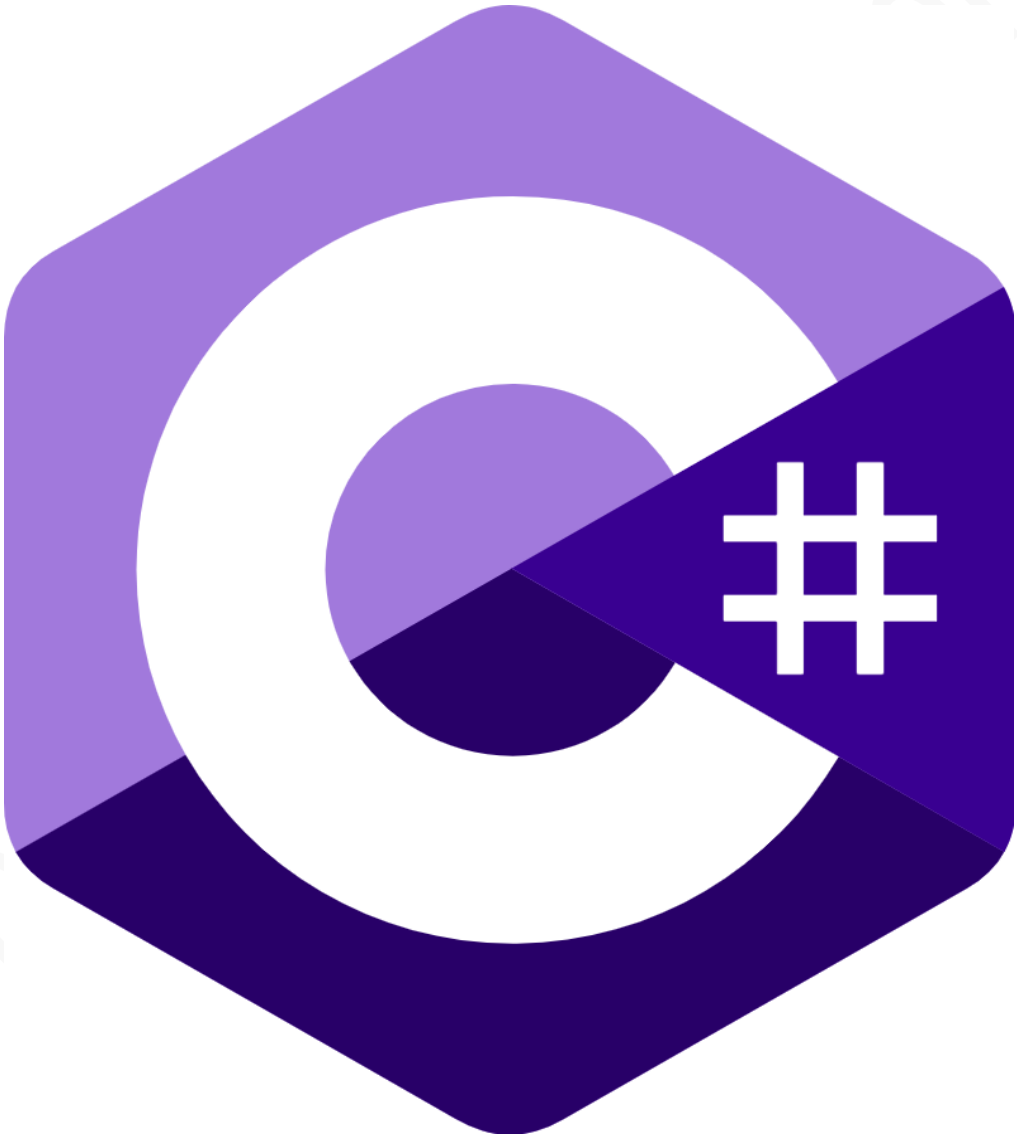
LẬP TRÌNH C# - SOLOLEARN

Student : Green Wolf

Date : 27 / 12 / 2020



1. Basic Concepts

1. What is C#?

Welcome to C#

C# is an elegant object-oriented language that enables developers to build a variety of secure and robust applications that run on the **.NET Framework.**

You can use C# to create Windows applications, Web services, mobile applications, client-server applications, database applications, and much, much more.

You will learn more about these concepts in the upcoming lessons!

C# applications run:

```
on the .NET Framework
```

```
using Java
```

```
only on Linux
```

The .NET Framework

The .NET Framework consists of the **Common Language Runtime (CLR)** and the .NET Framework **class library**.

The **CLR** is the foundation of the .NET Framework. It manages code at execution time, providing core services such as memory management, code accuracy, and many other aspects of your code.

The **class library** is a collection of classes, interfaces, and value types that enable you to accomplish a range of common programming tasks, such as data collection, file access, and working with text.

C# programs use the .NET Framework class library extensively to do common tasks and provide various functionalities.

*These concepts might seem complex, but for now just remember that applications written in C# use the **.NET Framework** and its components.*

Which one is NOT part of the .NET Framework?

```
.Net Framework Class Library
```

```
Operation System
```

```
Common Language Runtime
```

2. Variables

Variables

Programs typically use data to perform tasks.

Creating a **variable** reserves a memory location, or a space in memory, for storing values. It is called **variable** because the information stored in that location can be changed when the program is running.

To use a variable, it must first be declared by specifying the **name** and **data type**.

A variable name, also called an **identifier**, can contain letters, numbers and the underscore character (_) and must start with a letter or underscore.

Although the name of a variable can be any set of letters and numbers, the best identifier is descriptive of the data it will contain. This is very important in order to create clear, understandable and readable code!

*For example, **firstName** and **lastName** are good descriptive variable names, while **abc** and **xyz** are not.*

Which is a valid C# variable name?

```
1Star
```

```
#PersonName#
```

```
Bad_Var
```

Variable Types

A **data type** defines the information that can be stored in a variable, the size of needed memory and the operations that can be performed with the variable.

For example, to store an integer value (a whole number) in a variable, use the **int** keyword:

```
int myAge;
```

The code above declares a variable named **myAge** of type **integer**.

*A line of code that completes an action is called a statement. Each statement in C# must end with a **semicolon ";"**.*

You can assign the value of a variable when you declare it:

```
int myAge = 18;
```

or later in your code:

```
int myAge;
myAge = 18;
```

Remember that you need to declare the variable before using it.

Fill in the blanks to declare a variable named num of type integer and assign 42 to it.

```
int num;
num = 42;
```

Built-in Data Types

There are a number of built-in data types in C#. The most common are:

int - integer.

float - floating point number.

double - double-precision version of float.

char - a single character.

bool - Boolean that can have only one of two values: True or False.

string - a sequence of characters.

The statements below use C# data types:

```
int x = 42;
double pi = 3.14;
char y = 'Z';
bool isOnline = true;
string firstName = "David";
```

*Note that **char** values are assigned using single quotes and **string** values require double quotes.*

You will learn how to perform different operations with variables in the upcoming lessons!

Drag and drop the correct data types from the options below.

```
bool a = false;
double b = 4.22;
string c = "Hi";
int d = 11;
```

3. Your First C# Program

Your First C# Program

You can run, save, and share your C# codes on our **Code Playground**, without installing any additional software.

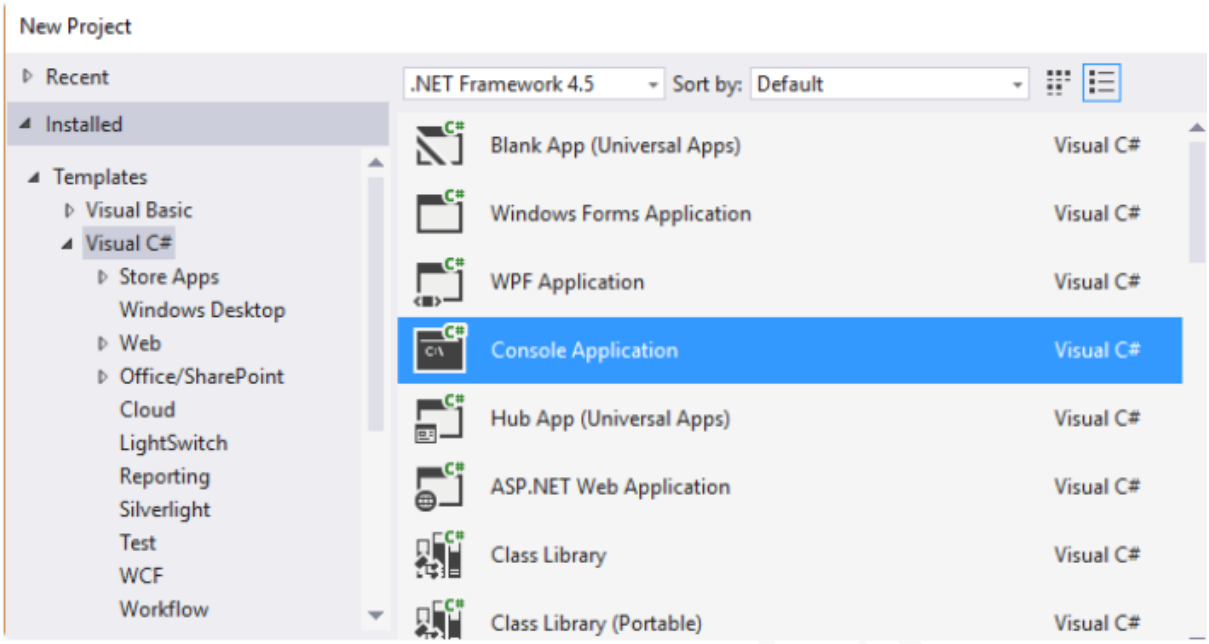
Reference this lesson if you need to install the software on your computer.

To create a C# program, you need to install an integrated development environment (IDE) with coding and debugging tools.

We will be using **Visual Studio Community Edition**, which is available to download for free.

After installing it, choose the default configuration.

Next, click **File->New->Project** and then choose **Console Application** as shown below:



Enter a name for your Project and click OK.

Console application *uses a text-only interface. We chose this type of application to focus on learning the fundamentals of C#.*

What is the name of the IDE used to create C# programs?

- Visual Studio
- 3D Studio
- CStudio
- Visual Maya

Visual Studio will automatically generate some code for your project:

```
using System;
```

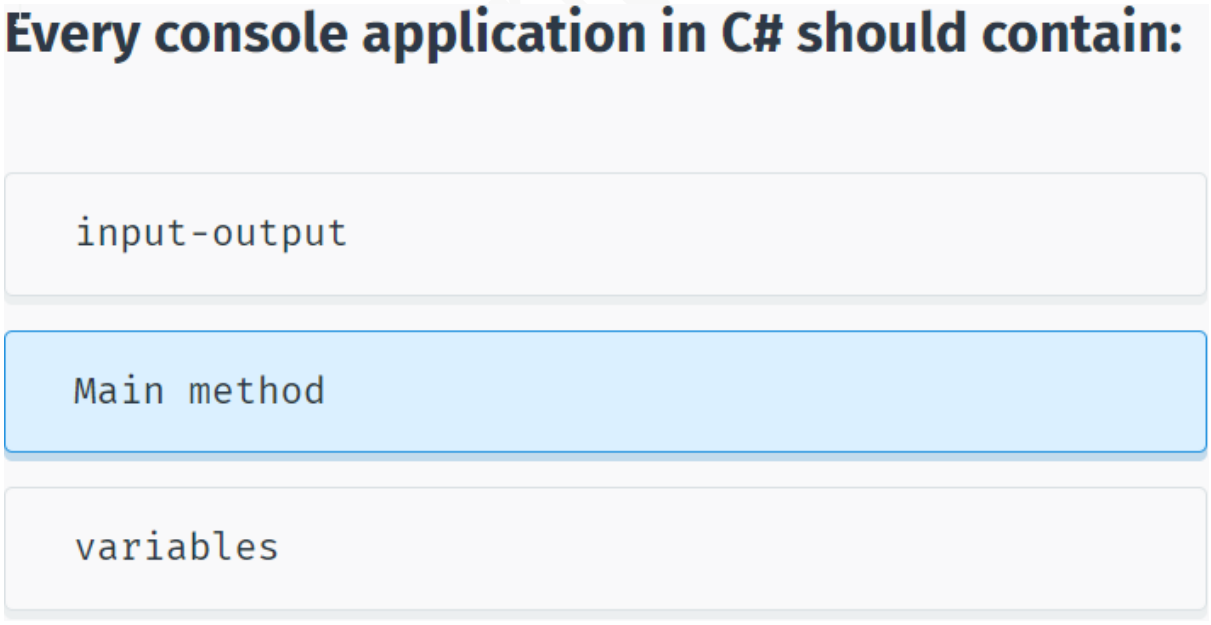
```
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SoloLearn
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

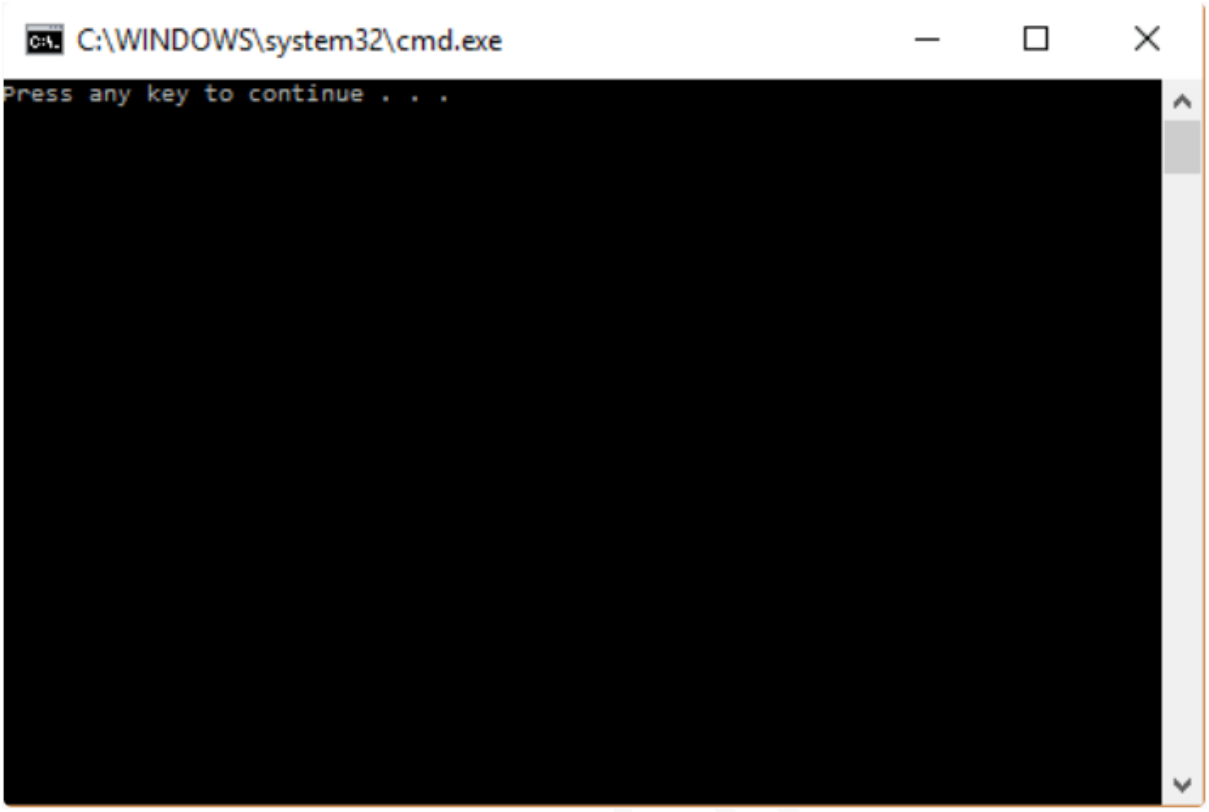
You will learn what each of the statements does in the upcoming lessons.

For now, remember that every C# console application must contain a **method (a function) named Main**. Main is the starting point of every application, i.e. the point where our program starts execution from.

We will learn about classes, methods, arguments, and namespaces in the upcoming lessons.



To run your program, press **Ctrl+F5**. You will see the following screen:



This is a console window. As we did not have any statements in our **Main** method, the program just produces a general message. Pressing any key will close the console.

Congratulations, you just created your first C# program.

Which type of application uses a text-only interface?

- Mobile Application
- Windows Application
- Console Application

4. Printing Text

Displaying Output

Most applications require some **input** from the user and give **output** as a result.

To display text to the console window you use the **Console.Write** or **Console.WriteLine** methods. The difference between these two is that **Console.WriteLine** is followed by a line terminator, which moves the cursor to the next line after the text output.

The program below will display Hello World! to the console window:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SoloLearn
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

*Note the **parentheses** after the **WriteLine** method. This is the way to pass data, or arguments, to methods. In our case **WriteLine** is the method and we pass "Hello World!" to it as an argument. String arguments must be enclosed in quotation marks.*

Drag and drop from the options below to display "Learning C#".

```
Console. WriteLine ( "Learning C#" );
```

We can display variable values to the console window:

```
int x = 89;
Console.WriteLine(x);//89
```

To display a **formatted string**, use the following syntax:

```
int x = 10;
double y = 20;
Console.WriteLine("x = {0}; y = {1}", x, y);
//x = 10; y = 20
```

As you can see, the value of **x** replaced **{0}** and the value of **y** replaced **{1}**.

You can have as many variable placeholders as you need. (i.e.: {3}, {4}, etc.).

What is the output of this code?

```
int a = 4;
int b = 2;
Console.Write(a);
Console.Write(b);
```

42

5. Getting User Input

User Input

You can also prompt the user to enter data and then use the **Console.ReadLine** method to assign the input to a string variable.

The following example asks the user for a name and then displays a message that includes the input:

```
string yourName;
Console.WriteLine("What is your name?");
yourName = Console.ReadLine();//->Green
Console.WriteLine("Hello {0}", yourName);
//What is your name?
//Green
```

The **Console.ReadLine** method waits for user input and then assigns it to the variable. The next statement displays a formatted string containing Hello with the user input. For example, if you enter David, the output will be Hello David.

*Note the empty parentheses in the **ReadLine** method. This means that it does not take any arguments.*

Drag and drop from the options below to take user input and store it in the temp variable:

```
string temp;

temp = [ Console ] . [ ReadLine ] ();
```

The **Console.ReadLine()** method returns a **string** value.

If you are expecting another type of value (such as int or double), the entered data must be converted to that type.

This can be done using the **Convert.ToXXX** methods, where XXX is the .NET name of the type that we want to convert to. For example, methods include **Convert.ToDouble** and **Convert.ToBoolean**.

For integer conversion, there are three alternatives available based on the bit size of the integer: **Convert.ToInt16**, **Convert.ToInt32** and **Convert.ToInt64**. The default int type in C# is 32-bit.

Let's create a program that takes an integer as input and displays it in a message:

```
int age = Convert.ToInt32(Console.ReadLine());
Console.WriteLine("You are {0} years old", age);
```

*If, in the program above, a non-integer value is entered (for example, letters), the **Convert** will fail and cause an error.*

Drag and drop from the options below to make the following program work:

```
double n;
string x = "77";
n = [ Convert ].[ ToDouble ](x);
```

6. Comments

Comments

Comments are explanatory statements that you can include in a program to benefit the reader of your code.

The compiler ignores everything that appears in the comment, so none of that information affects the result.

A comment beginning with two slashes (//) is called a single-line comment. The slashes tell the compiler to ignore everything that follows, until the end of the line.

```
// Prints Hello
Console.WriteLine("Hello");
```

When you run this code, Hello will be displayed to the screen. The // Prints Hello line is a comment and will not appear as output.

What is the output of this code?

```
int x = 8;
// x = 3;
Console.WriteLine(x);
```

- error
- 8
- 3

Multi-Line Comments

Comments that require multiple lines begin with /* and end with */ at the end of the comment block.

You can place them on the same line or insert one or more lines between them.

```
/* Some long
comment text
*/
int x = 42;
Console.WriteLine(x);//42
```

Adding comments to your code is good programming practice. It facilitates a clear understanding of the code for you and for others who read it.

Fill in the blanks to make the text a comment.

```
   /*   Declaring an integer
     and printing it   */
int x = 42;
Console.WriteLine(x);
```

7. The var Keyword

The var Keyword

A variable can be explicitly declared with its **type** before it is used.

Alternatively, C# provides a handy function to enable the compiler to determine the type of the variable automatically based on the expression it is assigned to.

The var keyword is used for those scenarios:

```
var num = 15;
```

The code above makes the compiler determine the type of the variable. Since the value assigned to the variable is an integer, the variable will be declared as an integer automatically.

What is the type of the temp variable?
var temp = 14.55;

```
Double
```

Variables declared using the var keyword are called **implicitly typed** variables.

Implicitly typed variables must be initialized with a value.

For example, the following program will cause an error:

```
var num;
num = 42;->error!
```

*Although it is easy and convenient to declare variables using the **var** keyword, overuse can harm the readability of your code. Best practice is to explicitly declare variables.*

What is the output of this code?

```
var n1;
n1 = true;
Console.WriteLine(n1);
```

1

true

error

8. Constants

Constants

Constants store a value that cannot be changed from their initial assignment.

To declare a constant, use the **const** modifier.

For example:

```
const double PI = 3.14;
```

The value of const PI cannot be changed during program execution.

For example, an assignment statement later in the program will cause an error:

```
const double PI = 3.14;
PI = 8; ->error
```

*Constants **must** be initialized with a value when declared.*

Fill in the blank to make the variable num a constant.

```
const int num=2;
```

9. Arithmetic Operators

Operators

An **operator** is a symbol that performs mathematical or logical manipulations.

Arithmetic Operators

C# supports the following arithmetic operators:

Operator	Symbol	Form
Addition	+	x + y
Subtraction	-	x - y
Multiplication	*	x * y
Division	/	x / y
Modulus	%	x % y

For example:

```
int x = 10;
int y = 4;
Console.WriteLine(x-y);//6
```

*Tap **Try It Yourself** to play around with the code!*

Fill in the blanks to display the result of the multiplication of x and y.

```
int x = 42;

int y = 7;

int z = x  *  y;

Console.WriteLine(  z  );
```

Division

The division operator (/) divides the first operand by the second. If the operands are both integers, any remainder is dropped in order to return an integer value.

Example:

```
int x = 10 / 4;
Console.WriteLine(x);//2
```

Division by 0 is undefined and will crash your program.

What is the output of this code?

```
int x = 16;
int y = 5;
Console.WriteLine(x/y);
```

3

Modulus

The modulus operator (%) is informally known as the remainder operator because it returns the remainder of an integer division.

For example:

```
int x = 25 % 7;
Console.WriteLine(x);//4
```

Which operator is used to determine the remainder?

+

*

%

Operator Precedence

Operator **precedence** determines the grouping of terms in an expression, which affects how an expression is evaluated. Certain operators take higher precedence over others; for example, the multiplication operator has higher precedence than the addition operator.

For example:

```
int x = 4+3*2;
Console.WriteLine(x);//10
```

The program evaluates 3*2 first, and then adds the result to 4.

As in mathematics, using **parentheses** alters operator precedence.

```
int x = (4 + 3) *2;
Console.WriteLine(x);//14
```

The operations within parentheses are performed first. If there are parenthetical expressions nested within one another, the expression within the innermost parentheses is evaluated first.

If none of the expressions are in parentheses, multiplicative (multiplication, division, modulus) operators will be evaluated before additive (addition, subtraction) operators. Operators of equal precedence are evaluated from left to right.

Fill in the missing parentheses to have x equal 15.

```
int x =    (    2 + 3    )    * 3;

Console.WriteLine(x);
```

10. Assignment & Increment Operators

Assignment Operators

The = **assignment** operator assigns the value on the right side of the operator to the variable on the left side.

C# also provides **compound assignment operators** that perform an operation and an assignment in one statement.

For example:

```
int x = 42;
x += 2; // equivalent to x = x + 2
Console.WriteLine(x);//44
x -= 6; // equivalent to x = x - 6
Console.WriteLine(x);//38
```

What is the alternative for x = x + 5?

```
x = y + 5;
```

```
x -= 4;
```

```
x += 5;
```

The same shorthand syntax applies to the multiplication, division, and modulus operators.

```
x *= 8; // equivalent to x = x * 8
x /= 5; // equivalent to x = x / 5
x %= 2; // equivalent to x = x % 2
```

The same shorthand syntax applies to the multiplication, division, and modulus operators.

Fill in the missing part of the following code to divide x by 3 using the shorthand division operator.

```
int x = 42;

x [ / ] = [ 3 ] ;
```

Increment Operator

The **increment** operator is used to increase an integer's value by one, and is a commonly used C# operator.

```
x++; //equivalent to x = x + 1
```

For example:

```
int x = 10;
x++;
Console.WriteLine(x);//11
```

The increment operator is used to increase an integer's value by one.

What is the output of this code?

```
int x = 6;
x++;
Console.WriteLine(x);
```

7

Prefix & Postfix Forms

The increment operator has two forms, **prefix** and **postfix**

```
++x; //prefix
x++; //postfix
```

Prefix increments the value, and then proceeds with the expression.

Postfix evaluates the expression and then performs the incrementing.

Prefix example:

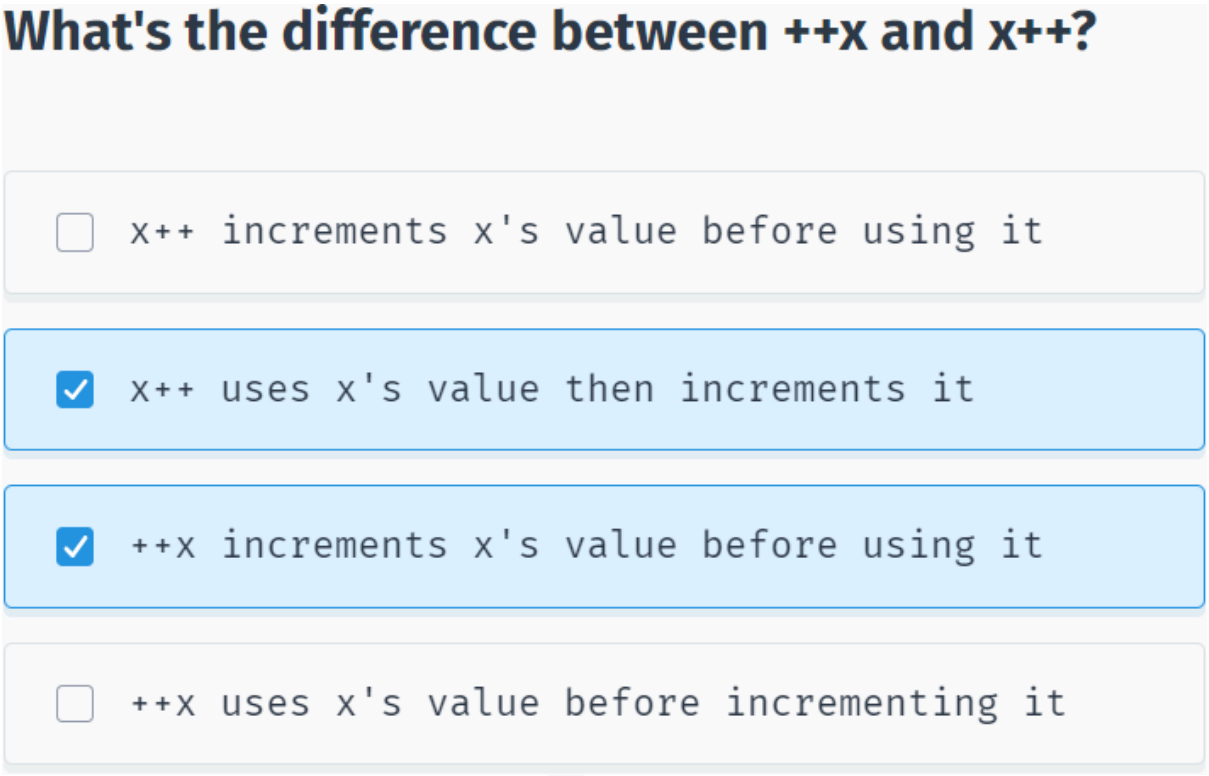
```
int x = 3;
int y = ++x;
//x is 4; y is 4
```

Postfix example:

```
int x = 3;
int y = x++;
//x is 4; y is 3
```

*The **prefix** example increments the value of x, and then assigns it to y.*

*The **postfix** example assigns the value of x to y, and then increments x.*



Decrement Operator

The **decrement** operator (--) works in much the same way as the increment operator, but instead of increasing the value, it decreases it by one.

```
--x; // prefix
x--; // postfix
```

The decrement operator (--) works in much the same way as the increment operator.

Fill in the missing operator to decrease the value of x by one.

```
int x = 42;

x --;

Console.WriteLine(x);
```

11. Module 1 Quiz

Rearrange the code to create a valid program.

```
static void Main(string[] args) {
```

```
string msg = "Hello";
```

```
Console.WriteLine(msg);
```

```
}
```

Drag and drop from the options below to display the value of x to the screen.

```
static void [ Main ](string[] args)
{
    int x = (4 + 3) * 2;
    [ Console ].[ WriteLine ](x);
}
```

What is the output of this code?

```
int a = 4;
int b = 6;
b = a++;
Console.WriteLine(++b);
```

5

Fill in the blanks to declare two variables of type int and display their sum to the screen.

```
int x = 8;

    int     y = 15;

Console.WriteLine(x  +  y);
```

What is the output of this code?

```
int x = 15;
int y = 6;
x %= y;
Console.WriteLine(x);
```

```
3
```

2. Conditionals and Loops

1. The if-else Statement

The if Statement

The **if** statement is a conditional statement that executes a block of code when a condition is true.

The general form of the if statement is:

```
if (condition)
{
    // Execute this code when condition is true
}
```

The condition can be any expression that returns true or false.

For example:

```
int x = 8;
int y = 3;
if (x > y)
{
     Console.WriteLine("x is greater than y");
     //x is greater than y
}
```

The code above will evaluate the condition **x > y**. If it is true, the code inside the if block will execute.

When only one line of code is in the if block, the curly braces can be omitted.

For example:

```
if (x > y)
Console.WriteLine("x is greater than y");
```

Fill in the blanks to display Welcome to the screen when age is greater than 16.

```
int age = 24;

    if (age  >  16)

{

    Console.WriteLine("Welcome");

}
```

Relational Operators

Use **relational operators** to evaluate conditions. In addition to the less than (<) and greater than (>) operators, the following operators are available:

Operator	Description	Example	
>=	Greater than or equal to	7 >= 4	True
<=	Less than or equal to	7 <= 4	False
==	Equal to	7 == 4	False
!=	Not equal to	7 != 4	True

Example:

```
int a=7, b=7;
if (a == b) {
     Console.WriteLine("Equal");
     //Equal
}
```

Which is the correct operator for equality testing?

- ==
- >=
- !=
- <=

The else Clause

An optional **else** clause can be specified to execute a block of code when the condition in the **if** statement evaluates to **false**.

Syntax:

```
if (condition)
{
   //statements
}
else
{
   //statements
}
```

For example:

```
int mark = 85;
if (mark < 50)
{
     Console.WriteLine("You failed.");
}
else
```

```
{
    Console.WriteLine("You passed.");
}
```

Fill in the blanks to find the larger of two variables.

```
int a = 42;
int b = 88;
if (a > b)
{
   Console.WriteLine(a);
}
else
{
   Console.WriteLine( b );
}
```

Nested if Statements

You can also include, or **nest**, if statements within another if statement.

For example:

```
int mark = 100;
if (mark >= 50) {
```

```
      Console.WriteLine("You passed.");
      if (mark == 100) {
          Console.WriteLine("Perfect!");
      }
else {
      Console.WriteLine("You failed.");
}
```

You can nest an unlimited number of if-else statements.

For example:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SoloLearn
{
    class Program
    {
        static void Main(string[] args)
        {
            int age = 17;
            if (age > 14) {
                if(age > 18) {
                    Console.WriteLine("Adult");
                }
                else {
                    Console.WriteLine("Teenager");
                }
            }
            else {
                if (age > 0) {
                    Console.WriteLine("Child");
                }
                else {
                    Console.WriteLine("Something's wrong");
                }
            }
        }
```

```
    }
}
```

*Remember that all **else** clauses must have corresponding **if** statements.*

What is the output of this code?

```
int a = 8;
int b = ++a;
if(a > 5)
  b -= 3;
else
  b = 9;
Console.WriteLine(b);
```

6

The if-else if Statement

The **if-else if** statement can be used to decide among three or more actions.

For example:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SoloLearn
{
    class Program
    {
        static void Main(string[] args)
        {
            int x = 33;
```

```
            if (x == 8) {
                Console.WriteLine("Value of x is 8");
            }
            else if (x == 18) {
                Console.WriteLine("Value of x is 18");
            }
            else if (x == 33) {
                Console.WriteLine("Value of x is 33");
            }
            else {
                Console.WriteLine("No match");
            }
        }
    }
}
```

*Remember, that an **if** can have zero or more **else if**'s and they must come before the last **else**, which is optional.*

*Once an **else if** succeeds, none of the remaining **else if**'s or **else** clause will be tested.*

How many nested if statements can an if statement contain?

- Only two
- As many as you want
- None

2. The switch Statement

switch

The **switch** statement provides a more elegant way to test a variable for equality against a list of values.

Each value is called a **case**, and the variable being switched on is checked for each switch case.

For example:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SoloLearn
{
    class Program
    {
        static void Main(string[] args)
        {
            int num = 3;
            switch (num)
            {
                case 1:
                    Console.WriteLine("one");
                    break;
                case 2:
                    Console.WriteLine("two");
                    break;
                case 3:
                    Console.WriteLine("three");
                    break;
            }
        }
    }
}
```

Each **case** represents a value to be checked, followed by a colon, and the statements to get executed if that case is matched.

*A **switch** statement can include any number of **cases**. However, no two case labels may contain the same constant value.*

*The **break**; statement that ends each **case** will be covered shortly.*

Fill in the blanks to create a valid switch statement.

```
int x = 33;

  switch  (x) {

    case 8:

      Console.WriteLine("Value is 8");

      break;

    case 18  :

      Console.WriteLine("Value is 18");

      break;

     case  33:

      Console.WriteLine("Value is 33");

      break;

}
```

The default Case

In a switch statement, the optional **default** case is executed when none of the previous cases match.

Example:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SoloLearn
{
    class Program
    {
        static void Main(string[] args)
        {
            int age = 88;
            switch (age) {
                case 16:
                    Console.WriteLine("Too young");
                    break;
                case 42:
                    Console.WriteLine("Adult");
                    break;
                case 70:
                    Console.WriteLine("Senior");
                    break;
                default:
                    Console.WriteLine("The default case");
                    break;
            }
        }
    }
}
```

*The **default** code executes when none of the cases matches the switch expression.*

Drag and drop from the options below to test the x variable with a switch statement.

```
switch (x) {
  case 10:
    Console.WriteLine("Ten");
    break;
  case 20:
    Console.WriteLine("Twenty");
    break;
  default:
    Console.WriteLine("No match");.
    break;
}
```

The break Statement

The role of the **break** statement is to terminate the **switch** statement.

Without it, execution continues past the matching **case** statements and falls through to the next case statements, even when the case labels don't match the switch variable.

This behavior is called **fallthrough** and modern C# compilers will not compile such code. All case and default code must end with a **break** statement.

*The **break** statement can also be used to break out of a loop. You will learn about loops in the coming lessons.*

What would occur if we forget to include a break statement at the end of case code?

```
nothing
```

```
"break" will be printed
```

```
compile error
```

3. The while Loop

3. Methods
4. Classes & Objects
5. Arrays & Strings
6. More On Classes
7. Inheritance & Polymorphism
8. Structs, Enums, Exceptions & Files
9. Generics
Dictionary

Noun

basic concepts : những khái niệm cơ bản

language : ngôn ngữ

applications : những ứng dụng

Web services : dịch vụ web
client-server : máy khách-máy chủ
database : cơ sở dữ liệu
.NET Framework : là một nền tảng lập trình
Common Language Runtime : ngôn ngữ thực thi tổng quát
class library : lớp thư viện
foundation : nền tảng
memory management : quản lý bộ nhớ
collection : bộ sưu tập
aspects of code : các khía cạnh của code
execution time : thời gian thực hiện
accuracy : sự chính xác
core services : những dịch vụ cốt lõi
task : nhiệm vụ
various functionalities : các chức năng khác nhau
components : các thành phần
variables : biến (ví dụ $f(x) = x^2$ ->x chính là biến trong hàm)
memory location : vị trí bộ nhớ
name : tên
data type : kiểu dữ liệu
underscore character : '_' dấu gạch dưới
semicolon : ';' dấu chấm phẩy
single quotes : " ' " dấu nháy đơn
parentheses : '(' dấu ngoặc đơn
quotation marks : ' " ' dấu ngoặc kép
two slashes : '//' 2 dấu xẹt
information : thông tin
identifier : sự định danh
statement : câu lệnh
operation : sự điều hành
sequence : sự liên tục
character : chữ cái

additional software : phần mềm bổ sung
classes : các lớp
methods : các phương thức
arguments : các đối số
namespaces : không gian tên
general message : thông báo chung
text-only interface : giao diện thuần văn bản
a line terminator : dấu xuống dòng
console : bàn điều khiển
cursor : con trỏ
formatted string : chuỗi định dạng
syntax : cú pháp
conversion : sự chuyển đổi
alternatives : lựa chọn thay thế
default : mặc định
precedence : quyền ưu tiên
Operator : hệ điều hành
compound assignment operators : toán tử gán ghép
Prefix : tiền tố
Postfix : hậu tố
nested if Statements : lệnh if lồng nhau
equality : sự bằng nhau

Verb

enable : cho phép, kích hoạt
build : xây dựng
create : chế tạo
run : chạy
consists of : bao gồm
collect : sưu tầm
accomplish : đạt được
file access : truy cập file

reserves : dự trữ
store : lưu trữ
declare : khai báo
specify : xác định
complete : hoàn thành
descriptive of : mô tả về
contain : lưu trữ
perform : biểu diễn
install : cài đặt
Pressing any key : nhấn phím bất kỳ
pass data : truyền dữ liệu
display : trưng bày
include : bao gồm
prompt : nhắc nhở
assign : chỉ định
convert : chuyển đổi
expect : mong đợi
follow : theo
facilitate : tạo điều kiện
alter : thay đổi
evaluate : đánh giá

Adjective

elegant : thanh lịch
variety of : đa dạng
secure : an toàn
robust : mạnh mẽ
upcoming : sắp tới
complex : phức tạp
important : quan trọng
understandable : có thể hiểu được
readable : có thể đọc được

True : đúng

False : sai

different : khác nhau

empty : bỏ trống

explanatory : được giải thích

Other

such as : như là

extensively : một cách chuyên sâu

automatically : một cách tự động

implicitly : ngầm hiểu

explicitly : rõ ràng

END!