C# Get Started

❮ PreviousNext ❯

C# IDE

The easiest way to get started with C#, is to use an IDE.

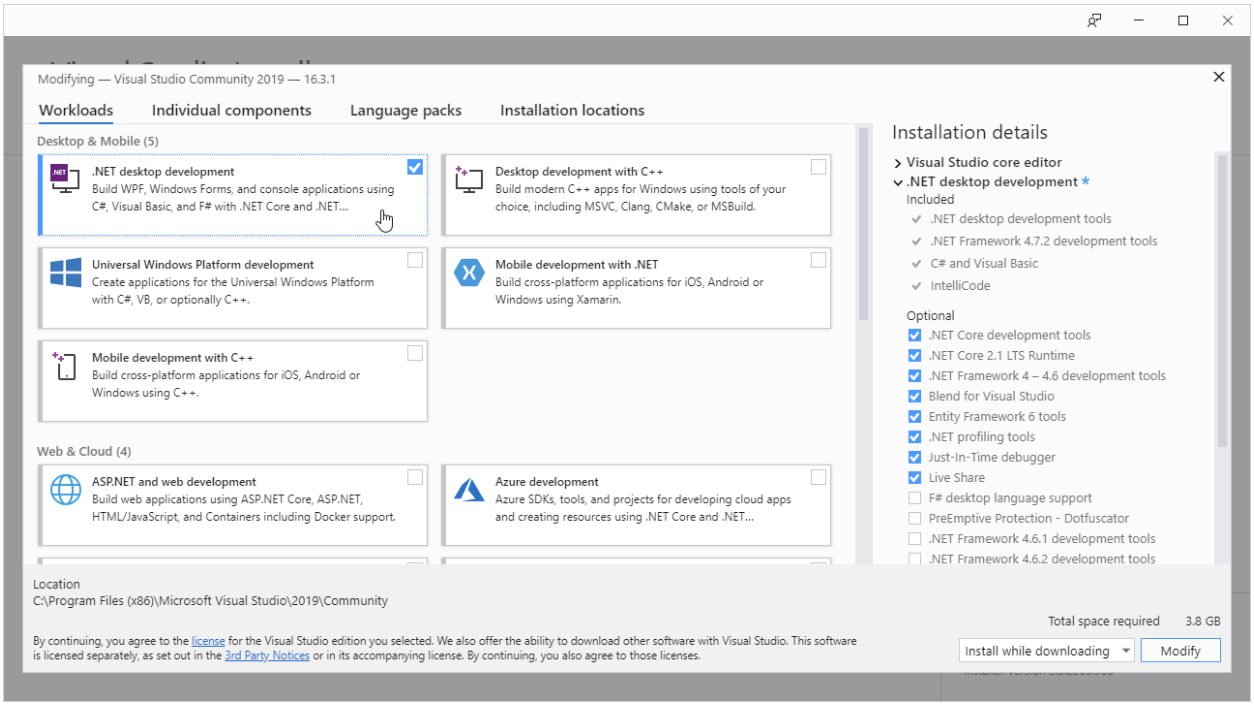
An IDE (Integrated Development Environment) is used to edit and compile code.

In our tutorial, we will use Visual Studio Community, which is free to download from https://visualstudio.microsoft.com/vs/community/.

Applications written in C# use the .NET Framework, so it makes sense to use Visual Studio, as the program, the framework, and the language, are all created by Microsoft.

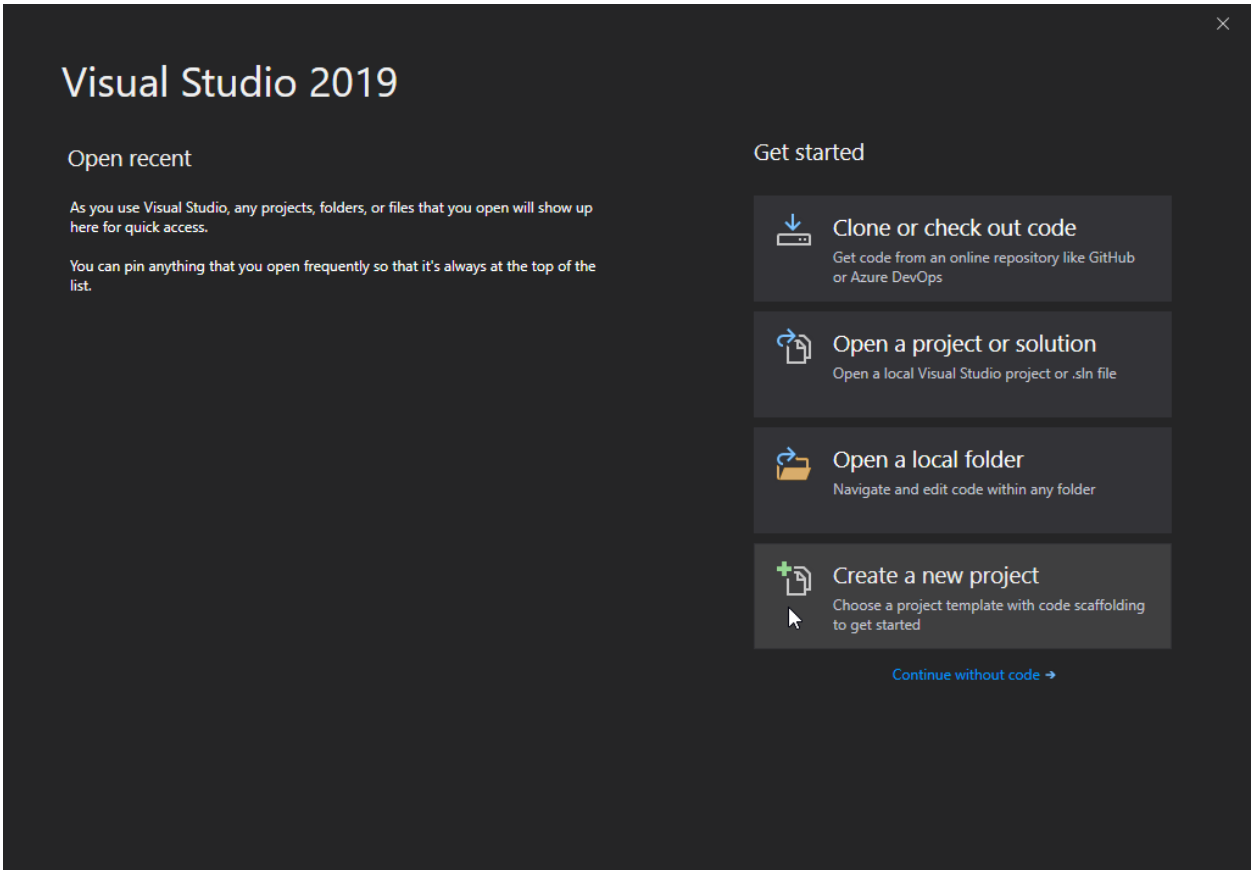
C# Install

Once the Visual Studio Installer is downloaded and installed, choose the .NET workload and click on the **Modify/Install** button:

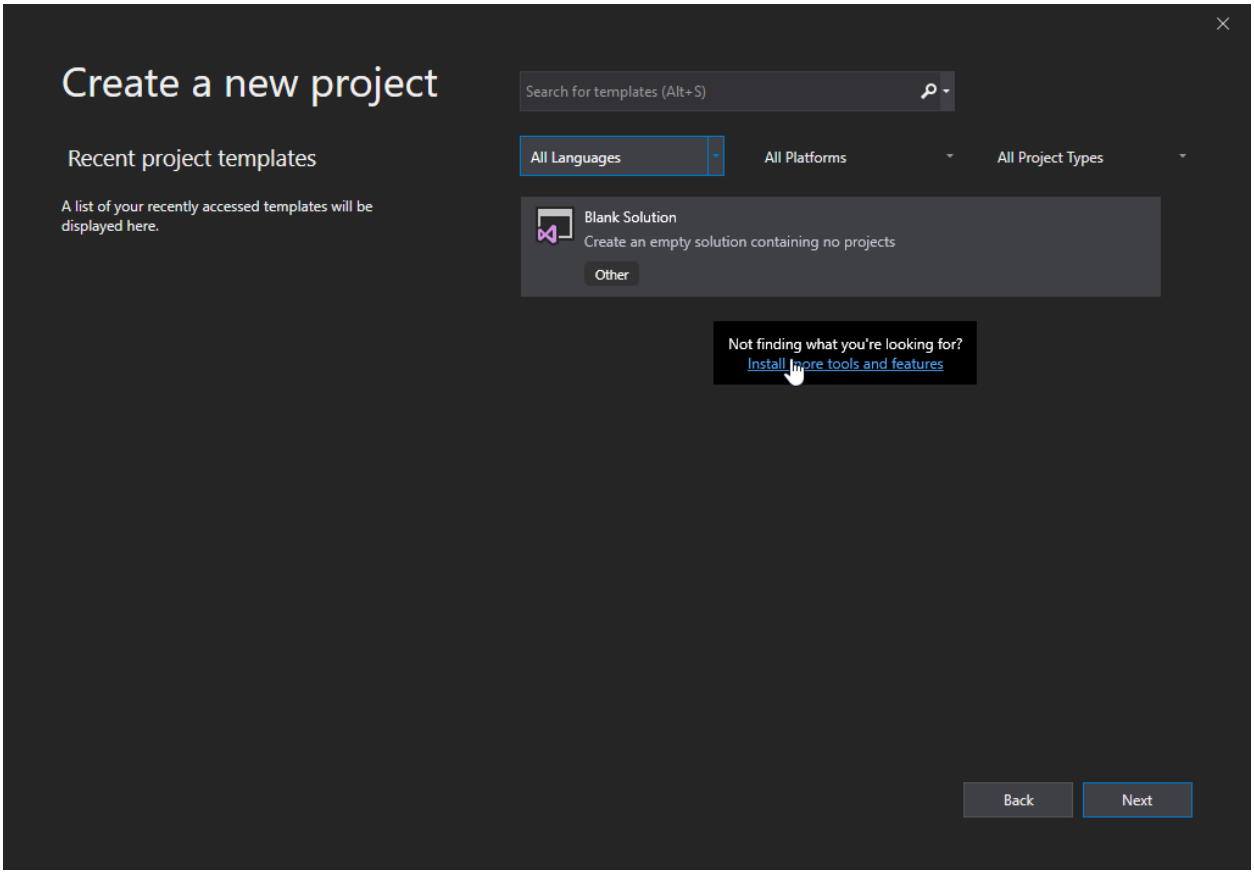


After the installation is complete, click on the **Launch** button to get started with Visual Studio.

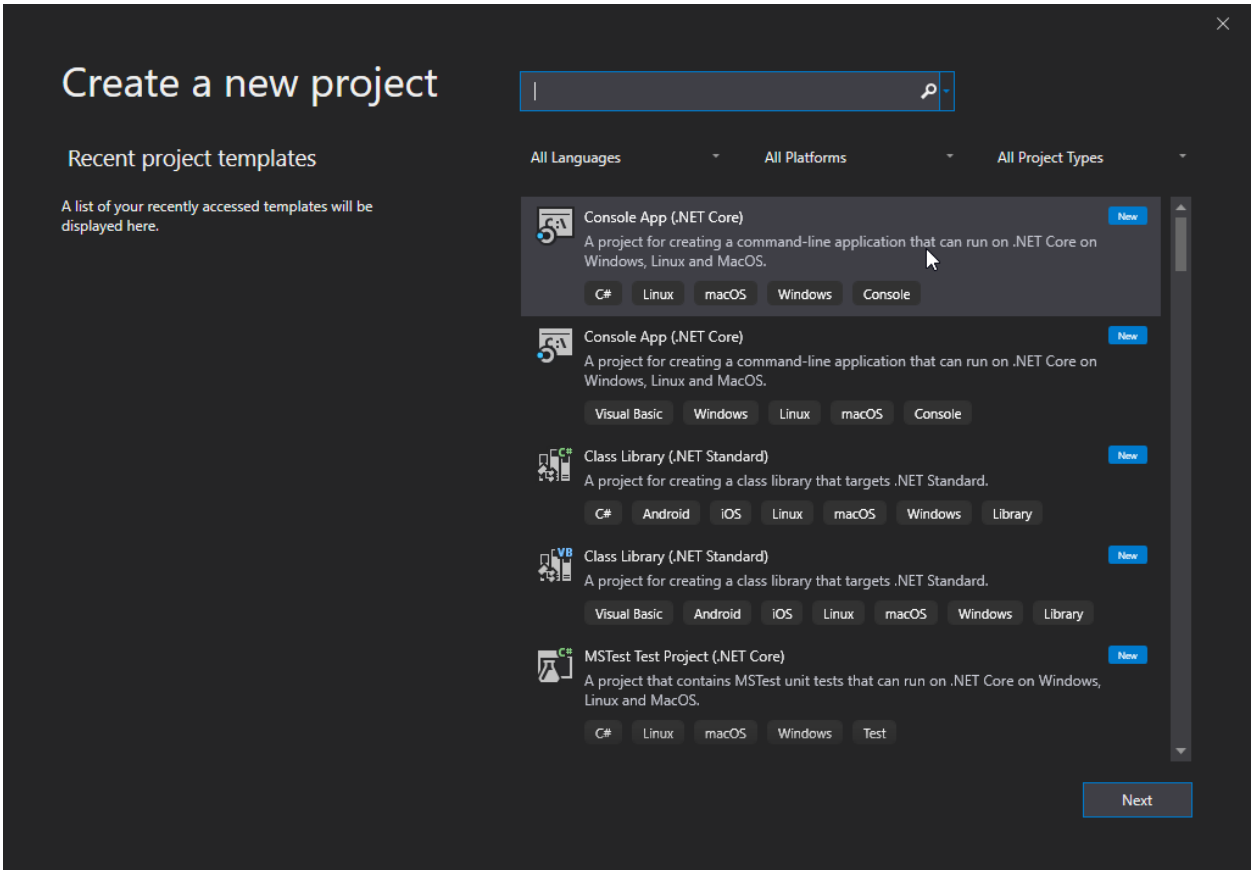
On the start window, choose **Create a new project**:



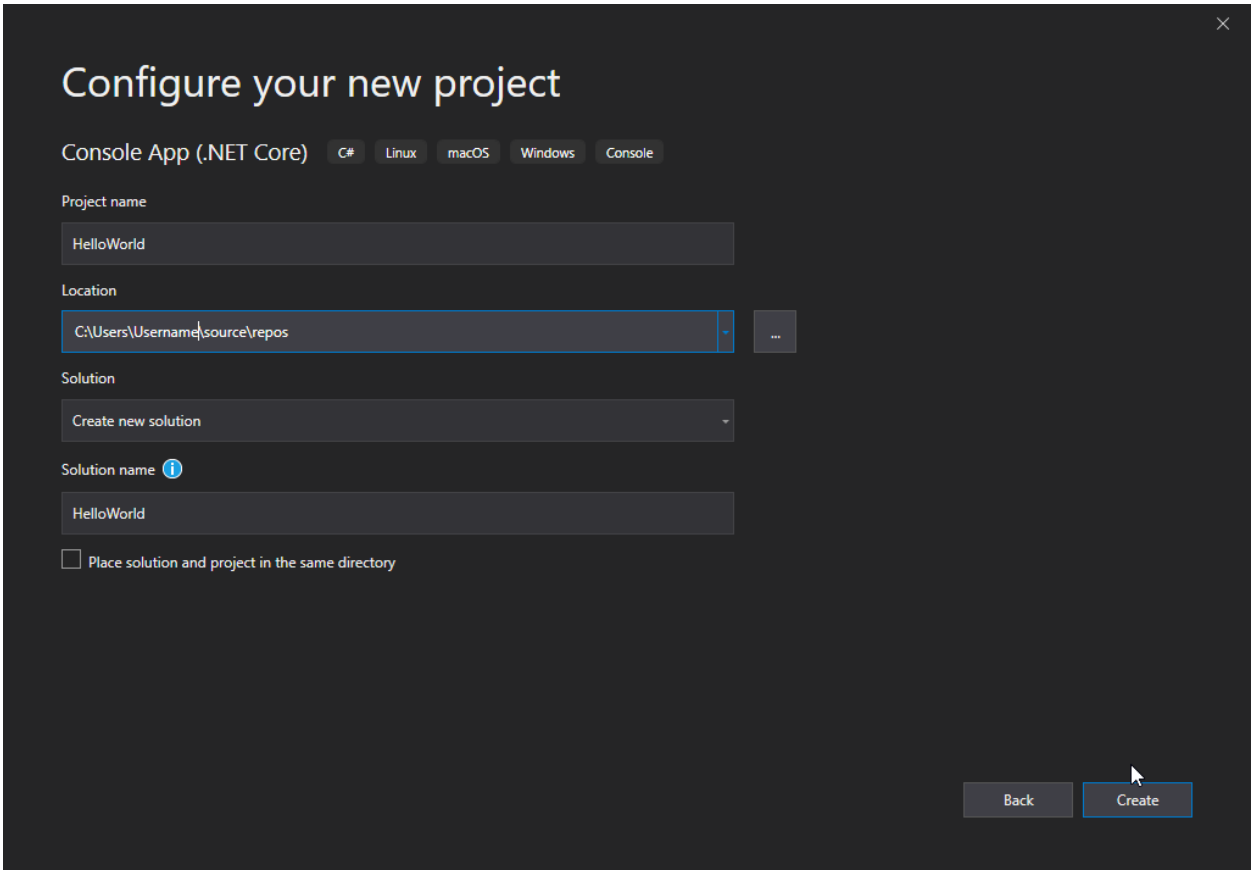
Then click on the "Install more tools and features" button:



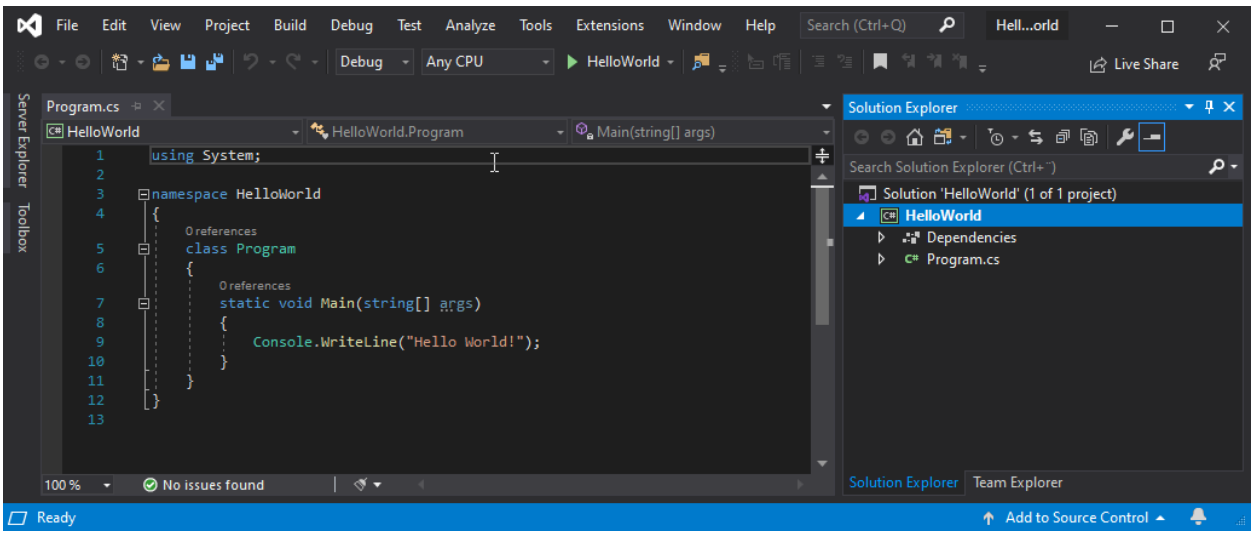
Choose "Console App (.NET Core)" from the list and click on the Next button:



Enter a name for your project, and click on the Create button:



Visual Studio will automatically generate some code for your project:



The code should look something like this:

Program.cs

```
using System;


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

Try it Yourself »

Don't worry if you don't understand the code above - we will discuss it in detail in later chapters. For now, focus on how to run the code.

Run the program by pressing the **F5** button on your keyboard (or click on "**Debug**" -> "**Start Debugging**"). This will compile and execute your code. The result will look something to this:

```
Hello World!
C:\Users\Username\source\repos\HelloWorld\HelloWorld\bin\Debug\netc
oreapp3.0\HelloWorld.exe (process 13784) exited with code 0.
To automatically close the console when debugging stops, enable
Tools->Options->Debugging->Automatically close the console when
debugging stops.
Press any key to close this window . . .
```

Congratulations! You have now written and executed your first C# program.

Learning C# At W3Schools

When learning C# at W3Schools.com, you can use our "Try it Yourself" tool, which shows both the code and the result. This will make it easier for you to understand every part as we move forward:

Program.cs

```
using System;


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

Result:

```
Hello World!
```

Try it Yourself »

C# Syntax

❮ Previous Next ❯

C# Syntax

In the previous chapter, we created a C# file called Program.cs, and we used the following code to print "Hello World" to the screen:

Program.cs

```
using System;


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

Result:

```
Hello World!
```

Try it Yourself »

Example explained

Line 1: `using System` means that we can use classes from the `System` namespace.

Line 2: A blank line. C# ignores white space. However, multiple lines makes the code more readable.

Line 3: `namespace` is used to organize your code, and it is a container for classes and other namespaces.

Line 4: The curly braces `{}` marks the beginning and the end of a block of code.

Line 5: `class` is a container for data and methods, which brings functionality to your program. Every line of code that runs in C# must be inside a class. In our example, we named the class Program.

Don't worry if you don't understand how `using System`, `namespace` and `class` works. Just think of it as something that (almost) always appears in your program, and that you will learn more about them in a later chapter.

Line 7: Another thing that always appear in a C# program, is the `Main` method. Any code inside its curly brackets `{}` will be executed. You don't have to understand the keywords before and after Main. You will get to know them bit by bit while reading this tutorial.

Line 9: `Console` is a class of the `System` namespace, which has a `WriteLine()` method that is used to output/print text. In our example it will output "Hello World!".

If you omit the `using System` line, you would have to write `System.Console.WriteLine()` to print/output text.

Note: Every C# statement ends with a semicolon `;`.

Note: C# is case-sensitive: "MyClass" and "myclass" has different meaning.

Note: Unlike Java, the name of the C# file does not have to match the class name, but they often do (for better organization). When saving the file, save it using a proper name and add ".cs" to the end of the filename. To run the example above on your computer, make sure that C# is properly installed: Go to the Get Started Chapter for how to install C#. The output should be:

```
Hello World!
```

C# Exercises

Exercise:

Insert the missing part of the code below to output "Hello World!".

```
static void ____(string[] args)
{
  ____.____("Hello World!");
}
```

Submit Answer »

Start the Exercise

C# Output

❮ PreviousNext ❯

C# Output

To output values or print text in C#, you can use the `WriteLine()` method:

Example Get your own C# Server

```
Console.WriteLine("Hello World!");
```

Try it Yourself »

You can add as many `WriteLine()` methods as you want. Note that it will add a new line for each method:

Example

```
Console.WriteLine("Hello World!");

Console.WriteLine("I am Learning C#");

Console.WriteLine("It is awesome!");
```

Try it Yourself »

You can also output numbers, and perform mathematical calculations:

Example

```
Console.WriteLine(3 + 3);
```

Try it Yourself »

The Write Method

There is also a `Write()` method, which is similar to `WriteLine()`.

The only difference is that it does not insert a new line at the end of the output:

Example

```
Console.Write("Hello World! ");

Console.Write("I will print on the same line.");
```

Try it Yourself »

Note that we add an extra space when needed (after "Hello World!" in the example above), for better readability.

C# Comments

❮ PreviousNext ❯

C# Comments

Comments can be used to explain C# code, and to make it more readable. It can also be used to prevent execution when testing alternative code.

Single-line Comments

Single-line comments start with two forward slashes (//).

Any text between // and the end of the line is ignored by C# (will not be executed).

This example uses a single-line comment before a line of code:

Example Get your own C# Server

```
// This is a comment

Console.WriteLine("Hello World!");
```

Try it Yourself »

This example uses a single-line comment at the end of a line of code:

Example

```
Console.WriteLine("Hello World!");  // This is a comment
```

Try it Yourself »

C# Multi-line Comments

Multi-line comments start with /* and ends with */.

Any text between /* and */ will be ignored by C#.

This example uses a multi-line comment (a comment block) to explain the code:

Example

```
/* The code below will print the words Hello World

to the screen, and it is amazing */
```

```
Console.WriteLine("Hello World!");
```

Try it Yourself »

Single or multi-line comments?

It is up to you which you want to use. Normally, we use `//` for short comments, and `/* */` for longer.

C# Exercises

Exercise:

Insert the missing parts to create two types of comments.

```
  This is a single-line comment
  This is a multi-line comment 
```

C# Variables

❮ Previous Next ❯

C# Variables

Variables are containers for storing data values.

In C#, there are different **types** of variables (defined with different keywords), for example:

- `int` - stores integers (whole numbers), without decimals, such as 123 or -123
- `double` - stores floating point numbers, with decimals, such as 19.99 or -19.99
- `char` - stores single characters, such as 'a' or 'B'. Char values are surrounded by single quotes
- `string` - stores text, such as "Hello World". String values are surrounded by double quotes
- `bool` - stores values with two states: true or false

Declaring (Creating) Variables

To create a variable, you must specify the type and assign it a value:

Syntax Get your own C# Server

```
type variableName = value;
```

Where *type* is a C# type (such as `int` or `string`), and *variableName* is the name of the variable (such as **x** or **name**). The **equal sign** is used to assign values to the variable.

To create a variable that should store text, look at the following example:

Example

Create a variable called **name** of type `string` and assign it the value "**John**":

```
string name = "John";

Console.WriteLine(name);
```

Try it Yourself »

To create a variable that should store a number, look at the following example:

Example

Create a variable called **myNum** of type `int` and assign it the value **15**:

```
int myNum = 15;
```

```
Console.WriteLine(myNum);
```

Try it Yourself »

You can also declare a variable without assigning the value, and assign the value later:

Example

```
int myNum;

myNum = 15;

Console.WriteLine(myNum);
```

Try it Yourself »

Note that if you assign a new value to an existing variable, it will overwrite the previous value:

Example

Change the value of `myNum` to 20:

```
int myNum = 15;

myNum = 20; // myNum is now 20

Console.WriteLine(myNum);
```

Try it Yourself »

Other Types

A demonstration of how to declare variables of other types:

Example

```
int myNum = 5;

double myDoubleNum = 5.99D;
```

```
char myLetter = 'D';

bool myBool = true;

string myText = "Hello";
```

C# Constants

❮ PreviousNext ❯

Constants

If you don't want others (or yourself) to overwrite existing values, you can add the `const` keyword in front of the variable type.

This will declare the variable as "constant", which means unchangeable and read-only:

Example[Get your own C# Server]

```
const int myNum = 15;

myNum = 20; // error
```

C# Display Variables

❮ PreviousNext ❯

Display Variables

The `WriteLine()` method is often used to display variable values to the console window.

To combine both text and a variable, use the + character:

Example Get your own C# Server

```
string name = "John";

Console.WriteLine("Hello " + name);
```

Try it Yourself »

You can also use the + character to add a variable to another variable:

Example

```
string firstName = "John ";

string lastName = "Doe";

string fullName = firstName + lastName;

Console.WriteLine(fullName);
```

Try it Yourself »

For numeric values, the + character works as a mathematical operator (notice that we use int (integer) variables here):

Example

```
int x = 5;

int y = 6;

Console.WriteLine(x + y); // Print the value of x + y
```

Try it Yourself »