وإعطاء قيمة أولية لها
Declaring and Initializing Variables

قبل توصيف المتغير لابد من تحديد نوع البيانات المخصص لهذا المتغير حيث تحتوي لغة VB .NET على تسع أنواع بيانات أولية (Primitive) كما هو واضح في الجدول رقم (٣,٢). وتم إطلاق اسم أولي على هذه الأنواع لتميزها عن أنواع البيانات المعقدة مثل استخدام الأصناف والتي سيتم مناقشتها في الفصل الرابع.

سيتم استخدام أول أربعة أنواع وهي: Byte و Short و Integer و Long لتعريف متغيرات رقمية صحيحة، بينما الأنواع الثلاثة Single و Double و Decimal فستستخدم لتعريف متغيرات رقمية كسرية، أما النوع Boolean فيستخدم لاحتواء قيمة من قيمتين True أو False.

الجدول رقم (٣,٢). أنواع البيانات الأولية.

	Type	Range of Values	Size
Numeric with no decimals	1. Byte 2. Short 3. Integer 4. Long	0 to 255 -32,768 to 32,767 -2,147,483,648 to 2,147,483,647 ±9,223,372,036,854,775,807	8 bits 16 bits 32 bits 64 bits
Numeric with decimals	5. Single 6. Double 7. Decimal	±1.5E-45 to ±3.4E+38 ±5.0E-324 to ±1.7E+308 1.0E-28 to 7.9E+28	32 bits 64 bits 128 bits
Other	8. Boolean 9. Char	True or False any Unicode character	16 bits 16 bits

يستخدم النوع Character لتعريف متغير يمكن أن يحتوي على حرف واحد فقط، ولتعريف متغير يحتوي على أكثر من حرف (نص) يستخدم الصنف String والذي سيعرض سريعاً في الفقرة التالية وبالتفصيل في الفصل الرابع. سوف نركز في الأمثلة القادمة على الأنواع Integer و Double و Boolean، ولكي يتم توصيف متغير لابد من استخدام الكلمة الأساسية Dim متبوعة باسم المتغير ثم الكلمة As ثم نوع البيانات.

```
' declare variables
Dim i As Integer
Dim d As Double
Dim b As Boolean
```

بعد ذلك يمكنك إسناد قيمة لكل متغير، والأوامر القادمة تُسند القيمة ١ للمتغير i والقيمة ٢,٥ للمتغير d والقيمة True للمتغير b حيث يطلق على المعامل يساوي (=) اسم Assignment Operator أي معامل الإسناد والذي يستخدم لإسناد القيمة التي تقع على يمينه إلى المتغير الذي يقع على يساره.

```
' populate the variables
i = 1
d = 2.5
b = True
```

يمكنك أيضاً توصيف المتغيرات وإسناد قيم أولية لها في أمر واحد، فعلى سبيل المثال تصف الأوامر القادمة المتغيرات كما كان سابقاً ثم تسند قيم أولية إليها:

```
' declare variables
Dim i As Integer = 1
Dim d As Double = 2.5
Dim b As Boolean = True
```

يمكنك أيضاً توصيف أكثر من متغير له نفس نوع البيانات في أمر واحد، فعلى سبيل المثال المتغيرات x و y و z يمكن توصيفه من النوع Integer كما يلي :

```
Dim x, y, z As Integer
```

وبشكل عام تستخدم المتغيرات لتخزين بيانات ويمكنك إسناد قيم أولية عند التوصيف أو إسنادها لاحقاً مستخدماً المعامل =.

تغيير أنواع البيانات
Changing Data Types

من خلال الجدول رقم (٣.٢) يمكن ملاحظة أن سعة أنواع البيانات تختلف، فعلى سبيل المثال المتغير من النوع Byte يستطيع أن يحتوي إلى القيمة ٢٥٥، أما المتغير من النوع Integer يستطيع أن يحتوي على ٢.١ بليون. وبالمثل فالمتغير من النوع Single يستطيع أن يحتوي على 3.4E+38، ولكن المتغير من النوع Double يستطيع أن يحتوي 1.7E+308.

افترض أنك عرفت المتغيرات i و d وأسندت لها قيم كما يلي :

```
Dim i As Integer = 123
Dim d As Double
D = i
```

يعرف الأمر الأول المتغير i ثم يسند له القيمة ١٢٣ ويعرف الأمر الثاني المتغير d ثم يسند الأمر الثالث محتويات المتغير i (١٢٣) إلى المتغير d، وبعد الانتهاء من تنفيذ الأوامر سيحتوي المتغير d على القيمة ١٢٣.

ولكن لاحظ الفرق عند تنفيذ الأوامر التالية :

```
Dim i As Integer
Dim d As Double = 456.789
i = d
```

يعرف الأمر الأول المتغير i من النوع Integer ويعرف الأمر الثاني المتغير d من النوع Double ويسند له القيمة ٤٥٦.٧٨٩ ثم يسند الأمر الثالث قيمة المتغير d داخل المتغير i، وبما أن المتغير i من النوع Integer فهذا يعني أنه يحتوي على أرقام صحيحة فقط، ولهذا فسوف يأخذ فقط الجزء الصحيح من قيمة المتغير d (٤٥٦) ويهمل الجزء العشري، وعلى هذا فإنه عند إسناد الأرقام غير الصحيحة إلى متغيرات صحيحة سيتم فقد الجزء العشري.

ولحل هذه المشكلة تخصص لغة VB .NET الخيار Option Strict لمنع حدوث ذلك (الفقد غير المقصود للجزء العشري للرقم). إن خيار Option Strict هو أحد خيارات مترجم اللغة والتي يمكن أن يسند له On أو Off، ولإسناد قيمة لهذا الخيار يمكن استخدام إحدى طريقتين. الأولى كتابة الأمر Option Strict On أو Option Strict Off في بداية ملف البرنامج Module، أو يمكنك الضغط بالزر الأيمن للفأرة على المشروع في نافذة متصفح الحلول لتفتح نافذة خصائص المشروع (Property Pages window) ثم اختر Build أسفل Common Properties ثم اختر On أو Off في القائمة Option Strict.

إذا تم ضبط الاختيار Option Strict على On فإن مترجم اللغة سيعطي خطأ وذلك عند أي أمر ربما يؤدي إلى الفقد غير المقصود للجزء العشري في الأرقام الكسرية. وفي هذه الحالة يمكن منع مترجم اللغة من الاعتراض وذلك بواسطة استدعاء أحد إجراءات الصنف Convert لكي يقوم بالتحويل قبل إسناد القيمة. انظر الأوامر التالية:

```
Dim i As Integer
Dim d As Double = 456.789
i = Convert.ToInt32 (d)
```

يستدعي الأمر الثالث الإجراء ToInt32 من الصنف Convert والذي يستقبل قيمة المتغير d ثم يقربها لأقرب رقم صحيح (٤٥٧) ثم يعيد النتيجة والتي تُسند للمتغير i، ولكن يجب ملاحظة أنك أنت المسؤول الآن عن فقد أي قيمة عشرية لأنه بهذا الأمر منعت المترجم من إظهار رسالة خطأ.

تحتوي لغة VB .NET على أمر آخر من أوامر المترجم وهو الخيار Option Explicit والذي يأخذ القيمة On أو القيمة Off أيضاً، فإذا أسندت القيمة On لهذا الخيار فإن المبرمج يجب أن يعرِّف أي متغير قبل استخدامه، أما إذا أسندت القيمة Off له فيجوز استخدام متغير دون تعريفه، وبذلك سيعرِّفه المترجم بشكل آلي عند استخدامه، ويبدو ذلك جيداً لدى كثير من المبرمجين ولكن المشكلة تكمن إذا أخطأت في كتابة اسم متغير معرف مسبقاً فسوف ينشئ المترجم متغيراً جديداً بهذا الاسم دون إخبارك بذلك، فعلى سبيل المثال إذا عرفت متغير examScore من النوع Integer وبعد ذلك في أوامر لاحقة أخطأت في كتابته، عندئذ ستنشئ لغة VB .NET متغيراً جديداً بالاسم الخطأ في حالة ما إذا كان الأمر Option Explicit يساوي Off.

استخدام الثوابت
Using Constants

في كثير من الأحيان يكون استخدام المتغير الثابت (Constant) مفيداً جداً حيث يُسند لهذا المتغير قيمة عند تعريفه ولا تتغير في بقية البرنامج. ومن أمثلة هذه القيم اسم الشركة رقم الهاتف والتي لا تتغير عادة أو يكون تغيرها نادراً.

ولتعريف متغير ثابت يتم استخدام الكلمة الأساسية Const بدلاً من الكلمة الأساسية Dim كما هو واضح في المثال التالي. ويجب إسناد قيمة للمتغير الثابت أثناء تعريفه وقد تعارف أن يكون اسم المتغير الثابت بحروف Capital فقط وعند تسميته بأكثر من كلمة يفصل بينها بعلامة Underscore (_). وعلى سبيل المثال لتعريف متغير ثابت يحتوي على معدل ضريبة المبيعات ٧,٥٪ نكتب الأمر التالي :

```
Const SALES_TAX_RATE As Double = 7.5;
```

هذا الأمر يعرِّف متغيراً ثابتاً باسم SALES_TAX_RATE من النوع Double مع إسناد القيمة ٧,٥ التي لا يمكن تغييرها لاحقاً في باقي البرنامج.

استخدام متغيرات الإشارة
Using Reference Variables

تحتوي لغة VB .NET على نوعين من المتغيرات وهما: المتغيرات الأولية (Primitive Variables) ومتغيرات الإشارة (Reference Variables)، ويندرج كل ما تقدم من متغيرات حتى الآن تحت نوع المتغيرات الأولية والتي يتم تعريفها من خلال أنواع البيانات الأولية التسعة، وقد سميت المتغيرات بهذا الاسم لأنها تخزن البيانات التي تسند إليها بداخلها.

أما متغيرات الإشارة فيتم تعريفها بواسطة اسم الصنف (Class Name) والذي يعمل كنوع بيانات، ومن ثم سيشير متغير الإشارة إلى كائن معرف من هذا الصنف والذي بدوره يحتوي على البيانات. فعلى سبيل المثال ربما لاحظت أن المتغير من النوع String (مجموعة من الحروف) لا ينتمي إلى المتغيرات الأولية ولكنه متغير إشارة الذي يشير إلى كائن من الصنف String أحد الأصناف المعرفة داخل لغة VB .NET.

سوف تتعرف أيضاً عزيزي القارئ على المصفوفات (آخر هذا الفصل) والتي يمكن التعامل معها من خلال تعريف متغيرات إشارة (Reference Variables)

يمكن تعريف متغير إشارة من الصنف String كما تعرف المتغيرات الأولية وذلك بواسطة كتابة اسم المتغير متبوعاً بنوع البيانات والذي سيكون في هذه الحالة الصنف String، فعلى سبيل المثال يمكن تعريف متغير s من الصنف String كما يلي :

```
Dim s As string
```

عندئذ سيتم إنشاء متغير باسم s لا يحتوي على قيمة ابتدائية والذي أيضاً لم يشاور على كائن من الصنف String حتى الآن. وفي الحقيقة إن هذا المتغير لا يشاور على أي شيء لأنه يحتوي على قيمة Null ويمكنك إسناد قيمة لهذا المتغير مستخدماً نفس أسلوب إسناد القيم للمتغيرات الأولية، حيث يوضح السطر القادم كيفية إسناد التحية "السلام عليكم" إلى المتغير s:

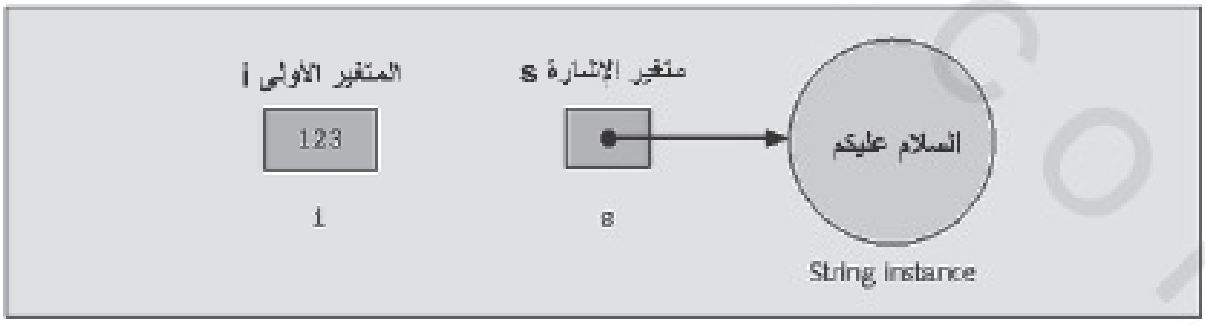
```
s = "السلام عليكم"
```

يمكن أيضاً دمج الخطوتين السابقتين في خطوة واحدة وهي تعريف متغير إشارة من الصنف String مع إسناد قيمة نصية له كما هو واضح في المثال التالي:

```
Dim s As String = "السلام عليكم"
```

في السابق تم تعريف متغير أولي اسمه i من النوع Integer وأسند له القيمة ١٢٣ ولكن يجب ملاحظة أن هذا المتغير s متغير إشارة من الصنف String يشير فقط إلى كائن تم إنشاؤه من الصنف String، بمعنى آخر إن المتغير s يحتوي على عنوان كائن من الصنف String الذي يحتوي على البيانات "السلام عليكم" كما هو واضح في الشكل رقم (٣.٤).

إن معرفة الفرق بين المتغيرات الأولية ومتغيرات الإشارة هام جداً وذلك عند التعامل مع متغيرات الإشارة واستدعاء الإجراءات (Methods) المعرفة داخل أصنافها. وسوف نتعرض في الفصل الرابع لهذه الأفكار بشكل أكثر تفصيلاً وذلك من خلال الصنف String.



الشكل رقم (٣.٤). إنشاء متغيرات أولية وإشارة.

إنشاء ملف برنامج VB .NET لعرض المتغيرات

Creating a VB .NET Module to Demonstrate Variables

يوضح الشكل رقم (٣,٥) الملف البرمجي VariableDemo.vb والذي يعرض توصيف المتغيرات والثوابت.

```
' Chapter 3 VariableDemo Example 2
Option Strict On
Module VariableDemo
   Sub Main()
      ' declare variables
      Dim i As Integer = 123
      Dim d As Double = 456.789
      Dim b As Boolean = True
      Dim s As String = "Hello Again"
      ' a constant must be populated when it is declared
      Const SALES_TAX_RATE As Double = 7.5
      ' display variable contents
      Console.WriteLine("i, an Integer datatype contains " & i)
      Console.WriteLine("d, a Double datatype contains " & d)
      Console.WriteLine("b, a Boolean datatype contains " & b)
      Console.WriteLine("s, a String contains " & s)
      Console.WriteLine("SALES_TAX_RATE, a constant datatype Double
        contains " & SALES_TAX_RATE)

      ' illustrate ToInt32 convert method
      i = Convert.ToInt32(d)
      Console.WriteLine("After invoking Convert.ToInt32(d), i
        contains " & i)
   End Sub
End Module
```

الشكل رقم (٣,٥). شفرة البرنامج VariableDemo.vb.

يبدأ هذا الملف بتعليق ثم بأمر Option Strict On يليه تعريف Module header ثم الإجراء Main.حيث يبدأ الإجراء Main بتوصيف ثلاثة متغيرات أولية ومتغير إشارة من الصنف String ومتغيراً ثابتاً.

```
' declare variables
Dim i As Integer = 123
Dim d As Double = 456.789
Dim b As Boolean = True
Dim s As String = "Hello Again"
' a constant must be populated when it is declared
Const SALES_TAX_RATE As Double = 7.5
```

بعد توصيف المتغيرات وإسناد قيم لها يمكن إضافة الأوامر المسؤولة عن عرض قيمها:

```
' display variable contents
Console.WriteLine("i, an Integer datatype contains " & i)
```

```
Console.WriteLine("d, a Double datatype contains " & d)
Console.WriteLine("b, a Boolean datatype contains " & b)
Console.WriteLine("s, a String contains " & s)
Console.WriteLine("SALES_TAX_RATE, a constant datatype Double contains " &
SALES_TAX_RATE)
```

تستدعي هذه الأوامر الإجراء WriteLine المعرف داخل الصنف Console، وهذا الإجراء يعني إرسال رسالة لتنفيذ هذا الإجراء. ولكتابة أمر لاستدعاء أحد الإجراءات يتم كتابة اسم متغير الإشارة أو اسم الصنف Console متبوعاً بنقطة ثم اسم الإجراء المراد تنفيذه (WriteLine)، وفي الغالب تحتاج أن ترسل بيانات (يطلق عليها Argument) إلى الإجراء المراد تنفيذه داخل أقواس. فعلى سبيل المثال إن أول أمر يقوم بإرسال البيانات التالية:

```
("i, an Integer Data Type contains" & i)
```

تمثل القيمة "i, an Integer Data Type contains" نصاً حرفياً ثابتاً (Literal String) ويمثل المعامل & معامل لصق (Concatenation Operator) والذي يلصق النص الحرفي الثابت مع قيمة المتغير i، حيث يحول معامل اللصق القيمة الرقمية أو قيمة Boolean بشكل تلقائي إلى قيمة نصية قبل استدعاء الإجراء WriteLine، ويوضح الشكل رقم (٣.٦) مخرجات البرنامج VariableDemo.vb

```
i, an Integer datatype contains 123
d, a Double datatype contains 456.789
b, a Boolean datatype contains True
s, a String contains Hello Again
SALES_TAX_RATE, a constant datatype Double contains 7.5
After invoking Convert.ToInt32(d), i contains 457
```

الشكل رقم (٣.٦). مخرجات البرنامج VariableDemo.vb.

تنفيذ العمليات الحسابية مع لغة VB .NET

Computing with VB .NET

تستخدم لغة VB .NET المعاملات الحسابية المشهورة (+ ، − ، / ، x) للجمع والطرح والقسمة والضرب مثلها مثل معظم اللغات، وتستخدم أيضاً الأقواس لتغيير أولوية تنفيذ أجزاء التعبير الرياضي. كما تستخدم اللغة

المعامل Mod لحساب باقي القسمة والذي يطلق عليه Remainder Operator أو Modulus Operator، وتُستخدم أيضاً اللغة المعامل \ الذي يطلق عليه معامل القسمة الصحيح Integer Division Operator للحصول على نتيجة صحيحة. أخيراً يُستخدم المعامل ^ لحساب الأس، ويسرد الجدول رقم (٣,٣) جميع هذه المعاملات

الجدول رقم (٣,٣). المعاملات الحسابية.

Operator	Description	Example	Result
+	addition	11 + 2	13
–	subtraction	11 – 2	9
*	multiplication	11 * 2	22
/	division	11 / 2	6
\	integer division	11 \ 2	5
Mod	remainder	11 Mod 2	1
^	exponentiation	4 ^ 2	16

لاحظ أن تحويل أنواع البيانات يجب أن يكون صحيحاً لكي يعمل معامل القسمة بشكل سليم وتحتوي، أيضاً لغة VB .NET بالإضافة إلى هذه المعاملات على الصنف Math والذي يتضمن على الإجراءات التي تُستخدم لحساب الأس والتقريب والعديد من العمليات الحسابية الأخرى، ويشمل الجدول رقم (٣,٤) على بعض هذه الإجراءات.

الجدول رقم (٣,٤). بعض إجراءات الصنف Math.

Method	Description
Pow(x,y)	Returns the value of x raised to the power of y
Round(x, n)	Returns x rounded to n decimals
Sqrt(x)	Returns the square root of x

ولاستدعاء أي من هذه الإجراءات، أكتب اسم الصنف Math متبوعاً بنقطة ثم اسم الإجراء (Method Name) ثم أي بيانات مطلوب إرسالها (Argument) وسوف يرسل الصنف Math نتيجة العمليات الحسابية بعد تنفيذ الإجراء، فعلى سبيل المثال الأمر التالي يستخدم لحساب الجذر التربيعي للرقم ١٦:

```
Dim answer As Double = Math.Sqrt(16)
```

فعند تنفيذ هذا الأمر ستحدث أربعة أشياء :

١- سيتم إنشاء متغير أولي اسمه Answer من النوع Double.

٢- سيتم استدعاء الإجراء Sqrt من الصنف Math الذي يستقبل الرقم ١٦.

٣- سيحسب الإجراء Sqrt قيمة الجذر التربيعي ويعيده.

٤- يتم إسناد القيمة العائدة إلى المتغير Answer.

يعرض البرنامج ComputationDemo.vb استخدام المعاملات الحسابية واثنان من إجراءات الصنف Math كما هو واضح في الشكل رقم (٣.٧).

يحتوي هذا البرنامج على الإجراء Main() مثله مثل البرامج السابقة والذي يتم تنفيذه بشكل تلقائي عند تشغيل البرنامج. وتبدأ شفرة هذا الإجراء بتوصيف ثلاث متغيرات a و b و c من النوع Integer ومتغير آخر d من النوع Double، ثم بعد ذلك يتم إسناد قيم لها. وبعد ذلك يتم استدعاء الإجراء WriteLine والمسؤول عن عرض نتائج حساب التعبيرات الحسابية بين المتغير a والمتغير b.

```
Console.WriteLine("a + b = " & (a + b))
Console.WriteLine("a - b = " & (a - b))
Console.WriteLine("a * b = " & (a * b))
```

إن شكل المعطيات في هذه الأوامر تختلف شيئاً ما عن ما تم عرضه سابقاً لأنها تحتوي على مصطلحات حسابية والتي يجب أن تحسب أولاً. فعلى سبيل المثال أول أمر يستقبل البيانات ("a + b = " & (a + b)) حيث يتم حساب التعبير (a + b) أولاً ثم لصق النتيجة على النص الثابت "a + b = " ثم تمرير النص الناتج إلى الإجراء WriteLine لعرضه على الشاشة.

يوضح الأمر التالي استخدام معامل باقي القسمة Mod حيث يقسم هذا الأمر محتويات المتغير a على محتويات المتغير b، ثم استنتاج باقي القسمة (١١ تقسيم ٢ يساوي ٥ والباقي ١):

```
Console.WriteLine("Remainder a Mod b = " & (a Mod b))
```

أما الأمر التالي فيوضح كيفية تقسيم محتويات المتغير a على محتويات المتغير b، ولكن الآن الناتج سيكون رقماً كسرياً وهو ٥.٥، لذلك لابد من استدعاء الإجراء Convert.ToInt32 والمسؤول عن تحويل الناتج إلى رقم صحيح قبل إسناده إلى المتغير c حيث نلاحظ أن الناتج تم تقريبه إلى ٦ :

```
' decimal results require Convert.ToInt32
c = Convert.ToInt32(a / b)
Console.WriteLine("Integer division a / b = " & c)
```

```
' Chapter 3 ComputationDemo Example 3

Option Strict On

Module ComputationDemo

    Sub Main()

        ' declare variables
        Dim a, b, c As Integer
        Dim d As Double

        ' populate variables
        a = 11
        b = 2
        c = 4

        ' add, subtract, multiply and display result
        Console.WriteLine("a + b = " & (a + b))
        Console.WriteLine("a - b = " & (a - b))
        Console.WriteLine("a * b = " & (a * b))

        ' Integer division
        Console.WriteLine("Remainder a Mod b = " & (a Mod b))
        c = Convert.ToInt32(a / b) ' decimal results require
            Convert.ToInt32
        Console.WriteLine("Division a / b = " & c)
        c = a \ b
        Console.WriteLine("Integer division a \ b = " & c)
        c = 4    ' repopulate c

        ' Double division
        d = a / b
        Console.WriteLine("Double division a / b = " & d)
        d = 456.789     ' repopulate d

        ' illustrate some Math class methods
        Console.WriteLine("c to b power = " & Math.Pow(c, b))
        Console.WriteLine("square root of c = " & Math.Sqrt(c))
        Console.WriteLine("d rounded is = " & Math.Round(d, 2))

    End Sub

End Module
```

الشكل رقم (٣,٧). شفرة البرنامج ComputationDemo.vb.

يوضح الأمر التالي معامل القسمة الصحيح حيث يتم تقسيم a على b تقسيماً صحيحاً ليكون الناتج ٥ :

```
c = a \ b
Console.WriteLine("Integer division a \ b = " & c)
```

يوضح الأمر التالي القسمة العشرية وذلك من خلال قسمة المتغير a على المتغير b وإسناد نتيجة القسمة للمتغير d وهو ٥.٥ والذي تم تعريفه من النوع Double، وهذا يعني أنه لا يوجد خوف من فقد القيم الكسرية ولا نحتاج إلى استدعاء إجراء الصنف Convert :

```
' Double division
d = a / b
Console.WriteLine("Double division a / b = " & d)
```

أما آخر ثلاثة أوامر في البرنامج فتوضح كيفية استخدام بعض إجراءات الصنف Math:

```
' illustrate some Math class methods
Console.WriteLine("c to b power = " & Math.Pow(c, b))
Console.WriteLine("square root of c = " & Math.Sqrt(c))
Console.WriteLine("d rounded is = " & Math.Round(d, 2))
```

يستدعي الأمر الأول الإجراء Pow والمسؤول عن حساب الأس حيث يرفع محتويات المتغير c (٤) إلى محتويات المتغير b (الذي يحتوي على ٢)، أما الأمر الثاني فيستدعي الإجراء Sqrt والمسؤول عن حساب قيمة الجذر التربيعي للمتغير c، بينما يستدعي الأمر الأخير الإجراء Round والمسؤول عن تقريب محتوى المتغير d (٤٥٦.٧٨٩) لأقرب رقمين عشريين (٤٥٦.٧٩)، ويوضح الشكل رقم (٣.٨) مخرجات هذا البرنامج.

```
a + b = 13
a - b = 9
a * b = 22
Remainder a Mod b = 1
Division a / b = 6
Integer division a \ b = 5
Double division a / b = 5.5
c to b power = 16
square root of c = 2
d rounded is = 456.79
```

الشكل رقم (٣.٨). مخرجات البرنامج ComputationDemo.vb.

كتابة أوامر صنع القرار
Writing Decision-Making Statements

في كثير من التطبيقات يريد المبرمج التأكد من صحة شرط ما لتنفيذ بعض الأوامر أو عدم تنفيذها. فعلى سبيل المثال إن تطبيق معالجة بطاقات الائتمان يحتاج من المبرمج أن يتحقق من عدم تعدي الحد الائتماني للبطاقة. وفي حال تعدي الحد الائتماني يتم رفض عملية الشراء، ولذلك يحتاج المبرمج إلى كتابة أوامر صنع القرار لتقييم هذا الشرط على أن يتم تنفيذ الأوامر أو عدم تنفيذها بناء على هذا التقييم.

تقدم لغة VB .NET أسلوبين لكتابة أوامر صنع القرار. الأول هو أمر If الشرطي، والأمر الثاني هو أمر Select Case. إن أمر if الشرطي مثل جميع أوامر if ببقية اللغات حيث يتم تقييم شرط معين، فإذا كانت نتيجة التقييم إيجابية عندئذ يتم تنفيذ أمر (أو مجموعة من الأوامر)، أما إذا كانت نتيجة الشرط سلبية فعندئذ يكون للمبرمج الخيار في تنفيذ أمر آخر (أو مجموعة من الأوامر). أما أمر Select Case فهي أيضاً مثل كثير من أوامر Select في لغات البرمجة الأخرى ويشبه أمر If متعدد المسارات حيث يتم حساب قيمة مصطلح ومقارنة نتيجة هذا التعبير مع عدة قيم (حيث تمثل كل قيمة حالة)، فإذا تساوى نتيجة التعبير وأحد هذه القيم يتم تنفيذ الأمر (الأوامر) المرافقة لهذه الحالة.

كتابة أمر If الشرطي
Writing If Statements

يتم تقييم تعبير منطقي لمعرفة ما إذا كانت نتيجة صائبة True أم خاطئة False، وعادة ما يتم استخدام معاملات منطقية Logical operators لمقارنة قيمتين (هل هما متساويين أم أن واحداً أكبر من الآخر). ففي مثال معالجة بيانات بطاقات الائتمان لمقارنة هل رصيد الائتمان الحالي (Balance) تعدى الحد الائتماني للبطاقة (Credit Limit) حيث يمكن كتابة التعبير المنطقي كالتالي: creditCardBalance >= creditLimit، وكأمثلة أخرى للتعبيرات examScore > passingScore و studentAge < 21 و studentId = scholarshipId. ويمكن استبدال التعبيرات المنطقية بقيمة منطقية (True أو False)، ويوضح الجدول رقم (٣,٥) المعاملات المنطقية.

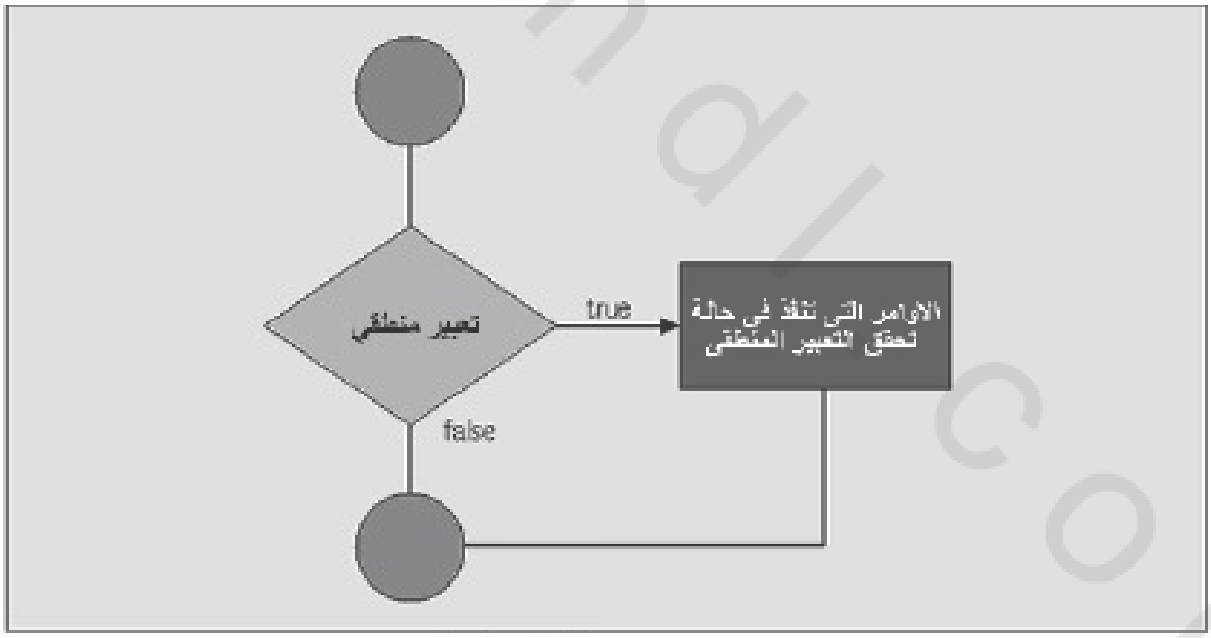
الجدول رقم (٣,٥). المعاملات المنطقية.

Operator	Description
=	equal to
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to
<>	not equal to

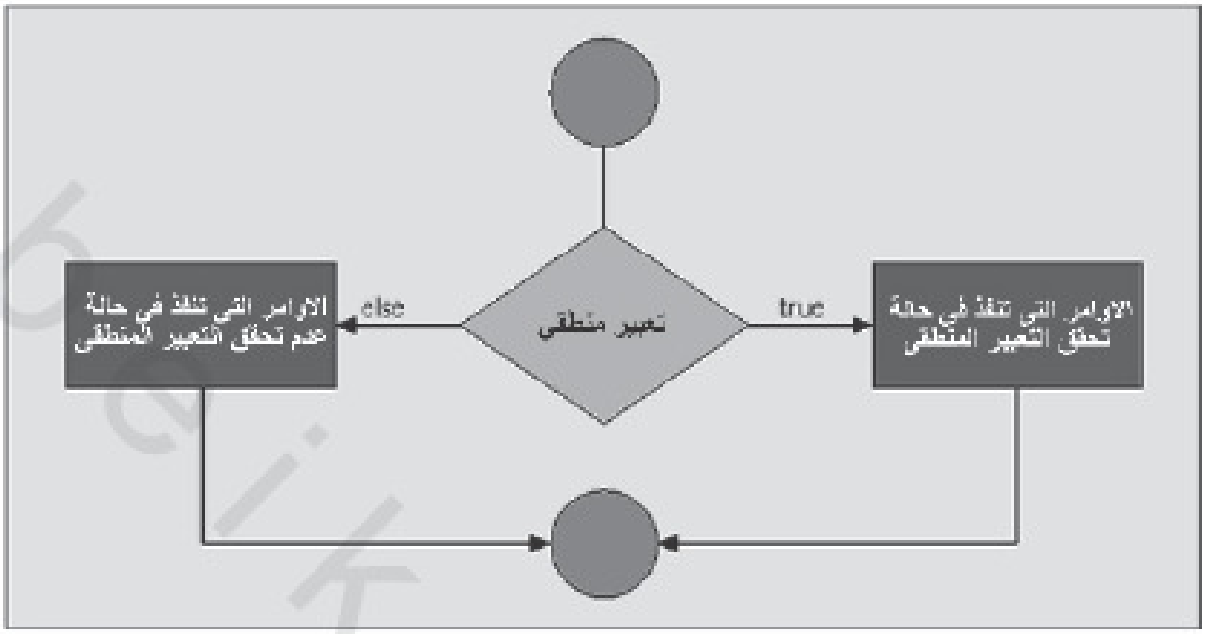
يأخذ أمر if الشرطي شكلاً من صورتين: الصورة الأولى (يطلق عليها أمر if البسيط) تقوم بتقييم التعبير ثم تنفيذ أمر (أو مجموعة من الأوامر) إذا كانت نتيجة التعبير إيجابية، أما الصورة الثانية فهي أمر If-Else التي تقوم بتقييم تعبير منطقي ثم تنفيذ أمر (أو مجموعة من الأوامر) إذا كانت نتيجة التعبير ايجابية أو تنفيذ أمر آخر أو أكثر إذا كانت نتيجة التعبير سلبية. ويوضح الشكل رقم (٣.٩) والشكل رقم (٣.١٠) التصور المنطقي لهاتين الصورتين.

يوضح الشكل رقم (٣.١١) الصيغة العامة لأمر if البسيط مع ملاحظة أنه يوجد شكلين لهذا الأمر، فإذا أردت أن تنفذ أمراً واحداً في حالة صحة التعبير إذن يُكتب هذا الأمر مباشرة بعد التعبير دون الحاجة إلى End if، أما إذا أردت تنفيذ أكثر من أمر في حالة صحة التعبير يتطلب كتابة هذه الأوامر بعد Then كل أمر في سطر منفصل ثم كتابة End If.

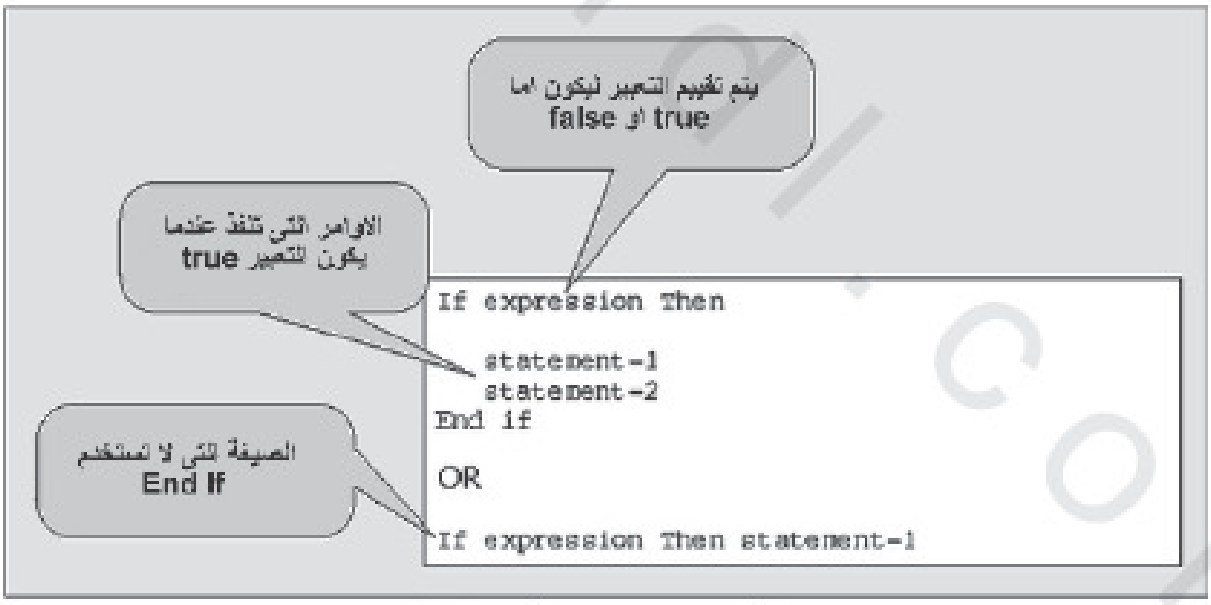
يوضح الشكل رقم (٣.١٢) صورة If-Else حيث يتم تنفيذ أمر أو أكثر في حالة صحة التعبير وتنفيذ أمر آخر أو أكثر في حالة عدم تحقق ذلك. يحتوي البرنامج الموضح في الشكل رقم (٣.١٣) على جميع صور أمر If الشرطي.



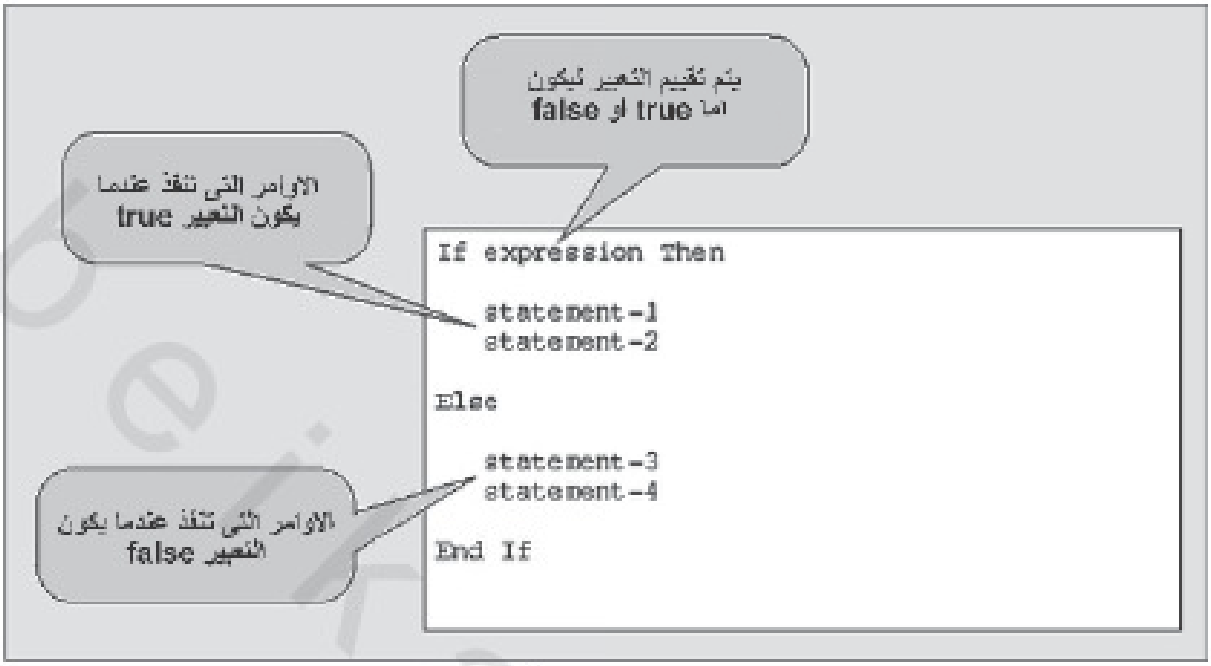
الشكل رقم (٣.٩). منطق أمر If البسيط.



الشكل رقم (٣,١٠). منطق أمر If-Else.



الشكل رقم (٣,١١). صيغة أمر If البسيط.



الشكل رقم (٣,١٢). صيغة أمر If-Else.

يشبه البرنامج الموضح في الشكل رقم (٣,١٣) البرنامج السابق حيث يتم استدعاء الإجراء Main عند تحميل ملف البرنامج ، وعندئذ سيتم تنفيذ أول أمر وهو توصيف المتغير i من النوع Integer وإسناد قيمة ابتدائية له

```
Dim i As Integer = 1
```

بعد ذلك يوجد أمر if البسيط الذي لا ينتهي بكلمة End If والمسؤول عن تقييم الشرط (i < 10) الذي يتوقف على صحته استدعاء الإجراء WriteLine لعرض النص "i < 10":

```
' simple If
If i < 10 Then Console.WriteLine("Simple If without End If: i < 10")
```

تُستخدم هذه الصورة من أمر If إذا كان المراد هو تنفيذ أمر واحد فقط عند تحقق صحة الشرط المصاحب لأمر If، أما إذا أردنا تنفيذ عدة أوامر عند صحة الشرط يتم استخدام أمر If الذي ينتهي End If كما هو واضح في المثال التالي:

```
' use End If
If i > 0 Then
Console.WriteLine("Simple If with End If: i > 0")
End If
```

```
' Chapter 3 IfDemo Example 4
Option Strict On
Module IfDemo
   Sub Main()
      Dim i As Integer = 1
      ' simple If
      If i < 10 Then Console.WriteLine("Simple If without End If:
         i < 10")
      ' use End If
      If i > 0 Then
         Console.WriteLine("Simple If with End If: i > 0")
      End If

      ' If that executes multiple statements
      If i = 1 Then
         Console.WriteLine("i equals 1")
         Console.WriteLine("You can write an If statement")
         Console.WriteLine("that executes multiple statements")
      End If

      ' If - Else
      If i = 2 Then
         Console.WriteLine("i equals 2")
      Else
         Console.WriteLine("If-Else: i does not equal 2")
      End If

      ' compound logical expressions
      If i = 1 Or i = 2 Then
         Console.WriteLine("Compound expression using Or:
           i equals 1 or 2")
      End If
      If i > 0 And i < 3 Then
         Console.WriteLine("Compound expression using And:
           i is > 0 and < 3")
      End If

      ' nested if to replace compound expression
      If i > 0 Then
         If i < 3 Then
            Console.WriteLine("Nested If: i is > 0 and < 3")
         End If
      End If

      ' ElseIf
      If i = 2 Then
         Console.WriteLine("i equals 2")
      ElseIf i <> 2 Then
         Console.WriteLine("Use ElseIf: i does not equal 2")
      End If

      ' Xor: only one expresison is true
      If i = 1 Xor i = 2 Xor i = 3 Then

          Console.WriteLine("Use Xor: i = 1 Xor i = 2 Xor
            i = 3 True")
       End If
       ' Xor: two expresisons are true
       If i = 1 Xor i = 2 Xor i < 3 Then
          Console.WriteLine("Use Xor: i = 1 Xor i = 2 Xor
            i < 3 True")
       Else
          Console.WriteLine("Use Xor: i = 1 Xor i = 2 Xor
            i < 3 False")
       End If

   End Sub
End Module
```

الشكل رقم (٣,١٣). شفرة البرنامج IfDemo.vb.

أما الأمر التالي فيوضح استخدام أمر If-Else والذي يستخدم لتنفيذ الأوامر المصاحبة لكلمة If إذا تحقق صحة الشرط وتنفيذ الأوامر المصاحبة لكلمة Else إذا لم يتحقق الشرط ، كما هو في هذا المثال حيث أن نتيجة الشرط i = 2 سالبة:

```
' If - Else
If i = 2 Then
      Console.WriteLine("i equals 2")
Else
      Console.WriteLine("If-Else: i does not equal 2")
End If
```

يتكون الشرط المركب (Compound Expression) من شرطين أو أكثر بينهما المعامل المنطقي or أو المعامل المنطقي and، فعلى سبيل المثال إن الشرط (i = 2 or i = 1) يُقرأ "هل المتغير i يحتوي على القيمة ٢ أو القيمة ١؟"، ولأن المتغير i يحتوي على القيمة ١ فإن نتيجة هذا الشرط صحيحة. أما الشرط (i < 3 and i > 0) والذي يُقرأ "هل المتغير i يحتوي على قيمة أكبر من ٠ وأقل من ٣؟" ولأن المتغير i يحتوي على ١ فإن نتيجة هذا الشرط أيضاً صحيحة.

```
' compound logical expressions
If i = 1 Or i = 2 Then
      Console.WriteLine("Compound expression using Or: i equals 1 or 2")
End If
If i > 0 And i < 3 Then
      Console.WriteLine("Compound expression using And: i is > 0 and < 3")
End If
```

يوجد شكل لأمر If وهو أمر If المتداخلة (Nested If) وهي عبارة عن أمر if داخل أمر if حيث يمكن غالباً استبدال أمر if المعتمد على الشرط المركب بأمر if المتداخل، فالمثال التالي يكافئ المثال السابق الذي يعتمد على الأمر المركب ويلاحظ بعد الأمر الثاني والثالث عن بداية السطر وهو ما يحسن من قراءة البرنامج.

```
' nested if to replace compound expression
If i > 0 Then
      If i < 2 Then
            Console.WriteLine("Nested If: i is > 0 and < 2")
      End If
End If
```

تقدم لغة VB .NET صورة أخرى هامة لأمر if وهو ElseIf والذي يربط مفهوم الكلمة If بالكلمة Else حيث تستخدم الكلمة ElseIf لتجربة شرط آخر عند عدم صحة الشروط السابقة، فالمثال التالي يختبر أولاً الشرط i = 2 ولأن قيمة المتغير i تساوي ١ فتكون نتيجة الشرط خاطئة، عندئذ يختبر الشرط المصاحب لكلمة ElseIf الذي تكون صائبة ولذلك سينفذ الأمر المصاحب لكلمة ElseIf

```
' ElseIf
If i = 2 Then
      Console.WriteLine("i equals 2")
ElseIf i <> 3 Then
      Console.WriteLine("Use ElseIf: i does not equal 2")
End If
```

ويوضح الشكل رقم (٣,١٤) مخرجات البرنامج.

```
Simple If without End If: i < 10
Simple If with End If: i > 0
i equals 1
You can write an if statement
that executes multiple statements
If-Else: i does not equal 2
Compound expression using Or: i equals 1 or 2
Compound expression using And: i is > 0 and < 3
Nested If: i is > 0 and < 3
Use ElseIf: i does not equal 2
Use Xor: i = 1 Xor i = 2 Xor i = 3 True
Use Xor: i = 1 Xor i = 2 Xor i < 3 False
```

الشكل رقم (٣,١٤). مخرجات البرنامج IfDemo.

كتابة أوامر Select Case
Writing Select Case Statements

تقدم أوامر لغة VB .NET أسلوباً بديلاً لتنفيذ مجموعة من الأوامر بناء على قيمة معينة أطلقت عليها Select Case والتي تشبه مجموعة من أمر If متعددة حيث يستخدم أمر Select Case عندما نريد أن نأخذ قراراً لتنفيذ أوامر مختلفة بناء علي قيم مختلفة.

فعلى سبيل المثال التمرين العملي رقم ٤ الذي تم حله مستخدماً أوامر if يمكن أيضاً حله مستخدماً أمر Select Case حيث يمكنك تعريف المتغير examScore ليحتوي علي درجة الاختبار والمطلوب هو عرض التقدير بناء على هذه الدرجة ، فعندئذ يمكن استخدام أمر Select Case كما يلي :

```
Select Case examScore
Case Is >= 90
      Console.WriteLine("The Grade is A")
Case 80 To 89
      Console.WriteLine("The Grade is B")
```

```
Case 70 To 79
      Console.WriteLine("The Grade is C")
Case 60 To 69
      Console.WriteLine("The Grade is D")
Case Else
      Console.WriteLine("The Grade is F")
End Select
```

إن تركيب أمر Select Case يتكون بداية بالكلمة المحجوزة Select Case متبوعاً باسم متغير (أو تعبير ما) ثم مجموعة من أوامر Case حيث يصاحب كل Case أمراً أو مجموعة من الأوامر التي تنفذ لو تحقق الشرط المصاحب لهذه الحالة مع قيمة المتغير. وفي المثال السابق يتم تقييم قيمة المتغير examScore مع استخدام أوامر Case لتغطية الحالات المختلفة لقيمة المتغير، وعند تحقق أحد الحالات يتم تنفيذ الأوامر المرافقة لهذه الحالة.

يجوز وضع قيمة مع كل Case ومطابقتها مع تعبير Select Case أو كتابة شرط منطقي (Case Is >= 90) أو تعريف مجموعة من القيم (Case 80 To 99) أو قائمة من القيم (Case 10, 15, 20). لاحظ استخدام الكلمات المحجوزة To و Is ولاحظ أيضاً أن أوامر Case Else يتم تنفيذها في حالة عدم تنفيذ أي من الحالات السابقة لها.

إن أمر Select Case جيد جداً حيث يمكن استخدامه لتبسيط شفرة البرنامج وإحلاله بدلاً من عدة أوامر If متداخلة.

كتابة الأوامر التكرارية
Writing Loops

افترض أنك تريد أن تكتب برنامجاً لعرض الأعداد ١ و ٢ و ٣ على الشاشة فيمكن عمل ذلك بكتابة الأوامر التالية:

```
Console.WriteLine(1);
Console.WriteLine(2);
Console.WriteLine(3);
```

ولكن هذا الأسلوب يعتبر مضيعة للوقت وخصوصاً عندما تكون عدد الأعداد كثيرة (مثل عرض الأعداد ١ إلى ١٠٠٠)، أما الأسلوب الأمثل لذلك هو كتابة تكرار (Loop) والذي يعتبر أداة برمجية جيدة لإعادة تنفيذ مجموعة من الأوامر، وحتى تحقق شرطاً معيناً فيمكن إعداد تكرار لجمع عدة قيم أو لعد أشياء أو للوصول لعناصر مصفوفة حيث تقدم لغة VB .NET ثلاثة مجموعات من الكلمات المحجوزة لكتابة التكرارات وهي: For Next و Do Until و Do While والتي سيتم شرحها بالتفصيل في الفقرات التالية.

كتابة التكرار Do While
Writing Do While Loops

توضع الأوامر التالية استخدام التكرار Do While لعرض الأرقام ١ إلى ٣ :

```
' do while loop
' declare and initialize loop counter variable
Dim i As Integer = 1
Do While i <= 3
      Console.WriteLine("do while loop: i = " & i)
      i += 1
Loop
```

يبدأ هذا المثال بتعريف المتغير i مع إسناد قيمة ابتدائية له ، ويطلق على هذا المتغير عداد التكرار (Loop Counter) لأنه يستخدم لعد مرات تنفيذ أوامر التكرار حيث تظهر الكلمات المحجوزة Do While متبوعاً بالشرط i <= 3 ثم يظهر جسم التكرار الذي يتكون من أمرين والذي يستمر تنفيذهما طالما نتيجة الشرط صحيحة، وعندما تصبح نتيجة الشرط سلبية يتوقف التكرار ليبدأ في تنفيذ الأوامر التي تليها.

يقوم الأمر الأول (من أوامر جسم التكرار) بعرض الرسالة النصية "do while loop: i = " متبوعاً بقيمة المتغير i، أما الأمر الثاني فهو مسؤول عن زيادة المتغير بمقدار واحد. لاحظ الأمر المستخدم للزيادة i += 1 والذي يكافئ الأمر i = i + 1.

وفي البداية يسند إلى المتغير i القيمة ١ ثم تنفيذ التكرار Do While حيث يتم تقييم الشرط i <= 3، ولأن المتغير i أقل من ٣ يتم عرض الرسالة ثم زيادة قيمة المتغير i بواحد ليصبح ٢ ثم تقييم الشرط الذي يكون صحيحاً، وبذلك يتم تنفيذ أوامر التكرار مرة أخرى بعرض الرسالة ثم زيادة المتغير i بواحد وهكذا إلى أن يصل المتغير إلى القيمة ٤ التي تجعل الشرط خاطئاً فعندئذ يتوقف تنفيذ أوامر التكرار Do While.

لاحظ عزيزي القارئ مدى سهولة تعديل البرنامج لطباعة الأعداد من ١ إلى ١٠٠٠ وذلك بتغيير شرط التكرار ليصبح i <= 1000. ربما يعد خطئاً أن نكتب تكراراً لا نهائياً (Infinite Loop) وهو التكرار الذي لا يتوقف بسبب عدم تغير حالة الشرط المرافق للتكرار من صواب إلى خطأ، فمثلاً عند إعادة كتابة المثال السابق دون كتابة الأمر المسؤول عن زيادة المتغير i فعندئذ ستظل قيمة المتغير i تساوي ١ وبذلك ستظل قيمة الشرط صواباً كما هو واضح في المثال التالي :

```
' infinite do while loop
Dim i As Integer = 1
Do While i <= 3
      Console.WriteLine("do while loop: i = " & i)
Loop
```

تقدم لغة VB .NET شكلاً آخر للتكرار Do While ويعمل مثله تماماً وهو التكرار While حيث يعرض المثال التالي الأرقام من ١ إلى ٣ مستخدماً التكرار While مثل المثال السابق:

```
' while loop
i = 1   ' re-initialize loop counter variable
While i <= 3
      Console.WriteLine("While loop: i = " & i)
      i += 1
End While
```

كتابة التكرار Do Until
Writing Do Until Loops

رأينا في المثال السابق كيفية كتابة التكرار Do While لعرض الأرقام من ١ إلى ٣ والذي يمكن إعادة كتابته مستخدماً التكرار Do Until كما هو واضح المثال التالي:

```
' do until loop
i = 1   ' re-initialize loop counter variable
Do Until i > 3
      Console.WriteLine("do until loop: i = " & i)
      i += 1
Loop
```

يوجد تشابه بين كل من التكرارين حيث يجب إسناد قيمة ابتدائية لعداد التكرار (المتغير i) قبل التكرار ثم يبدأ أمر التكرار Do While أو Do Until متبوعاً بشرط التكرار تليه الأوامر المراد تكرارها ثم ينتهي التكرار بكلمة Loop، ولكن الفرق الوحيد بين التكرار Do While والتكرار Do Until هو أن الأول يعيد تنفيذ أوامره طالما نتيجة الشرط صواباً، بينما الثاني يعيد تنفيذ أوامره حتى تصبح نتيجة الشرط صواباً.

كتابة تكرارات Post-Test
Writing Post-Test Loops

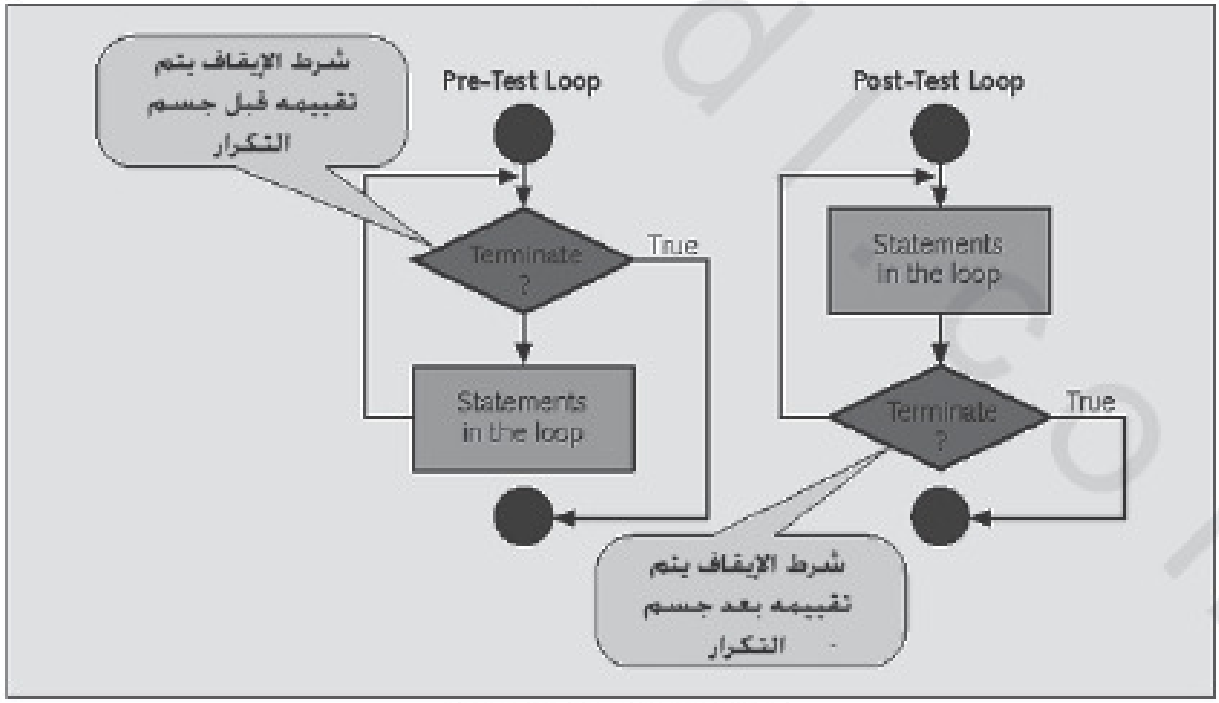
بشكل عام يمكن تصنيف تكرارات لغات البرمجة إلى نوعين وهما: النوع الأول هو Pre-test loop والذي يختبر الشرط ثم يقرر تنفيذ الأوامر أو التوقف عن التنفيذ بناء على قيمة الشرط، أما النوع الثاني فهو Post-test loop وهو عكس النوع الأول حيث ينفذ الأوامر المطلوب تكرارها أولاً ثم يختبر الشرط وبناء على نتيجة الشرط يقرر تنفيذ الأوامر مرة أخرى أو أن يتوقف عن التكرار، ويوضح الشكل رقم (٣,١٥) منطق هذين النوعين من التكرارات فالتكرار من النوع Post-test loop ينفذ أوامره على الأقل مرة واحدة (في حالة خطأ الشرط من المرة الأولى)، أما التكرار من النوع Pre-test loop ربما لا ينفذ أوامره إطلاقاً ويخرج من أول مرة وذلك في حال خطأ الشرط. لذلك يتم

استخدام التكرارات من النوع Post-test loop عندما نريد أن ننفذ الأوامر المراد تكرارها مرة واحدة على الأقل بغض النظر عن نتيجة الشرط، بينما نوع التكرارات Pre-test loop فيستخدم عندما نريد أن ننفذ أوامر التكرار بناء على قيمة الشرط وعندئذ ربما لا ننفذ أوامر التكرار مطلقاً وذلك عندما تكون نتيجة الشرط خطأ في أول مرة.

يمكن استخدام Do While و Do Until في كل من التكرار Pre-test loop والتكرار Post-test loop، أما For و While فيتم استخدامهما في حالة التكرار Pre-test loop فقط.

لكي نتمكن من استخدام Do While في التكرارات من النوع Post-test loop يتم تغيير مكان كلمة While من أعلى إلى أسفل كما هو واضح في المثال التالي الذي يعرض الأرقام من ١ إلى ٣:

```
' post-test do while loop
i = 1 ' re-initialize loop counter variable
Do
      Console.WriteLine("post-test executed do while loop: i = " & i ")
      i += 1
Loop While (i <= 3)
' post-test do until loop
i = 1 ' re-initialize loop counter variable
Do
      Console.WriteLine("post-test executed do while loop: i = " & i ")
      i += 1
Loop While (i > 3)
```



الشكل رقم (٣,١٥). منطق أوامر التكرار.

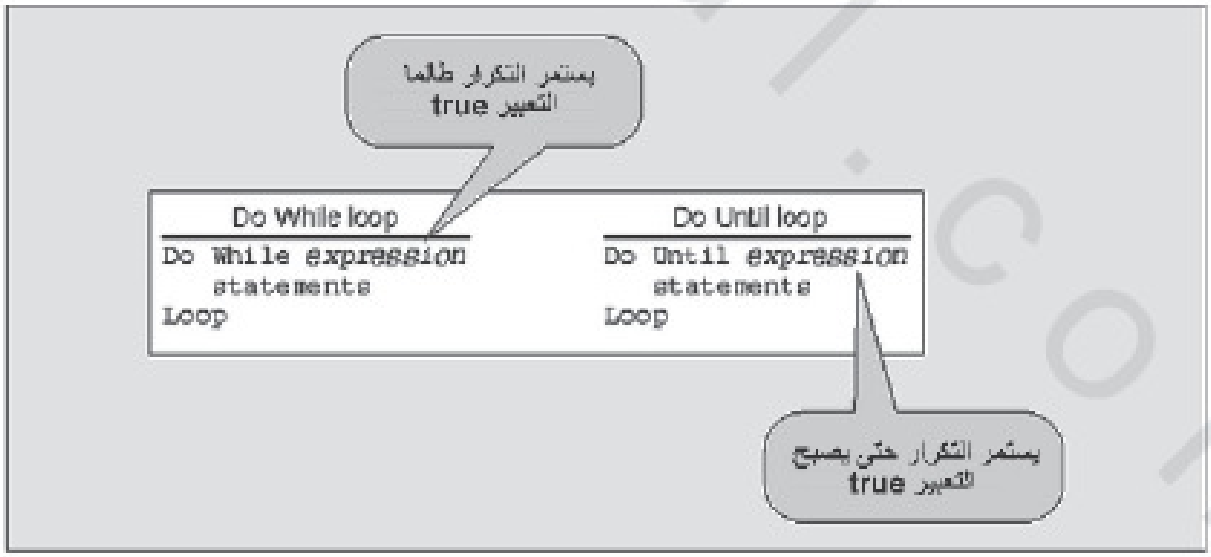
يوضح المثال التالي الفرق بين كل من التكرار Pre-test loop والتكرار Post-test loop:

```
' pre-test & post-test loop compared
i = 1   ' re-initialize loop counter variable
' pre-test evaluates the expression at beginning of loop
Do While i > 3   ' expression initially false
      Console.WriteLine("pre-test executed")
Loop


' post-test evaluates the expression at end of loop
Do
      Console.WriteLine("post-test executed")
Loop While i > 3   ' expression initially false
```

في البداية يسند إلى المتغير i القيمة ١ ثم يأتي Do While في حالة التكرار من النوع Pre-test loop والذي يختبر الشرط أولاً ليجده خطئاً، ولذلك لا تنفذ أوامر هذا التكرار مطلقاً لأن الشرط يختبر أولاً ويتم تنفيذ أوامر التكرار بناء على صحة الشرط.

أما التكرار الثاني فهو من النوع Post-test loop والذي يتم فيه اختبار الشرط في نهاية التكرار ولذلك ستنفذ الأوامر المطلوب تكرارها بغض النظر عن نتيجة الشرط الخاطئة. والمثال السابق يوضح أن كلاً من التكرارين يعملان بشكل مختلف تماماً كما يظهر الشكل رقم (٣.١٦) الفرق بين Do While و Do Until.



الشكل رقم (٣.١٦). بناء أمري التكرارين Do While و Do Until.

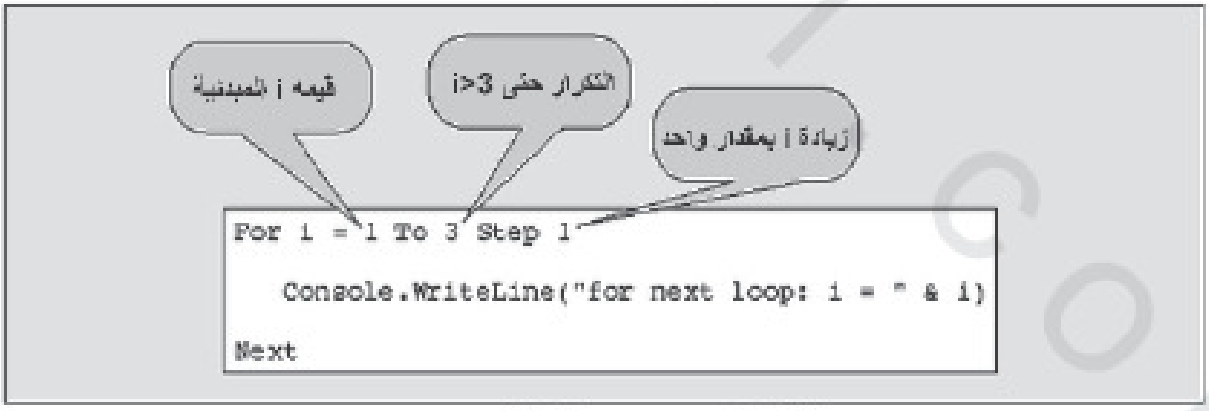
وبإيجاز يتم استخدام التكرار Post-test loop عندما نريد تنفيذ أوامر التكرار بغض النظر عن نتيجة الشرط في المرة الأولى، ويستخدم التكرار Pre-test loop عندما نريد تنفيذ أوامر التكرار بناء على نتيجة الشرط.

كتابة التكرار For Next
Writing For Next Loops

يتيح لك أمر التكرار For Next أن تكتب أمر إسناد قيمة ابتدائية لعداد التكرار (Loop Control) وزيادته (أو نقصانه) في رأس التكرار مما يؤدي إلى اختصار كتابة الأوامر. وكما ذكرنا سابقاً أنها تنتمي إلى نوع التكرار Pre-test loop بمعنى أنها تقوم بتقييم الشرط أولاً وقبل تنفيذ الأوامر المراد تكرارها بناء على نتيجة الشرط. يوضح المثال التالي نفس المثال السابق (طباعة الأرقام من ١ إلى ٣) ولكن مستخدماً التكرار For Next :

```
' for next loop
i = 1   ' re-initialize loop counter variable
For i = 1 To 3 Step 1
      Console.WriteLine("for next loop: i = " & i)
Next
```

كما هو واضح أن التكرار For Next تتطلب عدد خطوات أقل من أوامر التكرار السابقة وذلك لدمج كلاً من زيادة القيمة الابتدائية وإسنادها لعداد التكرار في رأس التكرار مع شرط التكرار، ويوضح الشكل رقم (٣.١٧) ما هي وظيفة كل جزء من التكرار.



الشكل رقم (٣.١٧). صيغة أمر التكرار For Next.

تستخدم الكلمة المحجوزة Step لزيادة/نقصان عداد التكرار وهي كلمة اختيارية أي يمكن حذفها وذلك إذا كان مقدار الزيادة واحد لأنه هو القيمة الافتراضية. وعلى أي حال إذا أردنا زيادة عداد التكرار بقيمة غير واحد

فيتطلب ذلك كتابة الكلمة Step كما هو موضح في المثال التالي والذي تزيد فيه قيمة المتغير i بمقدار ٢. لاحظ أن التكرار سوف يتوقف عندما تصبح قيمة i أكبر من ٣، وهذا يعني أن البرنامج سوف يعرض ١ ثم ٣ فقط:

```
' for next loop
For i = 1 To 3 Step 2
      Console.WriteLine("for next loop: i = " & i)
Next
```

كتابة التكرارات المتداخلة
Writing Nested Loops

في بعض الأحيان نريد أن نكتب تكراراً داخل تكرار والذي نطلق عليه التكرارات المتداخلة والذي يكون مفيداً جداً عند التعامل مع بيانات جدولية (مرتبة في صفوف وأعمدة)، فعندئذ يمكن استخدام التكرار الخارجي للتحرك عبر الأعمدة واستخدام التكرار الداخلي للتعامل مع عناصر كل صف.

يمكن تكوين تكرارات متداخلة من جميع أوامر التكرار المذكورة سابقاً من Do While و Do Until و For Next، ولكن استخدام For Next يقلل من عدد سطور الأوامر. ويوضح المثال التالي استخدام تكرار For Next داخل تكرار For Next حيث يستخدم التكرار الخارجي عداد التكرار m، بينما التكرار الداخلي فيستخدم عداد التكرار n.

```
' nested loop
Dim m, n As Integer
For m = 1 To 2
      For n = 1 To 3
            Console.WriteLine("nested loop: m = " & m & ", n = " & n)
      Next
Next
```

كما هو واضح في المثال أن التكرار الخارجي سيُنفذ مرتين ومع كل مرة سيتم تنفيذ التكرار الداخلي ثلاث مرات. وبمعنى آخر بينما يأخذ العداد m رقم ١ فإن العداد n سيأخذ القيم ١ ثم ٢ ثم ٣، ومع كل تنفيذ لأوامر التكرار الداخلي يتم عرض محتويات المتغيرات على الشاشة حيث يتم استدعاء الإجراء WriteLine ستة مرات (عدد مرات تنفيذ التكرار الداخلي x عدد مرات تنفيذ التكرار الخارجي). لاحظ أن المقطع Step قد تم حذفه حيث أن قيمة الزيادة الافتراضية واحد.

يحتوي الملف LoopDemo.vb على جميع الأمثلة المذكورة سابقاً والموضحة في الشكل رقم (٣.١٨)، ومخرجات هذا البرنامج موضحة في الشكل رقم (٣.١٩).

```
' Chapter 3 LoopDemo Example 5
Option Strict On
Module LoopDemo
   Sub Main()
   ' do while loop
      ' declare and initialize loop counter variable
   Dim i As Integer = 1
   Do While i <= 3
      Console.WriteLine("do while loop: i = " & i)
      i += 1
   Loop

   ' while loop
   i = 1  ' re-initialize loop counter variable
   While i <= 3
      Console.WriteLine("While loop: i = " & i)
      i += 1
   End While

   ' do until loop
   i = 1  ' re-initialize loop counter variable
   Do Until i > 3
      Console.WriteLine("do until loop: i = " & i)
      i += 1
   Loop

   ' pre-test & post-test loop compared
   i = 1  ' re-initialize loop counter variable
   ' pre-test evaluates the expression at start  of loop
   Do While i > 3  ' expression initially false
      Console.WriteLine("pre-test executed")
   Loop
   Do
      Console.WriteLine("post-test executed")
   Loop While i > 3  ' expression initially false

   ' for next loop
   For i = 1 To 3 Step 1
      Console.WriteLine("for next loop: i = " & i)
   Next

   ' nested loop
   Dim m, n As Integer
   For m = 1 To 2
      For n = 1 To 3
         Console.WriteLine("nested loop: m = " & m & ", n = " & n)
      Next
   Next

   End Sub
End Module
```

الشكل رقم (٣,١٨). شفرة البرنامج LoopDemo.vb.

```
do while loop: i = 1
do while loop: i = 2
do while loop: i = 3
While loop: i = 1
While loop: i = 2
While loop: i = 3
do until loop: i = 1
do until loop: i = 2
do until loop: i = 3
posttest executed
for next loop: i = 1
for next loop: i = 2
for next loop: i = 3
nested loop: m = 1, n = 1
nested loop: m = 1, n = 2
nested loop: m = 1, n = 3
nested loop: m = 2, n = 1
nested loop: m = 2, n = 2
nested loop: m = 2, n = 3
```

الشكل رقم (٣,١٩). مخرجات البرنامج LoopDemo.vb.

توصيف المصفوفات ومعاملتها

Declaring and Accessing Arrays

مثل معظم اللغات فإن لغة VB .NET تمكنك من تعريف مصفوفات (Arrays) وذلك لإنشاء مجموعة من المتغيرات من نفس نوع البيانات حيث تتكون المصفوفة من مجموعة عناصر تسلك سلوك المتغير غير أن جميع عناصر المصفوفة لابد أن تكون لها نفس نوع البيانات.

إن عناصر المصفوفة مثل جميع المتغيرات ربما تحتوي على بيانات أولية Primitive Data أو تشير إلى كائنات لصنف ما مثل الصنف String. ويصف الفصل الرابع متغيرات الإشارة بالتفصيل ويحتوي أيضاً على العديد من الأمثلة.

يوجد نوعان من المصفوفات وهما: مصفوفات ذات بعد واحد (One-Dimensional Arrays) ومصفوفات متعددة الأبعاد (Multi-Dimensional Arrays). تتكون المصفوفة ذات البعد الواحد من صف من العناصر، أما المصفوفة ذات البعدين فتتكون من صفوف وأعمدة، والمصفوفة ذات الأبعاد الثلاث فهي مثل المكعب الذي يتكون من صفوف وأعمدة وصفحات. على أي حال فإن لغة VB .NET تمثل المصفوفات ذات الأبعاد المتعددة كمصفوفات داخل مصفوفات، ولذلك أنت غير مقيد بشكل المستطيل لتعريف مصفوفة ذات بعدين أو مكعب لتعريف مصفوفة ذات ثلاث أبعاد.

استخدام المصفوفة ذات البعد الواحد
Using One-Dimensional Arrays

إذا أردت تعريف خمس متغيرات لحفظ درجات اختبار مواد دراسية فلابد من توصيفها كما يلي:

```
Dim testScore1 As Integer = 75
Dim testScore2 As Integer = 80
Dim testScore3 As Integer = 70
Dim testScore4 As Integer = 85
Dim testScore5 As Integer = 90
```

أما البديل الآخر فهو تعريف مصفوفة ذات بعد واحد تحتوي على خمس عناصر كما يلي:

```
' declare an integer array with 5 elements
Dim testScores(4) As Integer
```

يوصف هذا الأمر متغير إشارة اسمه testScore ليشير إلى كائن من النوع Arrays ثم إنشاء مصفوفة (كائن من النوع Arrays) والتي ستحتوي على خمس عناصر من النوع Integer. لاحظ عند توصيف هذه المصفوفة تم استخدام الرقم ٤ لتعريف خمس عناصر وذلك لأن لغة VB .NET تبدأ ترقيم عناصر المصفوفة من صفر، والرقم المستخدم في التعريف يشير إلى فهرس آخر عنصر وليس إلى حجم المصفوفة.

يمكن التعامل مع عناصر المصفوفة بكتابة أسم المصفوفة (المتغير testScore) متبوعاً بفهرس العنصر داخل قوسين. ولذلك يمكن التعامل مع العنصر الأول في المصفوفة مستخدماً testScore(0)، والتعامل مع العنصر الثاني مستخدماً testScore(1) وهكذا (لاحظ أن الفهرس يبدأ بصفر وليس بواحد). ولذلك فإن فهرس العنصر الخامس هو الرقم ٤.

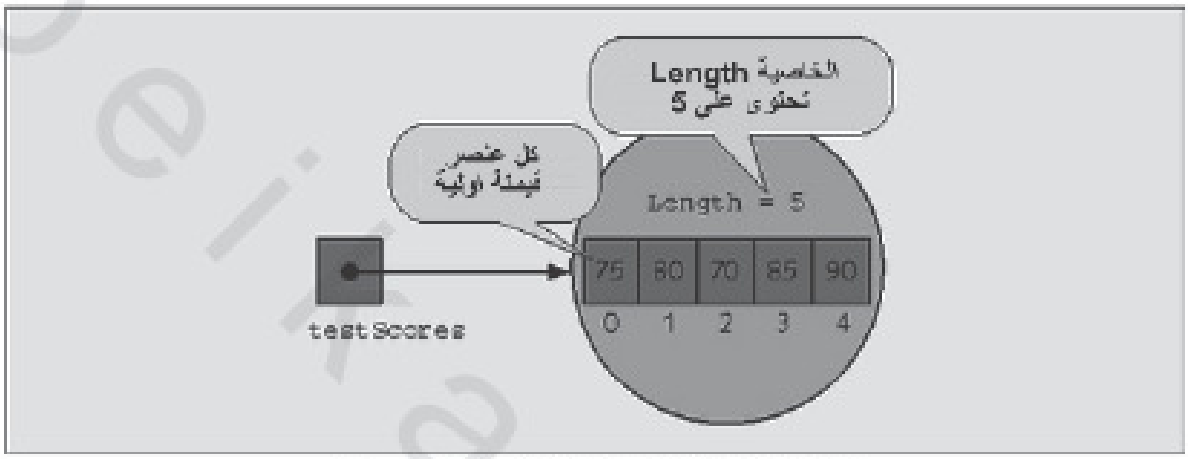
يسند إلى عناصر المصفوفة قيم بنفس الأسلوب السابق المستخدم لإسناد قيم لمتغيرات النوع Integer. ويستخدم المثال التالي لإسناد قيمة ٧٥ للعنصر الأول و ٨٠ للعنصر الثاني وهكذا.

```
testScores(0) = 75
testScores(1) = 80
testScores(2) = 70
testScores(3) = 85
testScores(4) = 90
```

ويمكن تعريف المصفوفة وإسناد قيم ابتدائية لعناصرها في أمر واحد كما هو واضح في المثال التالي والذي يبدو أقصر ويعمل بشكل أسرع مع المصفوفات الصغيرة الحجم:

```
Dim testScores() As Integer = {75, 80, 70, 85, 90}
```

من المهم عزيزي القارئ أن تلاحظ أن المتغير testScore هو متغير إشارة أي يشير إلى كائن من الصنف Arrays الذي يحتوي على خمس عناصر من النوع Integer كما هو واضح في الشكل رقم (٣،٢٠)، وكما هو ملاحظ أن كائن المصفوفة يحتوي على صفة (Attribute) يطلق عليها Length والتي تحتوي على عدد عناصر المصفوفة (خمس عناصر) وسوف نتعرف على هذه الصفة أكثر في المثال التالي.



الشكل رقم (٣،٢٠). مصفوفة تشمل خمسة عناصر Integer.

يمكن عرض محتويات عناصر المصفوفة بنفس الأسلوب المتبع لعرض محتويات المتغيرات العادية حيث تعرض الأوامر التالية محتويات كل عنصر من عناصر المصفوفة testScore، ولاحظ أن جميع الأوامر متشابهة عدا فهرس العنصر هو فقط الذي يتغير.

```
Console.WriteLine("test score 1 = " & testScores(0))
Console.WriteLine("test score 2 = " & testScores(1))
Console.WriteLine("test score 3 = " & testScores(2))
Console.WriteLine("test score 4 = " & testScores(3))
Console.WriteLine("test score 5 = " & testScores(4))
```

افترض أنك تريد أن تحسب متوسط محتويات عناصر المصفوفة فيجب عليك جمع المحتويات ثم تقسيمها على ٥ كما هو واضح في المثال التالي:

```
Dim average  As Double
average = average + testScores(0)
average = average + testScores(1)
average = average + testScores(2)
average = average + testScores(3)
average = average + testScores(4)
average = average / 5
Console.WriteLine ("average score is " & average)
```

لاحظ أن جميع أوامر الجمع متشابهة عدا قيمة فهرس العنصر حيث كان صفراً للعنصر الأول، و ١ للعنصر الثاني، وهكذا. لذلك يمكن استبدال أوامر الجمع هذه بأمر واحد فقط داخل تكرار مع استخدم عداد التكرار كفهرس للعناصر وخصوصاً عندما يكون حجم المصفوفة كبيراً (١٠٠ عنصر مثلاً).

ولتوضيح ذلك فإن المثال التالي يستخدم التكرار For لجمع عناصر المصفوفة:

```
Dim average As Single
Dim i As Integer
For i = 0 To 4
      average += testScores(i)
Next
```

نلاحظ أن عداد التكرار يبدأ من الصفر والذي يمثل فهرس العنصر الأول للمصفوفة ونهاية عداد التكرار ٤ والذي يمثل الفهرس العنصر الأخير للمصفوفة، ومن ثم فإن أول تكرار (عندما يكون المتغير i يساوي صفراً) سيتم جمع العنصر الأول للمصفوفة، والتكرار الثاني (عندما تكون i تساوي ١) سيتم جمع العنصر الثاني، وهكذا (لاحظ أنه يمكنك بكل سهولة تغيير حدود التكرار من صفر–٤ إلى ٠–٩٩ لجمع ١٠٠ عنصر داخل مصفوفة). وبعد أمر التكرار For يمكن إضافة أمر لقسمة المتغير على ٥ لحساب المتوسط. كما هو موضح في الشكل رقم (٣.٢٠) فإن كائن الصنف Arrays يحتوي على صفة Length والتي تُستخدم لاحتواء عدد عناصر المصفوفة والتي هي في هذه الحالة ٥ عناصر حيث يمكن استبدال القيمة النهائية لعداد التكرار i في التكرار السابق من ٤ إلى Length-1 مستخدماً الصفة Length التي تجعلنا لا نحتاج لتغيير التكرار For عندما يتغير حجم المصفوفة، ولذلك يمكن كتابة تكرار For لاستخدامه مع أي مصفوفة كما يلي:

```
Dim average As Single
Dim i As Integer
For i = 0 To testScores.Length - 1
      average += testScores(i)
Next
```

يحتوي الملف ArraysDemo.vb على جميع هذه الأمثلة والموضحة في الشكل رقم (٣.٢١)، ويوضح الشكل رقم (٣.٢٢) مخرجات هذا البرنامج.

```
' Chapter 3 ArrayDemo Example 6
Option Strict On
Module ArrayDemo
   Sub Main()
   ' declare an integer array with 5 elements
   Dim testScores(4) As Integer
   ' populate the array
   testScores(0) = 75
   testScores(1) = 80
   testScores(2) = 70
   testScores(3) = 85
   testScores(4) = 90

   'display the element contents
   Console.WriteLine("test score 1 = " & testScores(0))
   Console.WriteLine("test score 2 = " & testScores(1))
   Console.WriteLine("test score 3 = " & testScores(2))
   Console.WriteLine("test score 4 = " & testScores(3))
   Console.WriteLine("test score 5 = " & testScores(4))

   ' compute the average score using length attribute
   Dim average As Single
   Dim i As Integer
   For i = 0 To testScores.Length - 1
      average += testScores(i)
   Next
   average = average / testScores.Length
   Console.WriteLine("the average test score is " & _
      average)
   End Sub
End Module
```

الشكل رقم (٣,٢١). شفرة البرنامج ArrayDemo.vb.

```
test score 1 = 75
test score 2 = 80
test score 3 = 70
test score 4 = 85
test score 5 = 90
the average test score is 80
```

الشكل رقم (٣,٢٢). مخرجات البرنامج ArrayDemo.vb.

استخدام مصفوفات متعددة الأبعاد

Using Multidimensional Array

بالإضافة إلى المصفوفة ذات البعد الواحد، فإن لغة VB .NET تمكنك عزيزي القارئ من إنشاء مصفوفات ذات أبعاد متعددة والتعامل معها حيث تتشابه المصفوفة ذات البعدين (Two-Dimensional Arrays) مع الجدول الذي يتكون من صفوف وأعمدة، وتتشابه المصفوفة ذات الأبعاد الثلاثة (Three-Dimensional Arrays) مع المكعب الذي

يتكون من صفوف وأعمدة ومستويات مع العلم بأن كل بعد له الفهرس (Index) الخاص به. وغالباً يتم التعامل مع مصفوفات ذات بعد أو بعدين فقط.

فعلى سبيل المثال، يمكن تغيير مثال درجات الاختبار السابق ليشتمل على مصفوفة ذات بعدين الذي يتكون من خمسة صفوف وعمودين. تمثل الأعمدة الاختبارات بينما تمثل الصفوف الخمسة خمسة طلبة (كما هو واضح في الجدول رقم ٣.٦) ويحتوي العمود الأول على نفس محتويات المصفوفة ذات البعد الواحد الموجودة في المثال السابق.

الجدول رقم (٣.٦). درجات الاختبار.

	Test 1	Test 2
Student 1	75	80
Student 2	80	90
Student 3	70	60
Student 4	85	95
Student 5	90	100

يتم تعريف المصفوفة ذات البعدين بنفس الأسلوب المستخدم لتعريف المصفوفة ذات البعد الواحد ولكن يتم تحديد عدد كل من الصفوف والأعمدة، ويعرِّف البرنامج التالي مصفوفة ذات بعدين تحتوي على أرقام صحيحة وتسمى testScoreTable التي تتكون من خمسة صفوف وعمودين. لاحظ أنه قد تم كتابة فهرس آخر صف (٤) وفهرس آخر عمود (١)

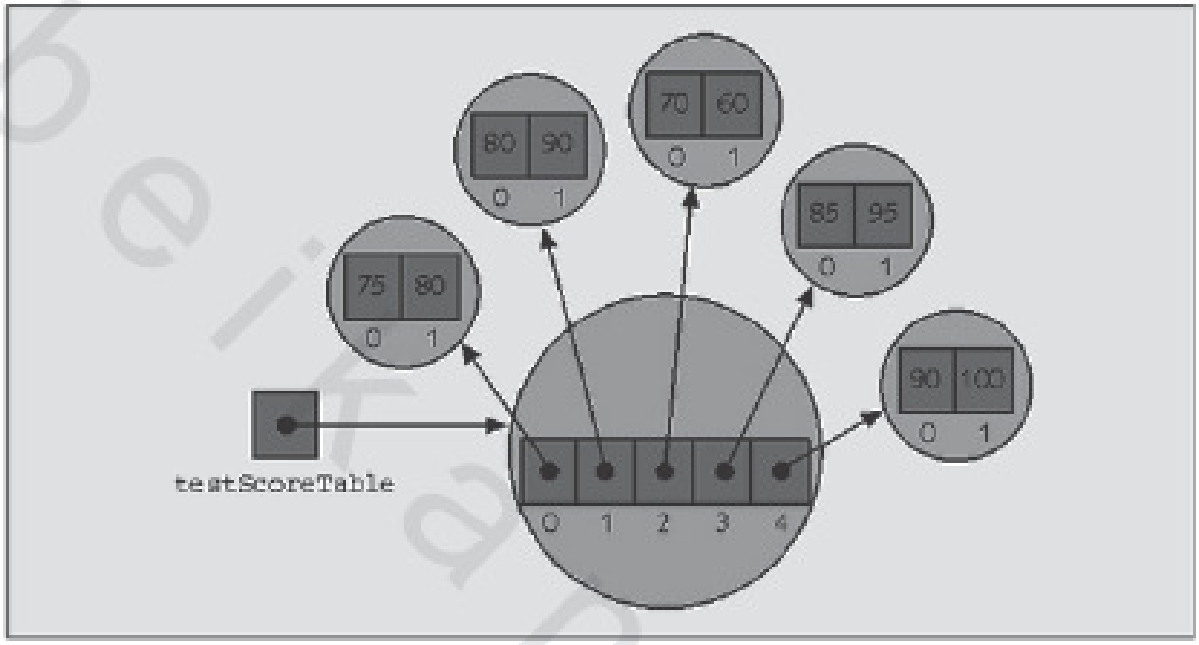
```
Dim testScoreTable(4, 1) As Integer
```

يتم إسناد القيم لعناصر المصفوفة مستخدماً الفهارس وبنفس الأسلوب المتبع لإسناد قيم لعناصر المصفوفة ذات البعد الواحد.

```
' populate the elements in column 1
testScoreTable(0, 0) = 75
testScoreTable(1, 0) = 80
testScoreTable(2, 0) = 70
testScoreTable(3, 0) = 85
testScoreTable(4, 0) = 90

' populate the elements in column 2
testScoreTable(0, 1) = 80
testScoreTable(1, 1) = 90
testScoreTable(2, 1) = 60
testScoreTable(3, 1) = 95
testScoreTable(4, 1) = 100
```

تمثل لغة VB .NET المصفوفات ذات الأبعاد المتعددة كمصفوفات ذات بعد واحد تحتوي على مصفوفات ذات بعد واحد وهكذا. ويوضح الشكل رقم (٣,٢٣) التركيب الداخلي للمصفوفة testScoreTable.



الشكل رقم (٣,٢٣). مصفوفة المصفوفات.

يشير متغير الإشارة testScoreTable إلى كائن مصفوفة ذات بعد واحد والتي تحتوي على خمسة عناصر (عنصر لكل صف)، وكل عنصر من هذه العناصر ما هو إلا متغير إشارة يشير إلى كائن مصفوفة آخر يحتوي على عنصرين (عنصر لكل عمود)، ومن الملاحظ أنه يوجد فعلياً ستة كائنات مصفوفة ذات بعد واحد وكل واحد منها يحتوي على عناصر مختلفة وله طول خاص به، ومن ثم فإن أطوال صفوف المصفوفة ذات البعدين ربما تختلف من صف إلى آخر، ولذلك يمكنك إنشاء مصفوفة تتكون من صفوف ذات أطوال مختلفة. بمعنى آخر يمكنك إنشاء مصفوفة ذات بعدين حيث يحتوي الصف الأول على عنصرين ويحتوي الصف الثاني على خمسة عناصر وهكذا.

يحتوي كل عمود من أعمدة هذه المصفوفة على درجات اختبار مختلف حيث يحتوي العمود الأول على درجات الاختبار الأول ويحتوي العمود الثاني على درجات الاختبار الثاني، وفي الحقيقة توجد هناك ثلاثة أساليب لحساب متوسط كل اختبار:

١- الطريقة الأولى: كتابة أوامر منفصلة لتجميع قيم كل عمود ثم حساب متوسطه.

٢- الطريقة الثانية: كتابة تكرارين منفصلين لحساب متوسط كل عمود.

٣- الطريقة الثالثة: كتابة تكرارات متداخلة (تكرار داخل تكرار) لكي تجمع عناصر الأعمدة ثم حساب متوسطها.

يوضح البرنامج التالي الطريقة الأولى لحساب مجموع عناصر العمود الأول ثم حساب متوسطه حيث تم استخدام المعامل += الذي يجمع قيمة المتغير average على أحد عناصر المصفوفة testScoreTable ثم إسناده للمتغير average، ويجب أن تلاحظ أيضاً أن فهرس الأعمدة ظل يأخذ القيمة صفر بينما تغيرت قيمة فهرس الصفوف من صف إلى أربعة. ربما تحتاج أن تعيد البرنامج نفسه لحساب متوسط العمود الثاني:

```
average = 0
average += testScoreTable(0,0)
average += testScoreTable(1,0)
average += testScoreTable(2,0)
average += testScoreTable(3,0)
average += testScoreTable(4,0)
Console.WriteLine("test 1 average is " & average/5)
```

يوضح البرنامج التالي كيفية حساب مجموع متوسط العمود الأول مستخدماً الطريقة الثانية وهي كتابة تكرار منفصل لكل عمود وربما تحتاج أيضاً أن تعيد البرنامج نفسه لحساب متوسط العمود الثاني:

```
average = 0
' sum column 1
For row = 0 To 4
      average += testScoreTable(row,0)
Next
Console.WriteLine("test 1 average is " & average/5)
```

في هذا البرنامج أطلقنا على فهرس الصفوف اسم row والذي تتغير قيمته داخل التكرار من صفر إلى ٤، أما فهرس الأعمدة ظل ثابتاً (صفر).

أما البرنامج التالي فيوضح حساب متوسط كل من العمود الأول والعمود الثاني مستخدماً الطريقة الثالثة وهو استخدام تكرارات متداخلة حيث يدور التكرار الخارجي (Outer Loop) حول الأعمدة، بينما التكرار الداخلي (Inner Loop) فيدور حول الصفوف، ولذلك فإن كل مرة تكرار للتكرار الخارجي يتكرر التكرار الداخلي خمس مرات.

```
' compute the average test score using nested for next loop
Dim row, col As Integer
Dim average As Single
For col = 0 To 1
      average = 0
```

```
        For row = 0 To 4
              average += testScoreTable(row, col)
Next    ' end of inner loop
        ' compute the column average
        average = average / 5
        Console.WriteLine("test " & (col + 1) & _
              " average score is " & average)
Next                        ' end of outer loop
```

بدأ البرنامج بتوصيف متغيرين عدّادين (المتغيرات col و row) بالإضافة للمتغير average الذي سوف يحتوي على النتائج النهائية، ثم بعد ذلك توجد شفرة التكرار الخارجي التي ستنفذ مرتين (مرة لكل عمود) حيث يستخدم العداد (col) كفهرس للأعمدة ويأخذ القيمة صفر في المرة الأولى عند تنفيذ التكرار الخارجي والقيمة واحد في المرة الثانية من تنفيذ التكرار الخارجي.

في بداية التكرار الخارجي يأخذ المتغير average القيمة صفر وذلك لضمان عدم احتوائه على قيمة مع العلم أن التكرار الخارجي سوف ينفذ مرتان (مرة لكل عمود)، وفي نهايته عند تنفيذ التكرار الخارجي للمرة الأولى سيحتوي على متوسط العمود الأول.

ثم يأتي التكرار الداخلي الذي يستخدم العداد row فهرساً للصفوف والذي ينفذ خمسة مرات وذلك مرة لكل صف حيث يحتوي هذا التكرار على أمر واحد وهو المسؤول عند جمع قيمة العنصر المفهرس بالعداد row والعداد col إلى قيمة المتغير average.

```
average += testScoreTable(row, col)
```

بعد تكرار تنفيذ التكرار الداخلي خمسة مرات سوف يحتوي المتغير average على مجموع عناصر العمود ثم يأتي الأمر المسؤول عن حساب المتوسط وذلك بقسمة المتغير average على خمسة ثم يأتي آخر أمر موجود في التكرار الخارجي لعرض محتويات المتغير average.

```
average = average / 5
Console.WriteLine("test " & (col + 1) & " average score is " & average)
```

يعرض الشكل رقم (٣.٢٤) البرنامج TwoDimArrayDemo.vb، كما يعرض الشكل رقم (٣.٢٥) مخرجات هذا البرنامج.

```
' Chapter 3 TwoDimArrayDemo Example 7
Option Strict On
Module TwoDimArrayDemo
   Sub Main()
   ' declare an integer array with 5 rows and 2 columns
   Dim testScoreTable(4, 1) As Integer
   ' populate the elements in column 1
   testScoreTable(0, 0) = 75
   testScoreTable(1, 0) = 80
   testScoreTable(2, 0) = 70
   testScoreTable(3, 0) = 85
   testScoreTable(4, 0) = 90
   ' populate the elements in column 1
   testScoreTable(0, 1) = 80
   testScoreTable(1, 1) = 90
   testScoreTable(2, 1) = 60
   testScoreTable(3, 1) = 95
   testScoreTable(4, 1) = 100

   ' compute the average test score using nested loop
   Dim row, col As Integer
   Dim average As Single
   For col = 0 To 1
      average = 0
      For row = 0 To 4
         average += testScoreTable(row, col)
   Next  ' end of inner loop

      ' compute the column average
      average = average / 5
      Console.WriteLine("test " & (col + 1) & _
         " average score is " & average)
   Next  ' end of outer loop
   End Sub
End Module
```

الشكل رقم (٣,٢٤). شفرة البرنامج TwoDimArrayDemo.vb.

```
test 1 average score is 80
test 2 average score is 85
```

الشكل رقم (٣,٢٥). مخرجات البرنامج TwoDimArrayDemo.vb.

ملخص الفصل
Chapter Summary

- مع أن لغة VB .NET لغة حديثة (تم إصدارها في بداية عام ٢٠٠٢م) ولكنها اكتسبت شهرة واسعة وذلك بسبب العديد من المميزات التي تحتويها مثل دعم تقنية الكائنات وسهولة تعلمها واستخدامها. والأهم من ذلك فإنها لغة تدعم تطوير برامج تعمل على شبكات الحاسب.
- تأتي قوة لغة VB .NET من وجود عدد هائل من الأصناف سابقة التعريف (Predefined classes) والتي بدورها تحتوي على أساليب (Methods) تستخدم في إنجاز مهام مختلفة بداية من مهمة تنسيق الأرقام إلى إنجاز الاتصال بالشبكات والتعامل مع قواعد البيانات.

- تمثل معرفات لغة VB .NET (Identifiers) أسماء تسند بواسطة المبرمج للأشياء مثل اسم الوحدة البرمجية، واسم الإجراء، واسم المتغير، وهكذا.
- يستخدم التعليق لإضافة شرح لشفرة البرنامج حيث يضاف التعليق في لغة VB .NET باستخدام علامة التنصيص الفردية (') والذي يطلق عليه Single quote وما يتم كتابته خلف هذا الرمز يمثل تعليقاً.
- إذا احتوى الملف البرمجي على الإجراء Main فعند تحميل هذا الملف في الذاكرة فإن هذا الإجراء يتم استدعاءه مباشرة وتنفيذ أوامره.
- المتغير (Variable) هو مكان في الذاكرة (Memory) يحتوي على بيانات، ولكي تستخدم متغيراً يجب عليك توصيفه أولاً حيث يحدد لكل متغير اسمه ونوع بياناته (Data Type) والقيمة التي سيحتويها.
- تحتوي لغة VB .NET على تسعة أنواع بيانات أولية (Primitive)، وتم إطلاق اسم أولي على هذه الأنواع لتمييزها عن أنواع البيانات المعقدة مثل استخدام الأصناف والتي سيتم مناقشتها في الفصل الرابع.
- لكي تقوم بتوصيف متغير لابد من استخدام الكلمة الأساسية Dim متبوعةً باسم المتغير ثم الكلمة As ثم نوع البيانات، كما يمكنك توصيف أكثر من متغير من نفس النوع داخل أمر واحد.
- يمكنك إسناد قيمة للمتغير مستخدماً المعامل يساوي = (Assignment Operator) أي معامل الإسناد والذي يستخدم لإسناد القيمة التي تقع على يمين المتغير إليه.
- إذا أسندت القيمة On للاختيار Option Explicit فإن المبرمج يجب أن يعرف أي متغير قبل استخدامه، أما إذا أسندت القيمة Off له فيجوز استخدام متغير دون تعريفه.
- إذا تم ضبط الأمر Option Strict على On فإن مترجم اللغة سيعطي خطأً وذلك عند أي أمر ربما يؤدي إلى الفقد غير المقصود للجزء العشري في الأرقام الكسرية.
- لتعريف متغير ثابت (Constant) يتم استخدام الكلمة الأساسية Const بدلاً من الكلمة الأساسية Dim، ويجب إسناد قيمة للمتغير الثابت أثناء تعريفه.
- تحتوي لغة VB .NET على نوعين من المتغيرات وهما: المتغيرات الأولية (Primitive Variables) ومتغيرات الإشارة (Reference Variables)، ويتم تعريف المتغيرات الأولية خلال أنواع البيانات الأولية التسعة، وقد سميت المتغيرات بهذا الاسم لأنها تخزن البيانات التي تسند إليها بداخلها، أما متغيرات الإشارة فيتم تعريفها بواسطة اسم الصنف (Class Name) والذي يعمل كنوع بيانات ومن ثم سيشير متغير الإشارة إلى كائن معرف من هذا الصنف والذي بدوره يحتوي على البيانات.
- يمثل المعامل & معامل لصق (Concatenation Operator) والذي يلصق نص حرفين معاً لإصدار نص واحد.

- تستخدم لغة VB .NET المعاملات الحسابية المشهورة (+ ، – ، / ، x) للجمع والطرح والقسمة والضرب مثلها مثل معظم اللغات، وتستخدم أيضاً الأقواس لتغيير أولوية تنفيذ أجزاء التعبير الرياضي.
- كما تستخدم لغة VB .NET المعامل Mod لحساب باقي القسمة والذي يطلق عليه Remainder Operator أو Module Operator.
- تحتوى لغة VB .NET أيضاً على الصنف Math والذي يتضمن الإجراءات التي تستخدم لحساب الأس، والتقريب، والعديد من العمليات الحسابية الأخرى.
- يمكن استخدام معامل الإسناد = مع المعاملات الحسابية المشهورة (+ ، – ، / ، x) لإنتاج معاملات إسناد مركبة (+= ، -= ، /= ، x=).
- تقدم لغة VB .NET أسلوبين لكتابة أوامر صنع القرار الأول هو أمر If الشرطي والأمر الثاني هو أمر Select Case.
- إن أمر If الشرطي مثل جميع أوامر If ببقية اللغات حيث يتم تقييم شرط معين فإذا كانت نتيجة التقييم إيجابية فعندئذ يتم تنفيذ أمر (أو مجموعة من الأوامر).
- يأخذ أمر If شكلاً من صورتين: الصورة الأولى (يطلق عليها أمر If البسيط) تقييم التعبير ثم تنفيذ أمر (أو مجموعة من الأوامر) إذا كانت نتيجة التعبير إيجابية، أما الصورة الثانية فهي أمر If-Else التي تقيم تعبيراً منطقياً ثم تنفيذ أمر (أو مجموعة من الأوامر) إذا كانت نتيجة التعبير إيجابية أو تنفيذ أمر آخر أو أكثر إذا كانت نتيجة التعبير سلبية.
- يوجد شكل لأمر If وهو أمر If المتداخل (Nested If) وهي عبارة عن أمر If داخل أمر If.
- يتكون الشرط المركب (Compound Expression) من شرطين أو أكثر بينها المعامل المنطقي or أو المعامل المنطقي and.
- يمثل المعامل = معامل مقارنة التساوي بين حدين، أما المعامل < > يمثل معامل مقارنة عدم التساوي بين حدين.
- يشبه أمر Select Case الأمر If متعدد المسارات حيث يتم حساب قيمة مصطلح ومقارنة نتيجة هذا التعبير مع عدة قيم (حيث تمثل كل قيمة حالة)، فإذا تساوى نتيجة التعبير وأحد هذه القيم يتم تنفيذ الأمر (أو الأوامر) المرافقة لهذه الحالة.
- تقدم لغة VB .NET ثلاثة مجموعات من الكلمات المحجوزة لكتابة التكرارات وهي: For Next و Do Until و Do While.

- بشكل عام يمكن تصنيف تكرارات لغات البرمجة إلى نوعين: النوع الأول هو Pre-test loop والذي يختبر الشرط ثم يقرر تنفيذ الأوامر أو التوقف عن التنفيذ بناء على قيمة الشرط، أما النوع الثاني فهو Post-test loop وهو عكس النوع الأول حيث ينفذ الأوامر المطلوب تكرارها أولاً ثم يختبر الشرط، وبناء على نتيجة الشرط يقرر تنفيذ الأوامر مرة أخرى أو أن يتوقف عن التكرار.
- التكرارات المتداخلة (Nested loops) هي تكرار داخل تكرار آخر.
- تتكون المصفوفة ذات البعد الواحد من صف من العناصر، أما المصفوفة ذات البعدين فتتكون من صفوف وأعمدة والمصفوفة ذات الأبعاد الثلاث فهي مثل المكعب الذي يتكون من صفوف وأعمدة وصفحات.
- يمكن التعامل مع عناصر المصفوفة بكتابة اسم المصفوفة متبوعاً بفهرس العنصر داخل قوسين.
- يحتوي كائن المصفوفة على صفة يطلق عليها Length والتي تحتوي على عدد عناصر المصفوفة.

المصطلحات الأساسية

Key Terms

بيانات مرسلة للإجراء (Argument)	تكرار (Pre-test)
معامل حسابي (Arithmetic Operator)	تكرار (Post-test)
معامل إسناد (Assignment Operator)	التكرار المتداخل (Nested loop)
معامل لصق (Concatenation Operator)	مصفوفة ذات بعد واحد (One-Dimensional Array)
متغير ثابت (Constant)	مصفوفة ذات بعدين (Two-Dimensional Array)
متغير أولي (Primitive Variable)	مصفوفة ذات ثلاث أبعاد (Three-Dimensional Array)
بيانات أولية (Primitive Data Type)	كلمة محجوزة (Keyword)
معرف (Identifier)	لا قيمة (Nothing)
متغير إشارة (Reference Variable)	الإجراء (Procedure)
مصطلح مركب (Compound Expression)	ملف البرنامج (Module)
الاختيار (Option Strict)	أمر If المتداخل (Nested If)
الاختيار (Option Explicit)	تعليق (Comment)

أسئلة المراجعة

Review Questions

١- ما هي بيئة التطوير المتكاملة IDE؟

٢- ما هي معرفات الفيجوال بيسك .نت؟ وما هي قواعد إنشاء أحدها؟

٣- ما هي الكلمة المحجوزة؟

٤- ما هو المتغير الأولي؟

٥- ما هو تعريف الثابت؟

٦- ماذا تعني الكلمة المحجوزة لا شيء Nothing؟

٧- ما هو المعطى؟

٨- ما هو الإجراء الرئيس؟

٩- اشرح الفرق بين معامل القسمة ومعامل الباقي.

١٠- ما الغرض من تعبير "اختيار الحالة" "Select Case"؟

١١- ما هي أنواع التكرار؟ وكيف يتعامل الفيجوال بيسك .نت مع كل منها؟

١٢- ما هو الفهرس؟

١٣- كيف يتم تعريف المصفوفة ذات البعد الواحد؟

١٤- كيف تستطيع قول أن المصفوفة في الحقيقة حالة؟

١٥- ما هو صفة طول المصفوفة ذات البعد الواحد؟

أسئلة المناقشة
Discussion Questions

١- ما هي مميزات استخدام الإجراءات بدلاً من الاعتماد على الإجراءات المعرفة داخل اللغة؟

٢- ما هو تأثير استخدام الكلمات المحجوزة داخل إجراءات البرنامج؟

٣- لماذا تعتقد أن لغة فيجوال بيسك .نت تقدم ثلاثة أوامر لكتابة أوامر التكرارات؟ أليس من الأسهل استخدام أمر تكرار واحد مثل For Next؟

٤- ما هي فائدة ضبط كل من الاختيار Option Explicit والاختيار Option Strict بالقيمة On؟

مشاريع الفصل
Projects

١- افترض أن هناك أحد البنوك يقدم حساباً يبدأ بقيمة ١٠٠ دولار وزيادة سنوية تساوي ٥٪. يتم حساب الدخل الكلي من المعادلة التالية:

```
newBalance = previousBalance + (1 + interestRate)
```

اكتب ملفاً برمجياً ComputeIntereset يُستخدم لحساب الدخل السنوي لمدة خمسة سنوات متتالية دون استخدام أوامر تكرار في المشروع الأول؟

٢- أعد كتابة المشروع الأول مستخدماً كلاً من التكرار Do While والتكرار Do Until والتكرار For Next؟ أي من هذه التكرارات مناسبة لهذا المشروع؟

٣- يحتوي الجدول التالي على مبيعات ربع سنوية لخمسة أقسام في شركة ما:

القسم	ربع السنة ١	ربع السنة ٢	ربع السنة ٣	ربع السنة ٤	المجموع
القسم ١	٧٥٠	٦٦٠	٩١٠	٨٠٠	
القسم ٢	٨٠٠	٧٠٠	٩٥٠	٩٠٠	
القسم ٣	٧٠٠	٦٠٠	٧٥٠	٦٠٠	
القسم ٤	٨٥٠	٨٠٠	١٠٠٠	٨٥٠	
القسم ٥	٩٠٠	٨٠٠	٩٦٠	٩٨٠	
المجموع					

اكتب ملفاً برمجياً SalesAnalysis يقوم بالتالي:

(أ) عرف مصفوفة ذات بعدين Sales، ثم قم بملء أعمدة المصفوفة بمحتويات الجدول العلوي.

(ب) اكتب أمر تكرار لحساب المجموع في العمود الأخير بشرط أن تظهر مجموع مبيعات كل قسم داخل أمر التكرار.

(ج) اكتب أمر تكرار لحساب المجموع في الصف الأخير بشرط أن تظهر مجموع مبيعات كل ربع سنة داخل أمر التكرار.

٤- أكمل برنامج المشروع الثالث بكتابة أمر تكرار متداخل لحساب واعرض التالي:

(أ) نسبة المبيعات الكلية لكل قسم / ربع سنوي.

(ب) نسبة المبيعات الكلية لكل ربع سنة.