بسم الله الرحمن الرحيم

مبادئ البرمجة

C#

فهرس

مقدمة

طوّرت شركة مايكروسوفت Microsoft منصّة عمل سمّتها دوت نت .NET framework وبيئة عمل سمّتها (Visual Studio IDE) حيث أصبح بإمكان مطوّر التّطبيقات الاستعانة بهذين الأخيرين بالإضافة للغة برمجة تعمل تحت هذه المنصّة أن يطوّر التّطبيقات المكتبيّة وتطبيقات الويب والتّطبيقات الموزّعة أيضاً.

سوف نتحدّث عن مبادئ لغة السي شارب (C#) التي طوّرتها مايكروسوفت خصّيصاً لمنصّة .NET).بعد ذلك يمكنك التوسع في لغة C# حيث زوّدت بالعديد من الميّزات التي تجعلها من أقوى لغات البرمجة الغرضيّة التّوجّه Object Oriented Programming ومن أهمّ هذه الميّزات:

أوّلاً : معالجة السّلاسل المحرفيّة.(Strings)

ثانياً : الرّسوميّات.(Graphics)

ثالثاً : الواجهات التّخاطبيّة.(Graphical User Interface)

رابعاً : معالجة الاستثناءات.(Exception Handling)

خامساً : النياسب المتعدّدة.(Multi-Threading)

سادساً : التّعامل مع الملفّات.(File Streams)

سابعاً : الوسائط المتعدّدة (صوت ،صورة ،فيديو(Multimedia))

ثامناً : التّكامل مع قواعد البيانات.(ADO.NET)

تاسعاً : التّطبيقات الشّبكيّة.(Network Programming)

أخيراً : التّطبيقات الموزّعة (Distributed Applications).

المتغيرات Variables

يتم استخدام المتغيرات لتخزين البيانات في الذاكرة أثناء تنفيذ البرنامج.

أنواع البيانات Data Types

اعتماداً على البيانات التي تحتاج إلى تخزينها ، هناك عدة أنواع مختلفة من أنواع البيانات .تتكون الأنواع البسيطة في C # من أربعة أنواع صحيحة مع إشارة وأربعة بدون إشارة ، وثلاثة أنواع من الفاصلة العائمة ، وكذلك bool و char .

Bool قيمة بوليانية لتخزين القيم المنطقية وهي أما true أو false

Int عدد صحيح مثل العدد 5

Float عدد حقيقي ذات الفاصلة العائمة بدقة عادية مثل 3.4

Double عدد حقيقي ذات الفاصلة العائمة بدقة مزدوجة.

والجدول 1 يوضح أنواع البيانات

Data	Type Size (Bits)	Range	Description
sbyte	8	-128 to 127	Signed integers
short	16	-32,768 to 32,767	
int	32	-2,147,483,648 to 2,147,483,647	
long	64	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	
byte	8	0 to 255	Unsigned integers
ushort	16	0 to 65,535	
uint	32	0 to 4,294,967,295	
ulong	64	0 to 18,446,744,073,709,551,615	
float	32	1.5×10^{-45} to 3.4×10^{38}, 7-digit precision	Floating-point numbers
double	64	5.0×10^{-324} to 1.7×10^{308}, 15-digit precision	
decimal	128	at least -7.9×10^{-28} to 7.9×10^{28}, with at least 28-digit precision	
char	16	stored as an integer between 0 and 65535	Unicode character
bool	4	Boolean value, true or false	Boolean value

جدول 1: أنواع البيانات

تعريف و تسمية المتغيرات Declaring And Naming Variables

لتعريف المتغير يجب أن نشير لنوعه و نعطه اسم وبشكل اختياري نسند إليه قيمة أولية.

```
<Type> <Name> ;
int _myVariable=100;
string my_Lable="some string";
bool myBool=False;
```

إن اسم المتغير يجب أن يبدأ بحرف Letter أو بخط سفلي Underscore بعد ذلك يمكن ان نكمل الاسم بأعداد أو أحرف أو بخط سفلي Underscore حتى 255 محرف. ولا يجوز أن يحتوي على رموزًا خاصّة مثل & $ # أو فراغات Space .

مثال First Name هذا الاسم غير صحيح لأنه يوجد فراغ.

مثال : تصريح عن متغير MyVariable_1_ من نوع int

```
int _MyVariable_1;
```

كما أن اسماء المتغيرات حساسة لحالة الأحرف أي يوجد اختلاف بين اسم المتغير A و اسم المتغير a. بالإضافة إلى أنه لا يمكن تسمية اسم محجوز ضمن لغة C# مثل new أو class أو غيرها. كذلك يشترط الإعلان عن المتغيرات قبل استخدامها .

ملاحظة (للاطلاع)

إعتبارا من C # 7.0 يمكن التصريح عن قيمة ثنائية وست عشرية .

```
int myHex = 0xF; // 15 in hexadecimal (base 16)
int myBin = 0b0100; // 4 in binary (base 2)
```

الإسناد Assignment

يتم تعيين قيمة للمتغير باستخدام إشارة التساوي ، باستخدام معامل الإسناد :

operator assignment (=) .

```
int MyInt;

MyInt=10;
```

يمكن الجمع بين التصريح والإسناد في تعليمة واحدة وتدعى هذه العملية بالتهيئة initialization.

```
int MyInt=10;
```

إذا كانت هناك حاجة لمتغيرات متعددة من نفس النوع ، فيوجد هناك طريقة مختصرة للتصريح أو تهيئة المتغير باستخدام معامل الفاصلة comma operator (,) .

```
int myInt = 10, myInt2 = 20, myInt3;
```

بمجرد التصريح عن متغير ، يمكن أن يستخدم من خلال الإشارة إلى اسم المتغير.

```
System.Console.Write(myInt); // "10"
```

أنواع البيانات الرقمية Integer Types

هناك أربعة أنواع صحيحة Signed Integer مع إشارة يمكن استخدامها بناءً على قيمة العدد .

```
// Signed integers
sbyte myInt8 = 2; // -128 to +127
short myInt16 = 1; // -32768 to +32767
int myInt32 = 0; // -2^31 to (+2^31)-1
long myInt64 =-1; // -2^63 to (+2^63)-1
```

يمكن استخدام الأنواع الرقمية بدون إشارة Unsigned Integer إذا احتجت فقط إلى تخزين القيم الموجبة .

```
// Unsigned integers
byte uInt8 = 0; // 0 to 255
ushort uInt16 = 1; // 0 to 65535
uint uInt32 = 2; // 0 to 2^32-1
ulong uInt64 = 3; // 0 to 2^64-1
```

أنواع الفاصلة العائمة Floating-Point Types

يمكن للأنواع الرقمية للفاصلة العائمة(العشرية) تخزين أرقام حقيقية بمستويات مختلفة من الدقة.

يجب إلحاق الحرف M إلى قيمة المتغير decimal. وعندما يتم اسناد قيمة أكبر من 28 خانة يتم تقريب القيمة .

كذلك يتم إلحاق الحرف f إلى قيمة المتغير float. وعندما يتم اسناد قيمة أكبر من 7 خانات يتم تقريب القيمة .

كما هو موضح في المثال التالي:

```
float myFloat = 3.14F; // 7 digits of precision
double myDouble = 3.14; // 15-16 digits of precision
decimal myDecimal = 3.14M; // 28-29 digits of precision
myFloat = 12345.6789F; // rounded to 12345.68
```

ملاحظة:

إن C# تفرق بين القيمة الصحيحة 1 وبين القيمة العشرية 1.0 حيث يختلفان بالنوع وعلى وجه الدقة لايسمح بالقيام بعملية الاسناد التالية :

```
int x = 1.0;
```

لأن المتغير على اليسار من نوع int والقيمة التي تم اسندها من نوع double

ولكن تسمح C# باسناد عدد صحيح لمتغير من نوع Double أو float أو decimal

```
double d = 1;
```

ينتج عن ذلك اعطاء نتيجة غير صحيحة لنأخذ المثال التالي:

```
double d = 1 / 3;
```

قد تتوقع أن يعطي المتغير d القيمة 0.3333333 وهي قيمة عشرية لكنه في الواقع سيعطي القيمة 0 والسبب هو أن العبارة على اليمن هي قسمة عددين صحيحين .

إحدى طرق حل هذه المشكلة هي بجعل الطرف الايمن عشري

```
double d = 1.0 / 3.0;//d=0.3333333
```

نوع محرف Char Type

Char يمكن أن يتضمن محرف واحد بترميز Unicode character

```
char c = '3'; // Unicode char
```

نوع بولياني Bool Type

يتم تخزين قيمة بوليانية اما True أو False ضمن المتغير ويستفاد منها في تعديل تدفق البرنامج.

مجال رؤية المتحولات Variable Scope

عند التصريح عن متحول ضمن تابع لا يمكن رؤية المتحول خارج نطاق التابع .

```
static void Main()
{
int localVar; // local variable
}
```

المعاملات Operators

معامل التشغيل هي رموز خاصة تستخدم مع القيم .تتكون من خمسة أنواع: الحساب ، الاسناد ، المقارنة المنطقية ، و العمليات على البتات.

المعاملات الحسابية Arithmetic Operators

تشمل المعاملات الحسابية العمليات الحسابية الأساسية الأربعة ،وكذلك معامل (%) ، والذي يستخدم للحصول على باقي القسمة.

```
float x = 3 + 2; // 5 // addition
x = 3 - 2; // 1 // subtraction
x = 3 * 2; // 6 // multiplication
x = 3 / 2; // 1 // division
x = 3 % 2; // 1 // modulus (division remainder)
```

لاحظ أن علامة القسمة تعطي نتيجة غير صحيحة .وذلك لأن 3 و 2 قيمتين صحيحتين .

و للحصول على القيمة الصحيحة نقوم بتغير نوع العدد الصحيح الى عدد عشري float

```
x = 3.0f / 2.0f; // 1.5
```

كذلك يمكن حل هذه المشكلة باستعمال قولبة الانماط (typecast)

```
x = (float) 3 / 2; // 1.5
```

معامل الاسناد Assignment Operators

هو معامل الاسناد (=) والذي يقوم بتعين قيمة للمتغير.

الجمع بين معاملات الاسناد Combined Assignment Operators

```
int x= 0;
x += 5; // x = x+5;
x -= 5; // x = x-5;
x *= 5; // x = x*5;
x /= 5; // x = x/5;
x %= 5; // x = x%5;
```

معاملات الزيادة والنقصان Increment and Decrement Operators

++ يقوم معامل الزيادة **Increment Operator** بزيادة قيمة المتغير بمقدار واحد فقط.

```
x++; // x = x+1;
```

-- معامل النقصان **Decrement Operator** يعمل على انقاص قيمة المتغير بمقدار واحد فقط.

```
x--; // x = x-1;
```

يمكن استخدام كلا معاملين قبل أو بعد المتغير.

```
x++; // post-increment بعد الزيادة
x--; // post-decrement بعد النقصان
++x; // pre-increment قبل الزيادة
--x; // pre-decrement قبل النقصان
```

إذا كان المعامل بعد اسم المتغير يؤدي ذلك إلى إرجاء عملية الزيادة أو النقصان وذلك بعد تنفيذ الجملة البرمجية التي يدخل المتغير فيها أولاً ثم إجراء عملية الزيادة أو لنقصان .

```
int x, y;
x = 5; y = x++; // y=5, x=6
x = 5; y = ++x; // y=6, x=6
```

معاملات المقارنة Comparison Operators

تقوم معاملات المقارنة بمقارنة قيمتين وإرجاع صواب True أو خطأ False . و تستخدم مع التعليمات الشرطية.

```
bool b = (2 == 3); // equal to (false)
b = (2 != 3); // not equal to (true)
b = (2 > 3); // greater than (false)
b = (2 < 3); // less than (true)
b = (2 >= 3); // greater than or equal to (false)
b = (2 <= 3); // less than or equal to (true)
```

المعاملات المنطقية Logical Operators

غالبًا ما تستخدم المعاملات المنطقية مع معاملات المقارنة .

معامل الضرب And (&&) يعطي صواب True إذا كان كل من يسار ويمين المعامل صواب True .

معامل الجمع OR (||) يعطي صواب إذا كان يسار أو يمين المعامل صواب True .

يتم استخدام معامل التشغيل المنطقي (!) not لعكس النتيجة.

```
bool b = (true && false); // logical and (false)
b = (true || false); // logical or (true)
b = !(true); // logical not (false)
```

معاملات البت Bitwise Operators

يمكن لمعاملات bitwise التعامل مع البت الفردي للعدد الصحيح integers .حيث يتم تمثيل العدد الصحيح بالنظام الثنائي قبل تطبيق معامل bitwise.

```
int x = 5 & 4; // and (0b101 & 0b100 = 0b100 = 4)
x = 5 | 4; // or (0b101 | 0b100 = 0b101 = 5)
x = 5 ^ 4; // xor (0b101 ^ 0b100 = 0b001 = 1)
x = 4 << 1; // left shift (0b100 << 1 = 0b1000 = 8)
x = 4 >> 1; // right shift (0b100 >> 1 = 0b10 = 2)
x = ~4; // invert (~0b00000100 = 0b11111011 = -5)
```

ملاحظة :

إن معامل الازاحة باتجاه اليسار left shift :يقوم بازاحة البتات باتجاه اليسار بمقدا ر قيمة العدد الصحيح الموجد على يمين المعامل << .

مثال:

```
x=5<< 2

//x=0b101 << 2

//x = 0b10100 =20
```

كذلك بالنسبة لمعامل الازاحة باتجاه اليمين right shift :يقوم بازاحة البتات باتجاه اليمين بمقدا ر قيمة العدد الصحيح الموجد على يمين المعامل >> .

مثال:

```
x=4 >> 2

//x=0b100 >> 2

//x = 0b001 =1
```

اسبقية المعاملات Operator Precedents

في C # ، يتم تنفيذ التعبير عادة من اليسار إلى اليمين .ومع ذلك عندما يحتوي التعبير على معاملات متعددة، فإن الأسبقية تكون حسب الجدول 2 التالي :

المعامل	الاسبقية
++ -- ! ~	1
* / %	2
+ -	3
<< >>	4
< <= > >=	5
== !=	6
&	7
^	8
|	9
&&	10
||	11
= op=	12

جدول 2:اسبقية المعاملات

```
bool x = 2+3 > 1*4 && 5/5 == 1; // true
```

كذلك يمكن التحكم في أسبقية المعاملات باستخدام الأقواس

```
bool x = ((2+3) > (1*4)) && ((5/5) == 1); // true
```

السلسلة المحرفية Strings

هو نوع من أنواع البيانات يستخدم لتخزين نصوص .

```
string a = "Hello";
```

يمكن للمعامل (+) الجمع بين السلاسل النصية معًا.

```
string b = a + " World"; // Hello World
a += " World"; // Hello World
```

عندما لا يكون أحد المعاملات من نوع Strings ، سيقوم معامل الجمع ضمنيًا بتحويل اي نوع ليس سلسلة non-string إلى سلسلة string ، مما يجعل التعليمة التالية صحيحة .

```
int i = 1;
string c = i + " is " + 1; // 1 is 1
```

يتم تحويل السلسلة ضمنيًا باستخدام تابع ()ToString

```
string d = i.ToString() + " is " + 1.ToString(); // 1 is 1
```

محارف الهروب Escape Characters

إن أردنا النزول لسطر جديد فسنكتب (Back Slash + n) \n ، عند هذه النقطةيقوم بالنزول إلى سطر جديد .

```
string myString = "Hello\nWorld";
```

يمكن تجاهل Back Slash \ بوضع @ قبل علامة التنصيص

```
string s1 = "c:\\Windows\\System32\\cmd.exe";
string s2 = @"c:\Windows\System32\cmd.exe";
```

وفي الجدول 3 يوضح محارف الهروب.

Character	Meaning	المعنى
\n	Newline	خط جديد
\t	Horizontal tab	مسافة أفقية
\v	Vertical tab	مسافة عامودية
\b	Backspace	مسافة للخلف
\r	Carriage return	إرجاع بداية السطر
\0	Null character	مسافة فارغة
\f	form feed	It skips to the start of the next page
\a	Alert sound	صوت تنبيه
\'	Single quote	علامة اقتباس واحد
\"	Double quote	علامة اقتباس مزدوج
\\	Backslash	علامة \
\uffff	Unicode character (four-digit hex number)	محرف: ارقم سداسي مكون من أربعة أرقام

جدول 3 : محارف الهروب

ملاحظة:

إن Slash هي التي تكتب بالشكل(/) أما الـ Back Slash فتكتب(\ ) .

خصائص و توابع السلسلة المحرفية String Members

إن string هو نوع type من فئة String Class حيث يضم مجموعة من التوابع يمكن الاستفادة منها .

Replace استبدال نص بنص.

Insert اضافة نص في مكان محدد .

Remove حذف جزء معين من النص من مكان محدد .

Substring اقتطاع نص محدد.

ToUpper تحويل النص الى احرف كبيرة

Length يعطي طول السلسلة المحرفية .

```
string a = "String";
string b = a.Replace("i", "o"); // Strong
b = a.Insert(0, "My "); // My String
b = a.Remove(0, 3); // ing
b = a.Substring(0, 3); // Str
b = a.ToUpper(); // STRING
int i = a.Length; // 6
```

مقارنة السلسلة المحرفية String Compare

من أجل عملية المقارنة نستخدم المعامل ==

```
string greeting = "Hi";
bool b = (greeting == "Hi"); // true
```

إلا أنه يفضل استدعاء تابع المقارنة بالطريقة التالية:

```
bool b = (greeting.Equals("Hi"));// true
```

ملاحظة للاطلاع

تعتبر فئة StringBuilder Class أفضل بديل عندما تحتاج سلسلة إلى تعديل عدة مرات.وذلك لأن فئة String Class ذات تكلفة اعلى .

```
System.Text.StringBuilder sb = new
System.Text.StringBuilder("Hello");
sb.Append(" World"); // Hello World
sb.Remove(0, 5); // World
sb.Insert(0, "Bye"); // Bye World
```

المصفوفات Arrays

المصفوفة هي بنية معطيات Data Structure تُستخدم لتخزين مجموعة من القيم جميعها من نفس نوع البيانات.

التصريح عن مصفوفة Array Declaration

للتصريح عن مصفوفة يتم إلحاق مجموعة من الأقواس المربعة [ ] بنوع البيانات التي ستحتويها

المصفوفة كما هو موضح في المثال التالي :

```
int[] x; // integer array
```

حيث تم التصريح عن مصفوفة جميع عناصرها ستكون من نوع عدد صحيح integer .

تخصيص المصفوفة Array Allocation

لتهيئة مصفوفة يمكنها استيعاب 3 عناصر من النوع int يكفي أن نكتب ما يلي:

```
int[] x = new int[3];
```

للعبارة السابقة وظيفتان:

الأولى هي التصريح عن المتغيّر x على أنّه مصفوفة عناصرها من النوع int وذلك عن طريق كتابة int[] أوّل العبارة.

والثانية هي إنشاء كائن Object المصفوفة وحجز 3 أماكن في الذاكرة بحيث يستطيع كلّ مكان منها استيعاب قيمة من النوع int وذلك عن طريق التعبير new int[3] ومن ثمّ إسناد المرجع لهذا الكائن إلى المتغير x. ويمكن أن نجري هذه العمليّة على شكل عبارتين منفصلتين.

```
int[] x ;
x= new int[3];
```

ملاحظة :

إن القيمة الافتراضيةلهذا النوع من البيانات هي الصفر.

اسناد قيم لعناصر المصفوفة Array Assignment

لكلّ عنصر في مصفوفة دليل index ، ويُعبّر عن ترتيب هذا العنصر ضمن المصفوفة. يبدأ ترقيم الأدلّة في أيّ مصفوفة بالصفر. أي أنّ دليل العنصر الأوّل هو الصفر. فالمصفوفة x التي صرّحنا عنها ,تحتوي على ثلاثة عناصر، دليل العنصر الأوّل هو 0، أمّا دليل العنصر الأخير فهو 2 كما هو واضح.

```
int[] x = new int[3];
x[0] = 1;
x[1] = 2;
x[2] = 3;
//System.Console.Write(x[0] + x[1] + x[2]); // "6"
```

كذلك يمكن اسناد قيم لعناصر المصفوفة بالطريقة التالية :

```
int[] y = new int[] { 1, 2, 3 };
//System.Console.Write(y[0] + y[1] + y[2]); // "6"
```

ملاحظة

يمكن استبعاد الكلمة new ونوع البيانات بشكل اختياري إذا تم التصريح والاسناد عن المصفوفة في نفس الوقت.

```
Int[] z = { 1, 2, 3 };
```

الوصول لعناصر المصفوفة Array Access

بمجرد تهيئة عناصر المصفوفة، يمكن الوصول إليها بواسطة الرجوع إلى فهارس العناصر داخل الأقواس المربعة.

```
System.Console.Write(x[0] + x[1] + x[2]); // "6"
```

المصفوفات متعددة الابعاد Multi-Dimensional Arrays

هناك نوعان:

مصفوفة متساوية الابعاد Rectangular Arrays

مصفوفة غير منتظمة الابعاد Jagged Arrays

مصفوفة متساوية الابعاد Rectangular Arrays

وهي مصفوفة لها نفس عدد الاعمدة والاسطر

```
string[,] x = new string[2, 2];
x[0, 0] = "00"; x[0, 1] = "01";
x[1, 0] = "10"; x[1, 1] = "11";
string[,] y = { { "00", "01" }, { "10", "11" } };
```

مصفوفة غير منتظمة الابعاد Jagged Arrays

وهي عبارة عن مصفوفة عناصرها مصفوفة ، ويمكن أن يكون لها أبعاد غير منتظمة.يتم تخصيص الأبعاد واحدة تلو الأخرى ويمكن للمصفوفة الفرعية تخصيصها بأحجام مختلفة.

```
string[][] a = new string[2][];
a[0] = new string[1]; a[0][0] = "00";
a[1] = new string[2]; a[1][0] = "10"; a[1][1] = "11";
```

من الممكن تعيين القيم أثناء الاسناد.

```
string[][] b = { new string[] {"00"},new string[] {"10","11"} };
```

الشروط Conditionals

تُستخدم التراكيب الشرطية Conditional statements لتنفيذ كتل التعليمات البرمجية المختلفة بناءً على شروط مختلفة.

التركيب إذا If Statement

الشكل الأول لتركيب If Statement

سيتم تنفيذ عبارة if فقط إذا تم تقييم الشرط داخل الأقواس إلى true. يمكن أن يشمل الشرط أيًا من معاملات المقارنة والمنطقية.

```
if ([condition])
{
    statement1;
    statement2;
    ...
}
```

الشكل الثاني لتركيب If Statement

هذا الشكل للتركيب الشرطي `if` يُستخدم عندما نريد الاختيار بين مجموعتين من العبارات البرمجيّة، والشكل العام له:

```
if ([condition])
{
    statement1;
    statement2;
    ...
}
else
{
    Statement3;
    Statement4;
    ...
}
```

إذا تحقق الشرط [condition] عندها تنفّذ التعليمات المجودة بعد `if` مباشرةً، وإلّا (أي لم يتحقق الشرط) يتم تنفيذ التعليمات الموجودة بعد `else` مباشرةً.

الشكل الثالث لتركيب If Statement if

إذا تحقّق الشرط [condition] عندها تنفّذ الحاضنة الموجودة (الأقواس المعقوفة) بعد if مباشرةً.

وإلّا إذا (else if) تحقّق الشرط [condition] يتم تنفيذ الحاضنة الموجودة بعد else if الأولى مباشرةً،

وإلّا (else) يتم تنفيذ الحاضنة الموجودة بعد else مباشرةً

```
if ([condition])
{
    statement1;
    statement2;
    ...
}
else if ([condition1])
{
    Statement3;
    Statement4;
    ...
}
else
{
    Statement5;
    Statement6;
    ...
}
```

ملاحظة:

يستخدم الشكل الثالث في حال وجود عدة كتل من التعليمات ,حيث تنفذ كتلة واحدة فقط.

ويمكن توسيع الكتل بإضافة else if كما هو موضح.

```
if ([condition]){statement1;}
else if ([condition1]){Statement2;}
else if ([condition1]){Statement3;}
else{Statement5;}
```

ملاحظة للاطلاع:

بالإضافة إلى عبارات if و switch ، يوجد معامل التشغيل الثلاثي؟ يمكن أن يستبدل هذا المشغل جملة if-else التي تعطي قيمة إلى متغير معين.إذا تحقق الشرط تأخذ X قيمة 0وإلا تأخذ قيمة 1

```
x = (x < 0.5) ? 0 : 1; // ternary operator (?:)
```

مثال:

```
int x = new System.Random().Next(3); // gives 0, 1 or 2

if (x < 1)
 {
      System.Console.Write(x + " < 1");
 }
else if (x > 1)
 {
      System.Console.Write(x + " > 1");
 }
else
 {
      System.Console.Write(x + " == 1");
 }
```

الشرح:

```
int x = new System.Random().Next(3);
```

السطر السابق يقوم باسناد قيمة عشوائية أما 0 أو 1 أو 2 للمتغير X . عند كل تشغيل للبرنامج سوف تلاحظ نتيجة مختلفة .

يتم اختبار أول شرط وهو x < 1 فاذا كان غير محقق false فإنه ينتقل لاختبار الشرط الثاني وهو x > 1 وهكذا دواليك.حتى يجد شرطاً محققاًعندها ستنفذ كتلة التعليمات الموافقة للشرط المحقق .

ملاحظة:

تم استخدام صف يسمى Random Class يحوي على عدّة طرائق تستطيع عبرها توليد أرقام عشوائية. فالطريقة Next مع بارمتر واحد تستطيع عبرها توليد أرقام عشوائية أكبر أو تساوي الصفر وتنتهي عند الرقم المُحدد ضمن البارمتر.

التركيب Switch Statement

يقدم C# تعليمة Switch كخيار آخر عوضاً عن تعليمة if الشكل الثالث للقيام بعملية اختيار كتلة تعليمات من بين مجموعة الكتل .

```
switch (testExpression)
{
case expressionlist 1: statement1; break;
case expressionlist 2: statement2; break;
default:statement3; break;
}
```

تقوم تعليمة Switch بمقارنة تعبير واحد testExpression مع مجموعة تعابير حيث كل تعبير مرتبط مع Case وعند تطابق قيمة التعبير testExpression مع أحد التعابير expressionlist فإنها ستنفذ كتلة التعليمات المرتبطة مع Case.

وفي حال عدم تطابق التعبير testExpression مع أي قيمة expressionlist فسوف يتم تنفيذ كتل التعليمات الافتراضية default وهي تعليمة اختيارية .

مثال:

```
int x = new System.Random().Next(3); // gives 0, 1 or 2

switch (x)
            {
                case 0: System.Console.Write(x + " is 0"); break;
                case 1: System.Console.Write(x + " is 1"); break;
                default: System.Console.Write(x + " is 2"); break;
            }
```

ملاحظة 1:

التعبير الذي يستخدمه Switch(testExpression) يجب ان يكون من الأنواع التالية:

char

string

bool

an integral value, such as an **int** or a **long**

an **enum** value

ملاحظة 2:

عند تنفيذ switch يتم تنفيذ الكود حتى يصل لتعليمة break ويتوقف البرنامج عندها . في حال لم يكن هناك break فان البرنامج سينفذ التعليمة التالية .

مثال:

```
int i = 1;
 switch (i)
 {
      case 1:
      case 2: Console.WriteLine("One or Two"); break;
      default: Console.WriteLine("Other"); break;
 }
//One or Two
```

الحلقات Loops

هناك أربعة تراكيب للحلقات في C # .تستخدم لتنفيذ كتلة التعليمات البرمجية عدة مرات .كما هو الحال مع العبارة الشرطية if ، يمكن ترك الأقواس المعقوفة للحلقات إذا كان هناك عبارة واحدة فقط في كتلة التعليمات البرمجية.

حلقة التكرار While Loop

تنفذ حلقة while كتلة التعليمات البرمجية فقط إذا كانت حالتها صحيحة وستستمر في التكرار طالما ظلت الحالة صحيحة .لاحظ أنه يتم التحقق من الشرط فقط في بداية كل تكرار (حلقة).

```
int i = 0;
while (i < 10)
{
System.Console.Write(i++); // 0-9
}
```

حلقة التكرار Do-while Loop

تعمل حلقة التكرار بالطريقة نفسها التي تعمل بها حلقة التكرار While Loop ، فيما عدا أنها تتحقق من الحالة بعد كتلة التعليمات ، وبالتالي ستعمل دائمًا عبر كتلة التعليمات مرة واحدة على الأقل .ضع في اعتبارك أن هذه الحلقة تنتهي بفاصلة منقوطة.

```
int j = 0;
do {
System.Console.Write(j++); // 0-9
} while (j < 10);
```

حلقة التكرار For Loop

يتم استخدام حلقة for لتكرار كتلة تعليمات عدد محدد من المرات .وتستخدم ثلاثة بارمترات . البارمتر الأول :يقوم بتهيئة العداد ويتم تنفيذها دائماً مرة واحدة ، قبل الحلقة.

يحتفظ البارمتر الثاني بشرط الحلقة ويتم فحصها قبل كل تكرار .يحتوي البارمتر الثالث على تعليمة زيادة العداد ويتم تنفيذها في نهاية كل تكرار.

```
for (int k = 0; k < 10; k++) {
System.Console.Write(k); // 0-9
}
```

الإجرائيات Methods

الإجرائيات هي كتل من التعليمات البرمجية القابلة لإعادة الاستخدام والتي سيتم تنفيذها فقط عند استدعائها.

تهيئة إجرائية Defining Methods

يمكن إنشاء method داخل Class بكتابة Void متبوعًا باسم method، ومجموعة من الأقواس ، ومن ثم الاقواس المعقوفة وبدخلها تكتب التعليمات البرمجية .إن التعليمة Void تعني أنها لن تُرجع قيمة.

```
class MyApp
{
	void MyPrint()
	{
		System.Console.WriteLine("Hello World");
	}
}
```

استدعاء اجرائية Calling Methods

يتم ذلك بكتابة بإنشاء كائن Object من نفس Class ومن ثم استدعاء الاجرائية منه.

```
class MyApp
{
	static void Main()
	{
		MyApp m = new MyApp();
		m.MyPrint(); // Hello World
	}
	void MyPrint()
	{
		System.Console.WriteLine("Hello World");
	}
}
```

بارمترات الإجرائية Method Parameters

يتم استخدام الأقواس التي تتبع اسم الإجرائية لتمرير الوسائط إلى الإجرائية. للقيام بذلك ، يجب أولاً تهيئة البارمتر .

```
void MyPrint(string s1, string s2)
{
	System.Console.WriteLine(s1 + s2);
}
```

قبل استدعاء الإجرائية تأكد من أن عدد الوسائط ونوعها هو نفسه.

```
void Main()
{
	MyApp m = new MyApp();
	m.MyPrint("Hello", " World"); // "Hello World"
}
```

ارجاع تعبير Return Statement

يمكن للاجرائية إرجاع قيمة .ويتم ذلك باستبدال الكلمة Void بنوع البيانات الذي ستعود به الاجرائية

```
string GetPrint()
{
return "Hello";
}
static void Main()
{
MyApp m = new MyApp();
System.Console.WriteLine(m.GetPrint()); // "Hello World"
}
```