Lập Trình C# - SoloLearn.com

**Student: GreenWolf Date: 23/12/2020**

# Basic Concepts

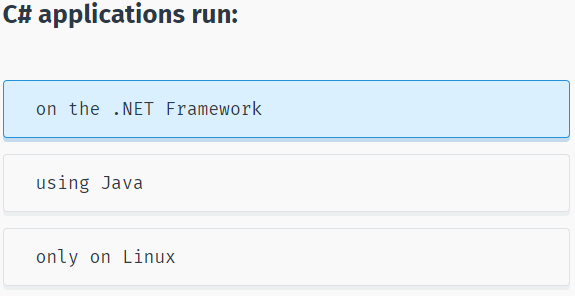
## What is C#?

**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 th*es*e concepts in the upcoming lessons!



**The .NET Framework**

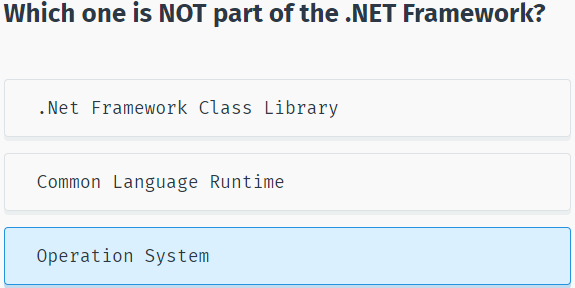
The .NET Framework consists of the **Common Language Runtime (CLR)** and the .NET Framework **class library**.

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.



## Variables

**Variables**

Programs typically use data to perform tasks.

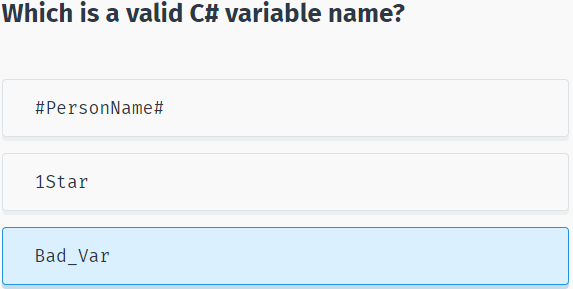
Creating a **variable** reserves a memory location, or a space in memory, for storing values. It is called **variable** because the information stored in that location can be changed when the program is running.

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.



**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:

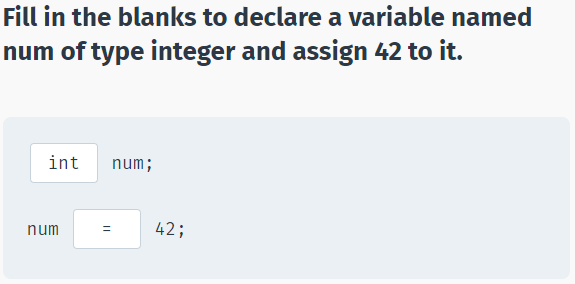
int myAge = 18;

or later in your code:

int myAge;

myAge = 18;

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



**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'

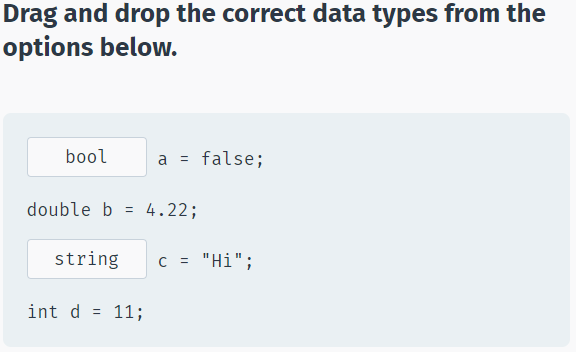
bool ​isOnline = true;

string ​firstName = “David”;

string secondName = “Nguyễn Lê An”

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!



## 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.

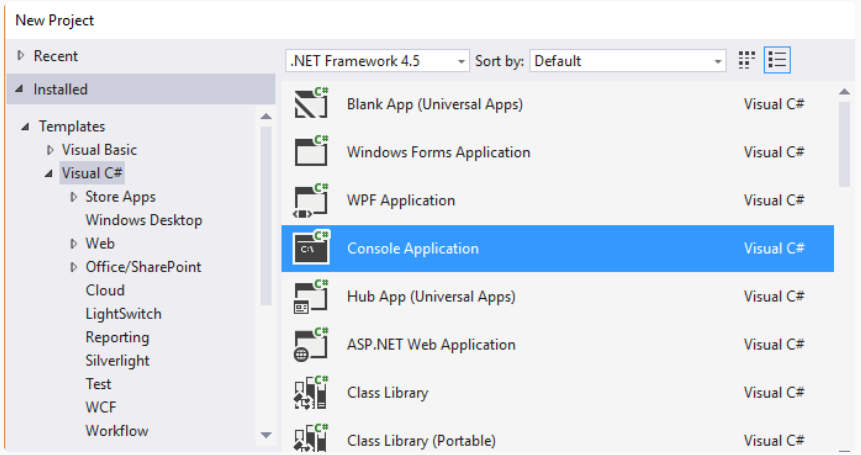
Reference this lesson if you need to install the software on your computer.

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.

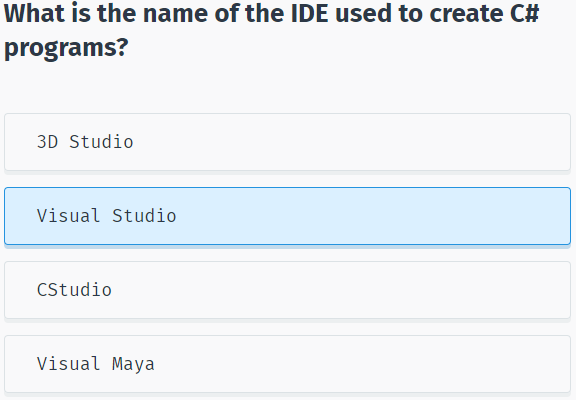
After installing it, choose the default configuration.

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#.



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)

{

}

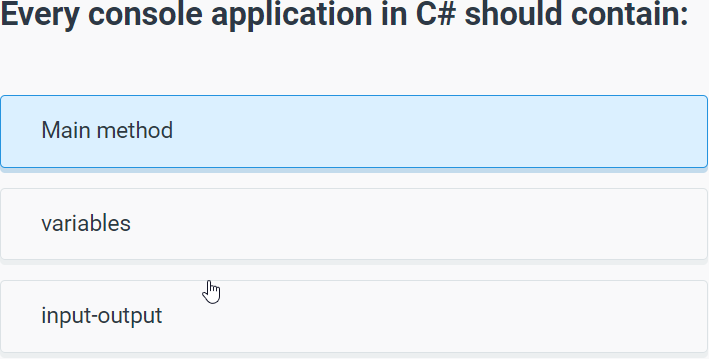
}

}

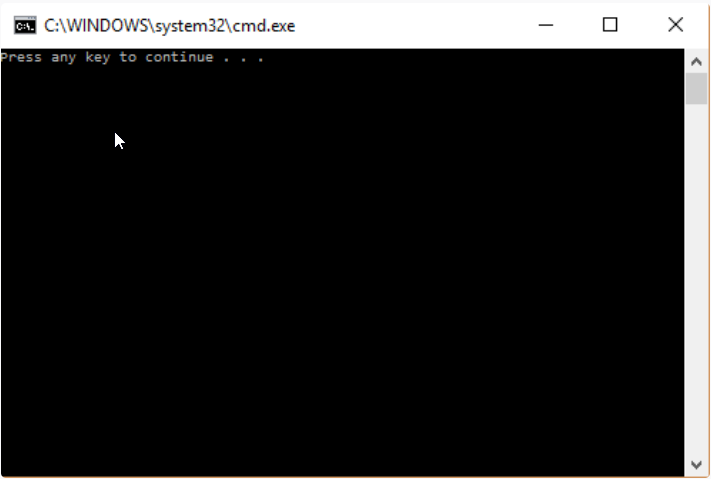
You will learn what each of the statements does in the upcoming lessons.

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.

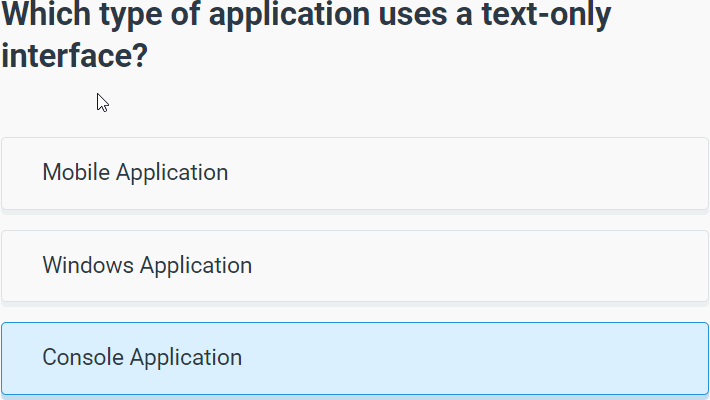


To run your program, press **Ctrl+F5**. You will see the following screen:



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.



## 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)

{

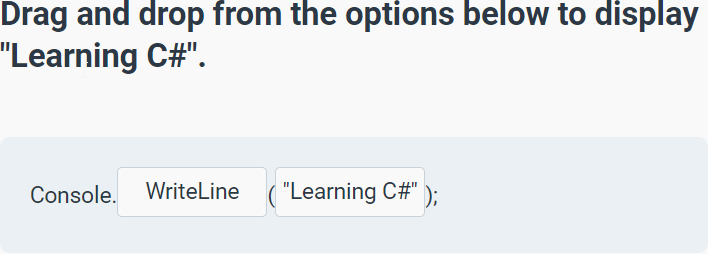
Console.WriteLine("Hello World!");

}

}

}

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.



We can display variable values 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)

{

int x = 89;

Console.WriteLine(x);

}

}

}

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

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 = 10;

double y = 20;

Console.WriteLine("x = {0}; y = {1}", x, y);

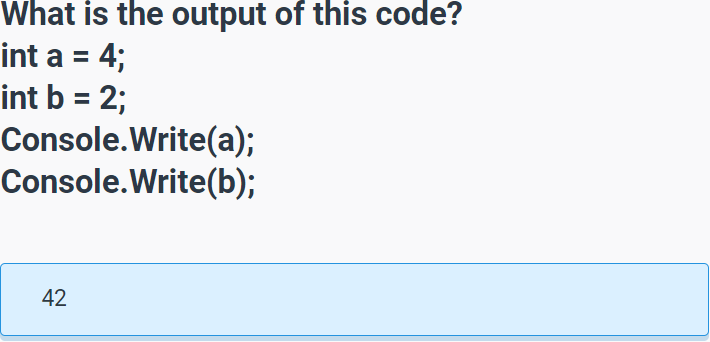
}

}

}

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.).



## 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:

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.OutputEncoding = System.Text.Encoding.Unicode;

Console.InputEncoding = System.Text.Encoding.Unicode;

string yourName;

Console.WriteLine("What is your name?");

yourName = Console.ReadLine();

Console.WriteLine("Hello {0}", yourName);

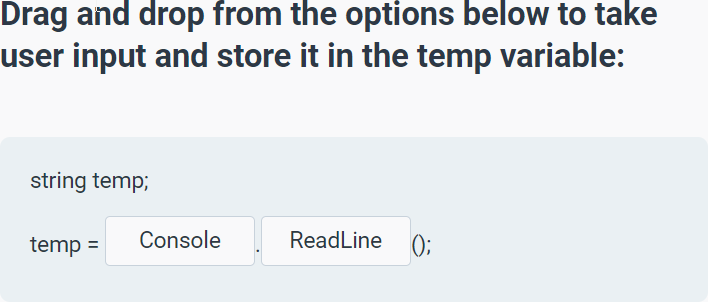
}

}

}

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.



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:

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.OutputEncoding = System.Text.Encoding.Unicode;

Console.InputEncoding = System.Text.Encoding.Unicode;

string yourName;

Console.Write("Xin nhập tên bạn: ");

yourName = Console.ReadLine();

Console.Write("Xin nhập tuổi của bạn: ");

uint age = 0;

while (!uint.TryParse(Console.ReadLine(), out age))

{

Console.Write("Bạn nhập sai - thử lại: ");

}

Console.WriteLine("Xin chào {0} {1} tuổi",

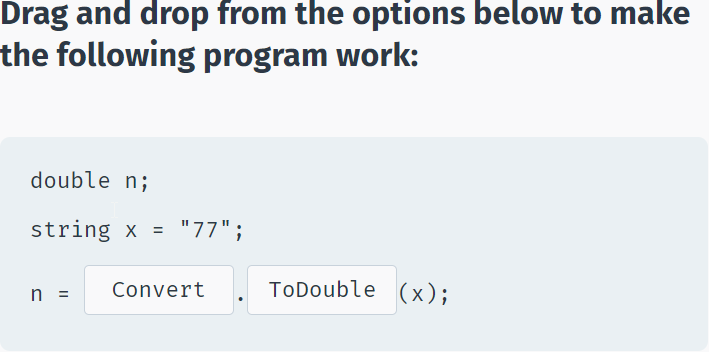
yourName, age);

}

}

}

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



## 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.

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)

{

// Prints Hello

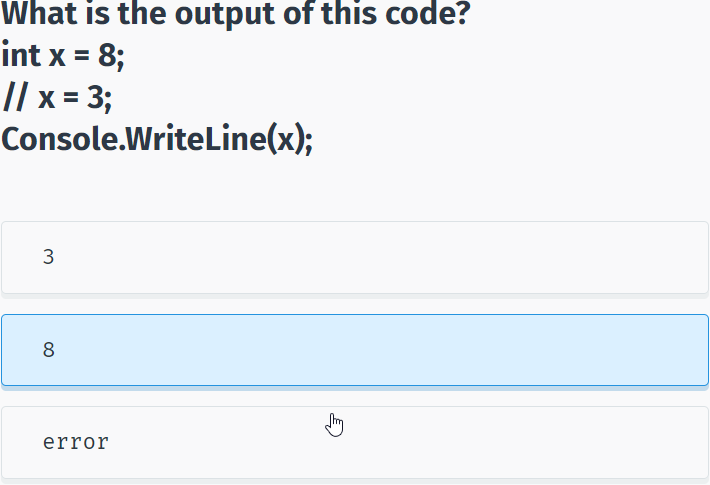
Console.WriteLine("Hello");

}

}

}

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.



**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.

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)

{

/\* Some long

comment text

\*/

int x = 42;

Console.WriteLine(x);

}

}

}

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.

# Conditionals and Loops

# Methods

# Classes & Objects

# Arrays & Strings

# More On Classes

# Inheritance & Polymorphism

# Inheritance & Polymorphism

# Generics

# Dictionary

Basic Concepts: Các khái niệm cơ bản

data types: Các kiểu dữ liệu

arrays : Mảng

pointers: Con trỏ

conditional statements: Câu lệnh điều kiện

loops: Vòng lặp

functions: Hàm

classes: lớp

objects: đối tượng

inheritance: Kế thừa

polymorphism: Đa hình

Basic Arithmetic: Số Học cơ bản

Assignment: Phân Công

Increment: Gia Tăng

decrement: giảm đi

cross-platform language: Ngôn ngữ đa nền tảng

popular: phổ biến

high-performance: Hiệu suất cao

applications: ứng dụng

operating systems: hệ điều hành

browsers: trình duyệt

video-games: trò chơi điện tử

programming language: ngôn ngữ lập trình

General purpose: Mục đích chung

template: bản mẫu

commands: câu lệnh

statements: Mệnh đề

irrespective of: ko phân biệt (bất kể)

entry point: Đầu vào

input: Nhập Liệu đầu vào

output: xuất hình đầu ra

streams: luồng xủ lý

combination: sự kết hợp

standard: tiêu chuẩn thông thường

display screen: màn hình hiển thị

terminate: chấm dứt

instruction: chỉ dẫn

offer: cung cấp

various headers: tiêu đề khác nhau

work properly: hoạt động đúng cách

namespace:không gian tên

declarative region: vùng khai báo

features: các tính chất

insert: thêm vào

Variables: Biến

memory location: vị trí bộ nhớ

reserves: dự trữ

declare: khai báo

syntax error: lỗi cú pháp

Specify: chỉ định

Addition: phép cộng

Subtraction: Phép Trừ

Multiplication: Phép Nhân

Division: Phép Chia lấy nguyên

Modulus: Phép Chia Lấy Dư

Parentheses: dấu ngoặc đơn

innermost: trong cùng

evaluate: đánh giá

expressions nested: biểu thức lồng nhau

quadratic equation: Phương Trình Bậc 2

intended to: mục đích

Expressions: biểu thức

legal: hợp pháp

illegal: bất hợp pháp

positive numbers: số dương

Case-Sensitivity: phân biệt chữ hoa chữ thường

meaningful: có ý nghĩa

practical: thiết thực

occasionally: thỉnh thoảng

necessary: cần thiết

elements: nhiều phần tử

rows: hàng

columns: cột

ampersand: dấu &

prototypes: nguyên mẫu

declaration: khai báo

arguments: đối số

parameters: tham số

result: kết quả

Overloading: quá tải

Reference: tham chiếu

identity: bản sắc, danh tính...

attributes: thuộc tính

behavior: hành vi

In regard to: liên quan đến

instantiation: sự tức thời (sự khởi tạo đối tượng )

Abstraction: Tính trừu tượng

.NET Framework: Môi trường làm việc với .net

Descriptive: mô tả

Fundamentals: nguyên tắc cơ bản

Explanatory: giải thích

# End!