­­

|  |
| --- |
|  |
| The Joy of Programming  Learning C# with Unity For Absolute Beginners |
| |  |  |  | | --- | --- | --- | | Kamran Bigdely-Shamloo | 6/4/19 |  | |

# [Download Unity](https://store.unity.com/download?ref=personal) (which includes Visual Studio).

If you have not installed visual studio, you can start coding in your browser. To do so, go to [here](https://docs.microsoft.com/en-us/dotnet/csharp/tutorials/intro-to-csharp/hello-world?tutorial-step=1) and click on the button “Enter focus mode”

# Other online C# editors:

https://try.dot.net/

https://dotnetfiddle.net/

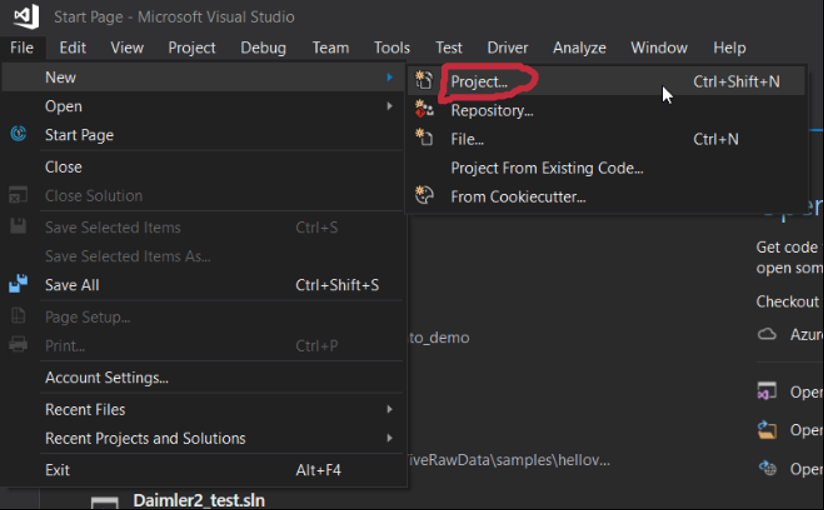
https://rextester.com/

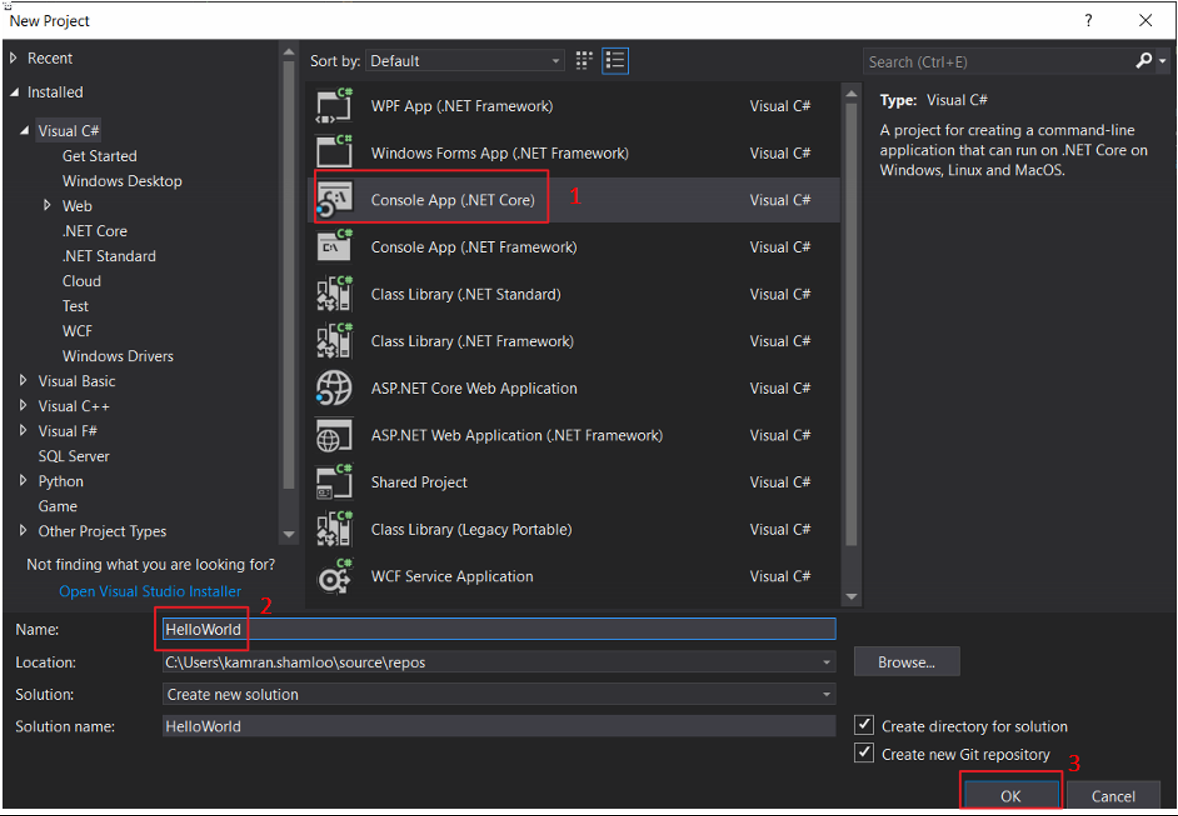
https://www.onlinegdb.com/online\_csharp\_compiler

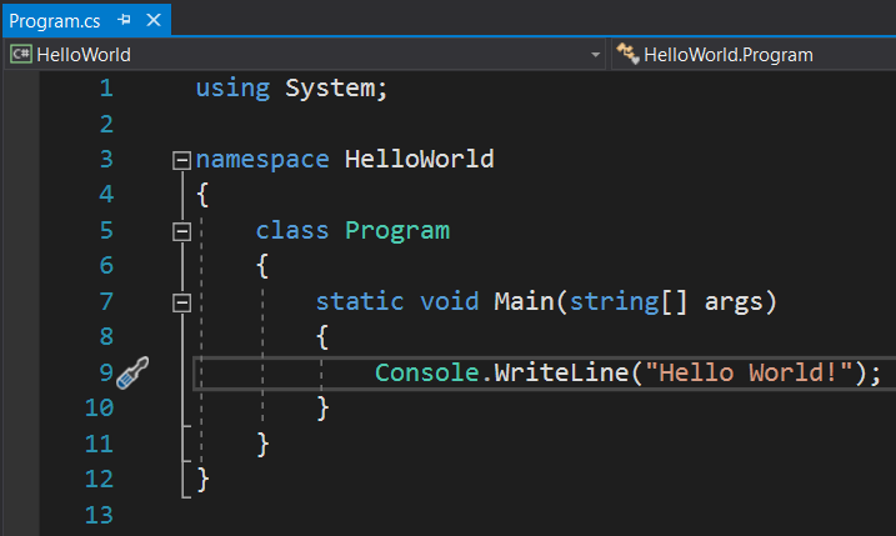
https://www.tutorialspoint.com/compile\_csharp\_online.php

# Your first program "Hello World"

Go to File > New > Project.

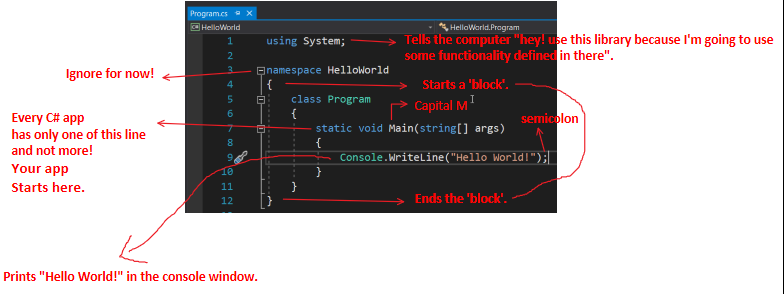






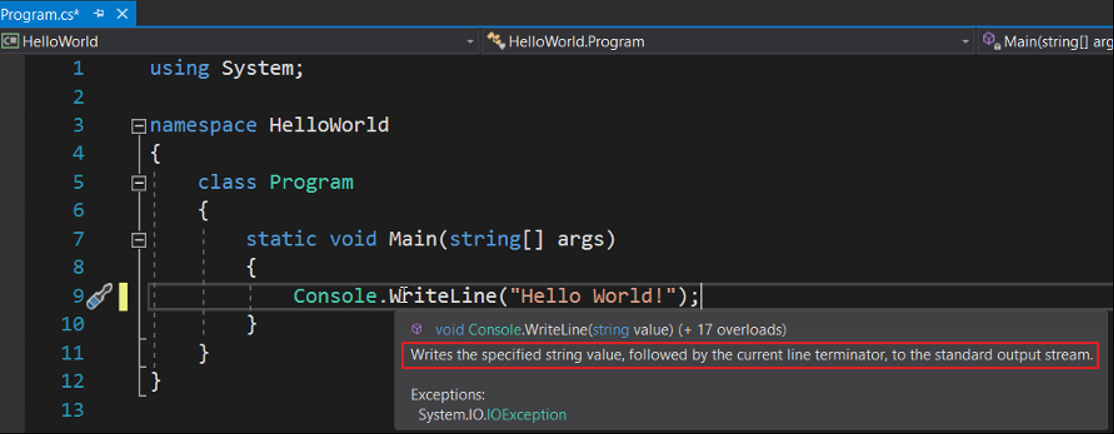
What do these text mean?

Let's study it line-by-line:



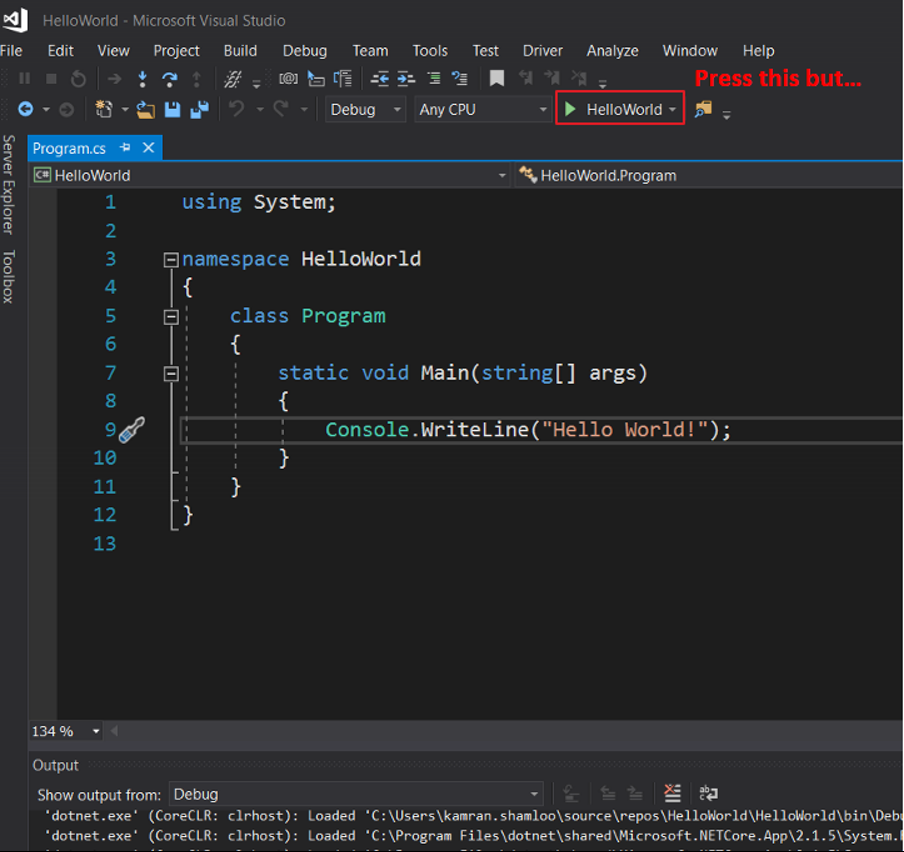
Note: C# language is case-sensitive! (i.e. 'main' and "Main" are different to C#)

Hover your mouse on "WriteLine(...)" method and read the hint (description box):



# How to run the program

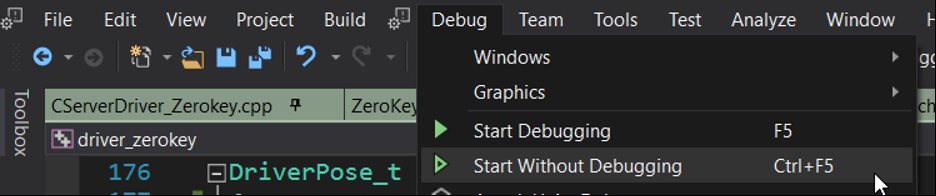
You can press the green play button at the top as follows:



The program runs fast and get closed quickly. You cannot see the output. How to see that?

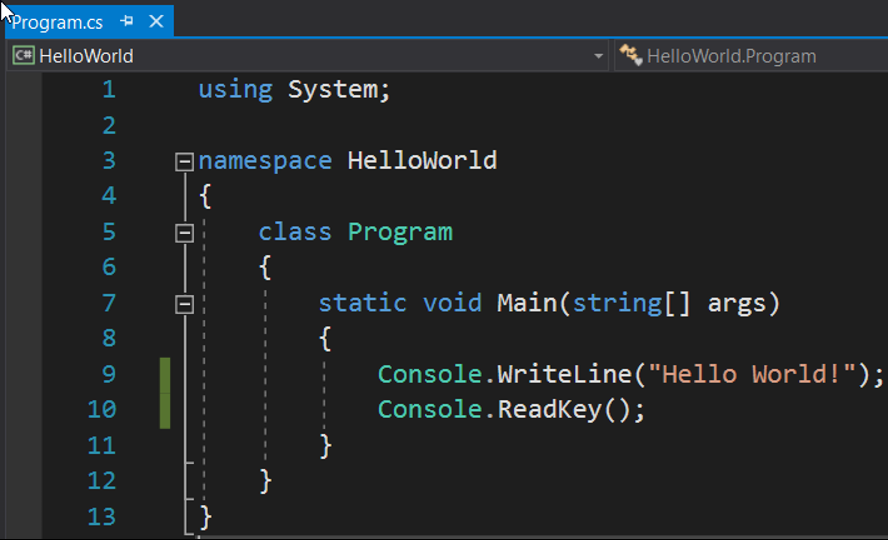
# How to stop Console window to let the programmer to see the output?

1 - Debug > Start Without Debugging (Ctrl + F5)

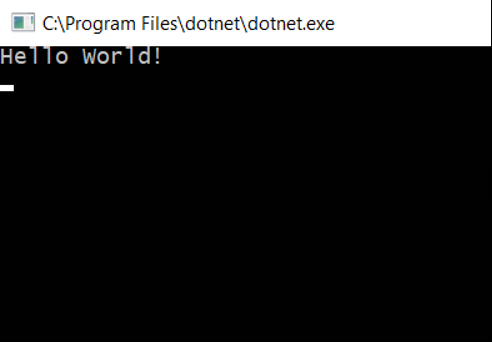


2 - Add the following at the end of your app:

Console.ReadKey();

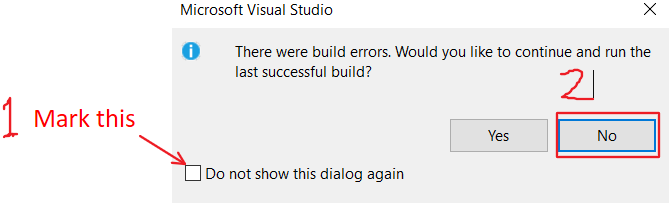


This is the console window (sometimes called output):



# Troubleshooting

If computer fails to run your program, you would see a window like this (Follow the instruction in red):

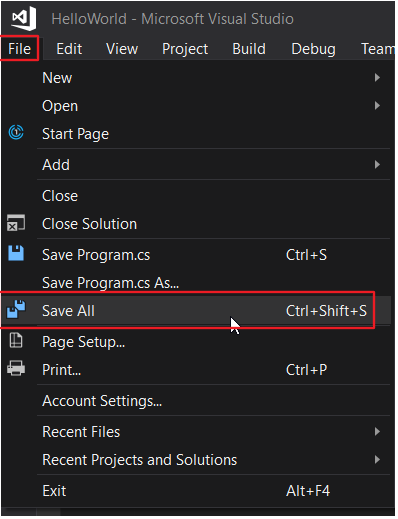


Several mistakes can cause the computer fails to run your program:

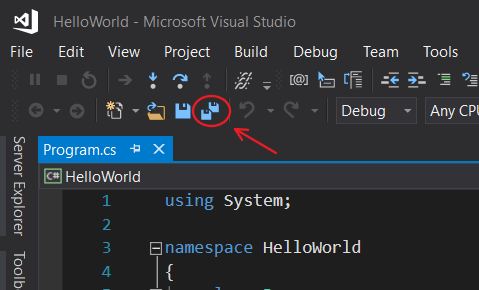
* Misspelling the commands. If so, you would see a red Squiggly (wavy) lines under it.
* Not adding semicolon ‘;’ at the end of the commands.
* Placing the commands outside of the curly brackets of the ‘Main’ method. The curly brackets are ‘{‘and ‘}’.

# Saving your work

To save your progress, go to **File > Save All** as follows:

****

Alternatively, you can click on this icon:



# Exercise: Say hello to yourself

Create a new Project called "MySecondApp". Complete the Main method by copying the two lines of code below and pasting them inside the body of your Main method:

Console.WriteLine("Hello World!");

Console.WriteLine("Hello [Insert your name here]!");

**Input Format**

There is no input for this challenge.

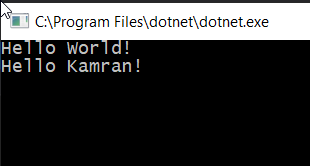
**Output Format**

You must print two lines of output:

1. Print Hello World! on the first line.

2. Print Hello Kamran! on the second line.

**Sample Output**



# Exercise: Say hello to world and yourself in the same line.

Write a program to print out

hello world and hello [your name]

on the same line (e.g. on a single line).

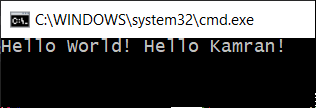
**Input Format**

There is no input for this challenge.

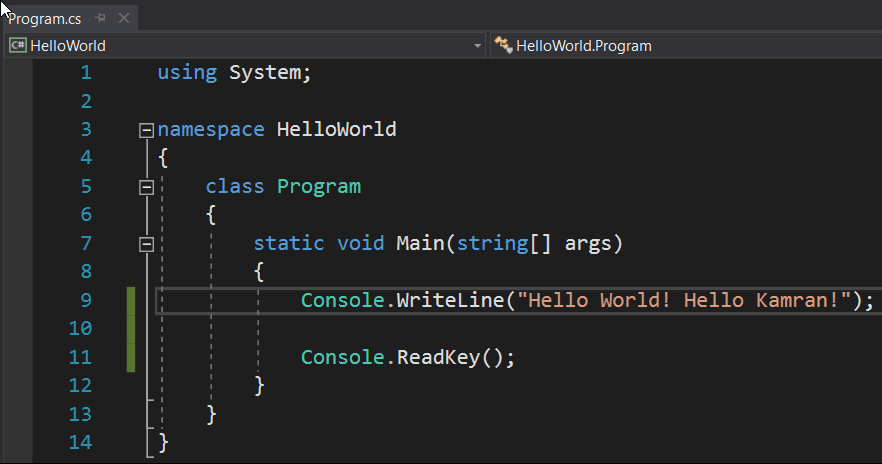
**Output Format**

You must print a single line of output.

**Sample Output**



**Solution**



Double Quotation, "your text here" is used to tell the computer "hey! This is a piece of text not a command".

# Exercise: Printing text vs printing numbers

Write a program that prints out 4.

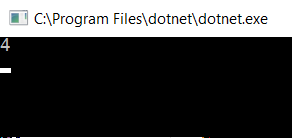
**Input Format**

There is no input for this challenge.

**Output Format**

You must print 4.

**Sample Output**



**Solution**

Both of the followings are correct:

1. Console.WriteLine(4);

2. Console.WriteLine("4");

**What's the difference?**

"4" means: character 4 e.g. the 4 as text. The 4 in the second solution means the

Number 4. The computer sees it as a number 4 not as a text.

To further understand the difference, solve the next challenge:

# Exercise: Printing text vs printing numbers (2)

Compare the output of

Console.WriteLine(4 + 2)

and

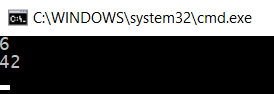
Console.WriteLine("4" + "2");

Explain the results.

**Solution**

Console.WriteLine(4 + 2) outputs: 6

Console.WriteLine("4" + "2") outputs: 42



When we write

Console.WriteLine(4 + 2)

We are asking the computer to calculate the sum of 4 and 2 and then print them. However, when we write

Console.WriteLine("4" + "2")

We are asking the computer to attach "4" text to "2" text. Microsoft has designed C# so that when we say "4" + "2", it concatenate (attach together) these two pieces of texts. Other programming languages might not do the same and simply tell you they don't understand what you mean.

# Comments

Comments are text in your program that are completely ignored by the computer. The computer is not even going to try to execute them. Here you can see some examples of comments:

// I'm a comment.

Console.WriteLine(4 + 2); // Prints out 6 in the console

/\* I'm a comment too \*/

/\* Behold! Let me introduce myself.

I'm a

Multi-line

Comment.

\*/

You put comments to explain a piece of code to yourself or your colleague.

Sometimes you use it to remind yourself to do something with the code in future.

# Exercise: Mathematical operations

Write a program to print the result of the following operations:

• -1 + 4

• -1 + 4 \* 6

• (-1 + 4) \* 6

• -1 + (4 \* 6)

• 4 / 2;

• 5 / 2;

• 3 / 7;

• 5.0 / 2;

• 14 + -4 \* 6 / 11;

• 6 % 5;

• (35 + 5) % 7;

Before running the app, guess the output. Compare your guess with the output and explain any possible differences.

**Input Format**

There is no input for this challenge.

**Output Format**

You must print out the result of each operation on a separate line.

**Solution**

static void Main(string[] args)

{

Console.WriteLine(-1 + 4); // 3

Console.WriteLine(-1 + 4 \* 6); // 23

Console.WriteLine((-1 + 4) \* 6); // 18

Console.WriteLine(-1 + (4 \* 6)); // 23

Console.WriteLine(4 / 2); // 2

Console.WriteLine(5 / 2); // 2 why? Because as long as the numbers in both sides of division operator '/' are integer numbers (whole number), computer performs integer division. In this type of division, computer ignores fractional part.

Console.WriteLine(3 / 7); // 0

Console.WriteLine(5.0 / 2); // 2.5 why? Because computer consider 5.0 as a fractional number and performs fractional division (also called floating point division).

Console.WriteLine(14 + -4 \* 6 / 11); // 12

Console.WriteLine(6 % 5); // 1, '%' is called remainder operator (also called modulo /ˈmäjəˌlō/)

Console.WriteLine((35 + 5) % 7); //5

Console.ReadKey();

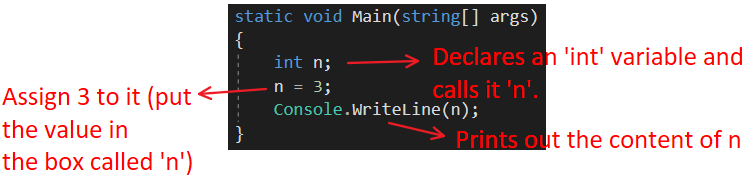
}

**Output**

Machine generated alternative text:
C:\Program Files\dotnet\dotnet.exe 
23 
18 
23 
12 

# Variables

Assume you want to keep numbers 3 and 8 in your program to perform some calculations. How do you do that? That’s why we have variables in programming languages. A variable is like a box that can hold something such as a number or a piece of text. For example,



An ‘**int**’ variable can hold **whole numbers** such as 0, -234 and 4398 but *cannot* hold **fractional numbers** such as 0.5, 2.84 and -89.51.

To hold fractional values (e.g. 3.25), you can use a ‘**double’** variable:

double fractionalVariable = 3.25;

Console.WriteLine(fractionalVariable);

# Exercise: Sum of two variables

Write a program that defines two variables ‘num1’ and ‘num2’. Print out their sum.

**Input Format**

There is no input for this challenge.

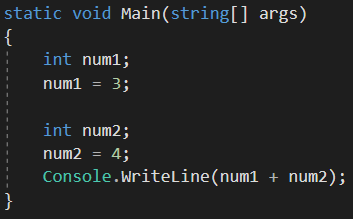
**Output Format**

You must print a single line of output.

**Sample Output**

7

**Solution**

****

Mini e­­xercise: In the above program, change the type of the variables from ‘int’ to ‘double’ and run it again.

We talked about **int** variables and **double** variables. There are many other types of variable. The next important type of variables are **string** variables. They can hold text:

string myName;

myName = "Kamran";

Console.WriteLine(myName); // Prints out “Kamran”.

# Exercise: Average of variables

Write a program that defines three variables ‘num1’, ‘num2’ and ‘num3’ with arbitrary values. Print out their average.

**Input Format**

There is no input for this challenge.

**Output Format**

Your output should be as follows:

The average is = [value]

You must print a single line of output.

**Sample Output**

The average is = 4.25

Note: Your output is likely to be different because it depends on your arbitrary numbers.

**Solution**

double num1 = -10;

double num2 = 20;

double num3 = 30;

double average = (num1 + num2 + num3) / 3;

Console.WriteLine("The average is: ");

Console.WriteLine(average);

**Output**

The average is:

13.3333333333333

**Question:** What would happen if we define all variables as ‘int’?

**Answer**: It would give you an incorrect or inaccurate average (depending on your numbers). Here, it would output 13 instead of 13.333333 because the computer performs an ‘int’ division not ‘fractional’ division as explained.

# Formatting the output

For making the output more understandable and/or concise, we may want to format our outputs. For example, we used the following

Console.WriteLine("The average is: ");

Console.WriteLine(average);

to print out this:

The average is:

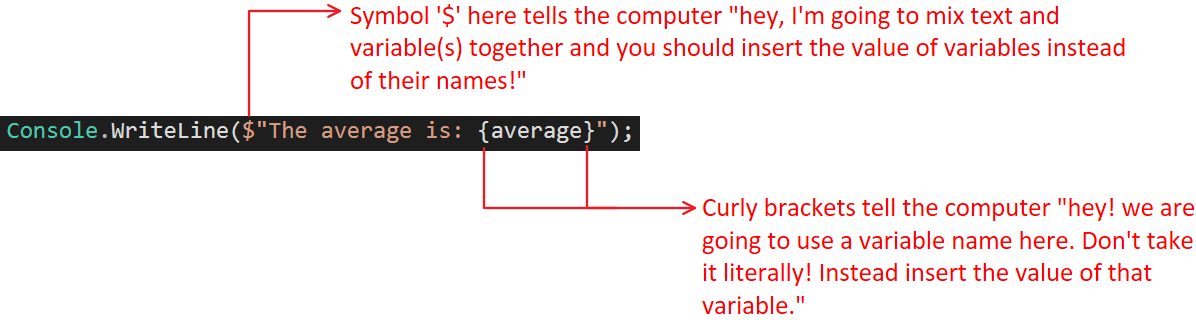
13.3333333333333

Now to make it concise, we may want to print out,

The average is 13.3333333333333.

but how?

To do that, we can use a new C#.6.0 feature called *string interpolation*:



# Exercise: Inserting numbers into text

Write a program that calculates the sum of two variable x1 and x2 and print them with a proper format.

**Input format**

This program has no input.

**Output format**

The output format should be as follows:

The sum of [FirstValue] and [SecondValue] is [SumOfTwoVariables].

**Sample output**

The sum of 2.25 and 8.75 is 11.

**Solution**

// Calculate the sum of two variables and format

// the output properly.

double x1 = 2.25;

double x2 = 8.75;

double sum = x1 + x2;

Console.WriteLine($"The sum of {x1} and {x2} is {sum}.");

**Output**

The sum of 2.25 and 8.75 is 11.

## Mini-Exercise: String and number concatenation

In the previous exercise (Inserting numbers into text), can we create the same output format by doing the following?

Console.WriteLine($"The sum of " + x1 + " and " + x2 + " is " + sum + ".");

Which method is easier for you?

# How to get input from users

Assume you want to write a program in which you ask the user to input text or a number. How do you do that? To do this, we write:

static void Main(string[] args)

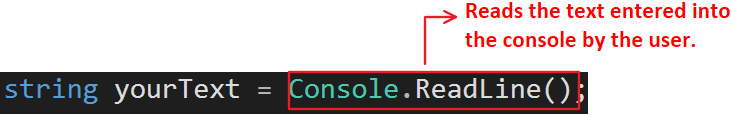
{

Console.WriteLine("Please enter a piece of text here:");

**string yourText = Console.ReadLine();**

Console.WriteLine("You entered:" + yourText);

}



# Exercise:

Write a program that gets 4 string values as follows:

1. your name
2. date of birth
3. place of your birth
4. a cool project you will worked on in the future

Then print the following (sample output):

***Margaret Hamilton*** is a software engineer. This person was born on ***August 17, 1936*** in ***Paoli, Indiana, USA***. This person worked on ***the Apollo project***.

To know more about her, click [here](https://en.wikipedia.org/wiki/Margaret_Hamilton_(software_engineer)#/media/File:Margaret_Hamilton_-_restoration.jpg).



Figure .Hamilton in 1969, standing next to listings of the software she and her MIT team produced for the Apollo project.

**Output format**

The output format should be as follows:

***[name]*** is a software engineer. This person was born on **[birth date]** in ***[birth place]***. This person worked on ***[project name].***

**Solution**

static void Main(string[] args)

{

    Console.WriteLine("Please, enter your name: ");

    string name = Console.ReadLine();

    Console.WriteLine("Please, enter your birth date:  ");

    string birthdate = Console.ReadLine();

    Console.WriteLine("Please, enter your birth place:  ");

    string birthPlace = Console.ReadLine();

    Console.WriteLine("Please, enter your cool project name:  ");

    string projectName = Console.ReadLine();

    Console.WriteLine($"{name} is a software engineer. This person was born on {birthdate} in {birthPlace}. This person worked on {projectName}.");

}

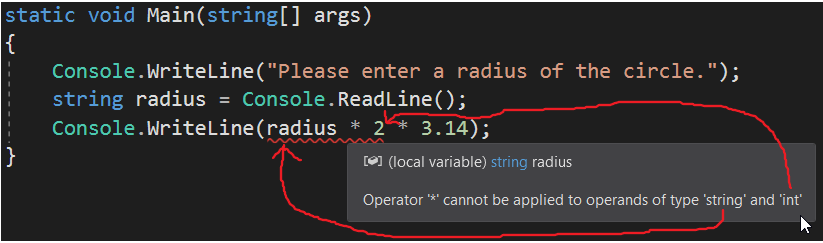
# Inputs as numbers

In the previous exercises, we learn to get string input from the user. In the following section, we will see how to get numbers from the user.

# Exercise: Calculate Circle perimeter

Write a program that asks the user to input a radius of a circle. Then the program should calculate and print out its perimeter.

Let’s try to write it with our current knowledge and see what would happen:



What’s going wrong here? As the computer is telling us, it won’t be able to multiply radius by 2. Why? Because from computer perspective, **radius** variable is just **a piece of text not a number**! In other words, it does not know how to multiply text by a number (**2** here). Even for us as human, it does not make sense to calculate the product of text and a number! (e.g. what is the product of “She went to bed” \* 2?).

Even if the user has entered 3 for radius, computer sees “3” as text not as a number. So how to tell computer this is the number 3 not the character ‘3’? We do this by a method that parses “3” into number 3:

double.Parse(radius)

Using this, we’ll have

static void Main(string[] args)

{

    Console.WriteLine("Please enter a radius of the circle.");

    string radius = Console.ReadLine();

    Console.WriteLine(double.Parse(radius) \* 2 \* 3.14); // this is line 15

}

The method double.Parse(radius) looks at the content of ‘radius’ and tries to convert it to a double value (i.e a fractional value). If ‘radius’ contains “6.8” then it converts it to 6.8.

## Mini-Exercise

One might ask what would happen if the ‘radius’ contains “peanut6.8”? Can it still convert it to 6.8? Try it yourself and see what would happen.

**Answer**

No. It cannot convert “peanut6.8” to 6.8. It gives you the following run-time error:

*Please enter a radius of the circle.*

*peanut6.8*

*Unhandled Exception: System.FormatException:* ***Input string was not in a correct format.***

*at System.Number.ParseDouble(String value, NumberStyles options, NumberFormatInfo numfmt)*

*at System.Double.Parse(String s)*

*at CirclePerimeterCalculationByGivenRadius.Program.Main(String[] args) in C:\Users\Kamran\Dropbox\Projects\CirclePerimeterCalculationByGivenRadius\CirclePerimeterCalculationByGivenRadius\Program.cs:****line 15***

# Converting string values to integer values

In the last example, we used double.Parse() to convert string to double values. We can also convert string values to integer values by using

int.Parse(stringVariable)

# Exercise:

Write a program that gets two numbers: 1 - the year you was born 2- current year. Then it calculates your age.

**Input Format**

The user enters two inputs (1 the year she was born 2- current year) in two separate lines.

**Output Format**

You are [age] years old.

**Sample Output**

*Please, enter the year your was born:*

***1985***

*Please enter the current year*

***2029***

*You are* ***44*** *years old.*

**Solution**

static void Main(string[] args)

{

    Console.WriteLine("Please, enter the year your was born:");

    int bornYear = int.Parse(Console.ReadLine());

    Console.WriteLine("Please enter the current year");

    int currentYear = int.Parse(Console.ReadLine());

    int age = currentYear - bornYear;

    Console.WriteLine($"You are {age} years old.");

}

# Exercise: Calculate the area and perimeter of a rectangle.

Write a program that receives width and height of a rectangle from the user and calculates its perimeter and area.

**Sample Output**

The user inputs is in bold. The rest is the output of the program:

*Please, enter the width (meter) and then height (meter) of a rectangle in separate lines:*

***3***

***5***

*The rectangle perimeter: 16*

*The rectangle area: 15*

**Solution**

static void Main(string[] args)

{

    Console.WriteLine("Please, enter the width (meter) and then height (meter) of a ractangle in separate lines:");

    double width= double.Parse(Console.ReadLine());

    double height = double.Parse(Console.ReadLine());

    double perimeter = (width + height) \* 2;

    double area = width \* height;

    Console.WriteLine($"The rectangle perimeter: {perimeter} ");

    Console.WriteLine($"The rectangle area: {area} ");

}

# Exercises For Home: Converting Among Temperature Scales

Write a program that gets the temperature in Celsius scale and convert it to Fahrenheit and Kelvin scales.

# Boolean Variables

We have already become familiar with variables of type int and double. Another super-useful type of variables are called boolean variables. They are defined as follows:

bool isJohnMarried;

isJohnMarried= true; // he is married.

bool storeClosed = false; // the store is still open.

As you see, boolean variables are like Yes/No. Their value can only be ***true*** or ***false***, nothing else.

**Note**: The boolean values are lower-case ‘true’ and lower-case ‘false’. If you write True or False (upper-case), the computer would complain and gives you an error.

Let’s write a simple program:

static void Main(string[] args)

{

    bool IraniansLoveKabob = true;

    Console.WriteLine(IraniansLoveKabob);

}

The output :

True

The Console.WriteLine() command printed out the content of ‘IraniansLoveKabo’. Note that the output is uppercase ‘True’ not ‘true’. Why? Because people at Microsoft somehow decided to convert ‘true’ to “True” when it gets printed out. Please not that you should always use ‘true’ and ‘false’ for boolean variables when programming in C#.

# Greater or Less that operators

There are many uses for boolean variables. You can use it to see the truthness of a comparison:

bool isLarger = 15 > 4;

Console.WriteLine(isLarger);

It outputs:

True

# Exercise:

Write a program that gets a temperature (in Celsius) from the user and prints out true if it’s above 45 degrees.

**Sample output**

Please, enter the air temperature (Celsius) at your location:

**31**

False

**Solution**

static void Main(string[] args)

{

    Console.WriteLine("Please, enter the air temprature (Celsius) at your location:");

    double temperature = double.Parse(Console.ReadLine());

    bool tooHot = temperature > 45;

    Console.WriteLine(tooHot);

}

# Conditional Statements

There are many situations in which we want to run different code based on a boolean value. Say, you have a noodle cooker. You want to write code for it such that:

* if the water gets cold (below 90 Celsius), the heating element turns on.

How do you write the ‘if’ part?

Engineers have developed a command for it. It is called ‘if’ statement. It looks like this:

If (something is true)  
{  
 doSomething.  
}

Now let’s write a real code for our noodle cooker:

static void Main(string[] args)

{

    Console.WriteLine("Please enter the noodle cooker's water temperature:");

    double temperature = double.Parse(Console.ReadLine());

    bool isCold = temperature < 90; // [Centigrade]

    if (isCold)

{

        Console.WriteLine("The heating element turns ON.");

}

}

# Exercise: Deep Fryer Temperature Control

You are a software developer for company that makes home deep fryers. Controlling the oil temperature of a deep fryer is the key factor to enhancing the food flavors. This product has a button for frying fish. The perfect temperature for frying fish is about 375 Fahrenheit. Write a program that receives the oil temperature and then controls the temperature as follows:

* If the oil temperature falls below 370, turn on the heating element.
* If the oil temperature increases above 380, turn off the heating element.

**Input Format**

The user enters a number (double).

**Sample Output**

***Here we provide two output samples:***

*Sample 1:*

*Please enter the deep fryer's oil temperature:*

***366***

*The heating element turns ON.*

*Sample 2:*

Please enter the deep fryer's oil temperature:

**383**

The heating element turns OFF.

**Solution**

static void Main(string[] args)

{

    Console.WriteLine("Please enter the deep fryer's oil temperature:");

    double temperature = double.Parse(Console.ReadLine());

    bool isCold = temperature < 370;  // [Fahrenheit]

    bool tooHot = temperature > 380;  // [Fahrenheit]

    if (isCold)  
 {

        Console.WriteLine("The heating element turns ON.");  
 }

    if (tooHot)  
 {

        Console.WriteLine("The heating element turns OFF.");  
 }

}

**Mini-Exercise**: Redo the last exercise by replace boolean variables with their equivalent statemens. For example, change

if (isCold)

To

If (temperature < 370).

**Mini-Exercise**: Redo the last exercise assuming we change the conditions as follows:

* If the oil temperature is **equal** **or below** 370, turn on the heating element.
* If the oil temperature is **equal** **or above** 380, turn off the heating element.

# Exercise: Largest number

Take three numbers (double) from the user and print the greatest number.

# Exercise: Even or odd

Write a program to check whether a number is even or odd.

# Exercise: Determine the sign of a number

Write a program that receives a number (double) and prints out:

‘-1’ if the number is negative

‘0’ if the number is zero

‘+1’ if the number is positive.

**Sample output**

*Enter a number and we will tell if you its sign:*

***-4.2***

*-1*

**Solution**

static void Main(string[] args)

{

    Console.WriteLine("Enter a number and we will tell if you its sign: ");

    double number = double.Parse(Console.ReadLine());

    if (number > 0)

    {

        Console.WriteLine("+1");

    }

    if (number < 0)

    {

        Console.WriteLine("-1");

    }

    if (number == 0)

    {

        Console.WriteLine("0");

    }

}

# If-else Statement

Here is how we write if-else statement.

**If** (something is true)  
{  
  *Do this.  
}*

**else**{ *Do that.  
}*

# Exercise: Minor/Adult

Write a program that receives the age of user and determins whether he or she is a mintor or an adult (below 18 is considered as minor). Code this program in two different ways:

1. Use if-else statement
2. Without using *else* statement.

Compare the two program.

# Exercise: Rewrite ‘Even or odd’ program using if-else statement.

# If-else if-else

Up to this point, we have learned if and if-else statement. There is a third thing called if-else if-else statement that is helpful in some situations. It looks like this:

if (something is true)  
{  
 do1...  
}  
else if (something else is true)  
{  
 do2...  
}  
else   
{  
 do3...  
}

# Exercise: Determine the Sign of a number using if-else statement.

Compare your program with the one you wrote before. Which one runs faster?

**Solution**

static void Main(string[] args)

{

    Console.WriteLine("Enter a number and we will tell if you its sign: ");

    double number = double.Parse(Console.ReadLine());

    if (number > 0)

    {

        Console.WriteLine("+1");

    }

    else if (number < 0)

    {

        Console.WriteLine("-1");

    }

    else

    {

        Console.WriteLine("0");

    }

}

# Equality operator ==

In many situations, we need to figure out whether something is equal to something else or not. To do so, we use equality operator:

If (myAge **==** yourAge)   
{  
Console.WriteLine(“we are at the same age”);  
}

**Note: Do not confuse ‘=’ with ‘==’:**

‘=’ 🡪 this is assignment operator e.g. string food = “Vietnamese pho”. It puts whatever comes in right (“Vietnamese pho”) into the left side (food).

‘==’ 🡪 It checks for equality e.g. myAge == yourAge. The statemnt is evaluated to true or false based on whether they are equal or not.

What would happen If your mistakenly use ‘=’ instead of ‘==’ like this?

If (myage = yourAge) // this is wrong.

The answer is the computer gives you an error preventing your program to run.

# Exercise: Write a program to input angles (int) of a triangle and check whether triangle is valid or not.

Note: a valid triangle:

- has three positive angles

- The sum of all three angles should equal to 180 degree.

# Inequality operator !=

If you want to check whether something is not euqal to somethig else, you say

If (myAge != yourAge)  
{  
 Console.WriteLine(“We are NOT at the same age”);  
}

# Exercise: Which one is a mamal?

Write a program that says whether an animal is a mamal or not.

**Input Format**

The program receives the animal name (string).

**Sample output**

*Choose one of the followings. We tell you whether it is a mammal or not.*

*Monotreme, Pinguin, Platypus, Human, Echidna*

*Platypus*

*Platypus is a mammal.*

**Solution**

static void Main(string[] args)

{

    Console.WriteLine("Choose one of the followings. We tell you whether it is a mammal or not. ");

    Console.WriteLine("Monotreme, Pinguin, Platypus, Human, Echidna");

    string animalName = Console.ReadLine();

    if (animalName != "Pinguin")

    {

        Console.WriteLine($"{animalName} is a mammal.");

    }

    else

    {

        Console.WriteLine($"{animalName} is NOT a mammal.");

    }

}

# String Comparison

When you want to see whether two string variabls have the same value or not, you can use the equality operator ‘==’. Also you can compare them like this:

If (yourName.Equals(“Zoe”)) // You can also use yourName == “Zoe”  
{  
 Console.WriteLine(“You should be Zoe!”);  
}

Note: there are some differences in some situation between == and .Equals(). We discussed some of the differences later and might ignore the rest as they are out of scope for this book.

# Exercise: Log-in password check

Assume you are a developer making a website. This website has a log-in page. In the login page, you only ask for a password. If the password is correct, you allow the user to enter the website.

Write a program that receives a string input from the user and checks whether the string matches the password.

**Sample output**

*Please enter the password to enter our secret cult:*

*JackIsBack!*

*Correct!, enter to our world with unfounded beliefs!*

**Solutions**

static void Main(string[] args)

{

    Console.WriteLine("Please enter the password to enter our secret cult: ");

    string userInput = Console.ReadLine();

    if (userInput == "JackIsBack!") // (userInput.Equals("JackIsBack!"))

    {

        Console.WriteLine("Correct!, enter to our world with unfounded beliefs!");

    }

    else

    {

        Console.WriteLine("Wrong value for password! Go away!");

    }

}

# Boolean Operations

## Relational Operators

|  |  |
| --- | --- |
| **Operator** | **Name** |
| == | equal to |
| != | not equal to |
| < | less than |
| <= | less than or equal to |
| > | greater than |
| >= | greater than or equal to |

# Exercise : Is it relational expression?

Which of the following are relational expressions in C#?

x == 3

x = 3

x >= 3

x \* 3

3 < x

x - 3 <= 10

## Logical Operators

|  |  |  |  |
| --- | --- | --- | --- |
| Operator | Name | Example | Result |
| **!** | NOT | !a | true if ‘a’ is false,  false if ‘a’ is true |
| **&&** | AND | a && b | true if both ‘a’ and ‘b’ are true. false otherwise |
| **||** | OR | a || b | true if ‘a’or ‘b’, or both are true. false if both are false. |

# Exercise: Hot Weather Warning

The weather is too hot and you want to write a program that alert people about it. (through their phone or something). Write a program that recieves the age of the person. If the person is younger than 7 or older than 65, the program should print out they should not go outside. Otherwise, it pritns out that they can go outside.

# Exercise: Robot burger flipper

You are a developer for a robot manifacturer. Their robot job is to flip burgers and cook them properly. For simplicity, assume they only check two parameters to see whether a burger is ready to serve or not:

1. Cooking time > 120 secodns
2. Cooking temprature > 350 F

If the cooking time of a burger is larger than 120 and its cooking temprature has been above 350, then the program should prints out “Burger Is Ready!”.

## Truth Tables

A truth table list all possible combinations of values for the variables in an express.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| a | b | a && b | a || b | !a |
| true | true | true | true | false |
| true | false | false | true | false |
| false | true | false | true | true |
| false | false | false | false | true |

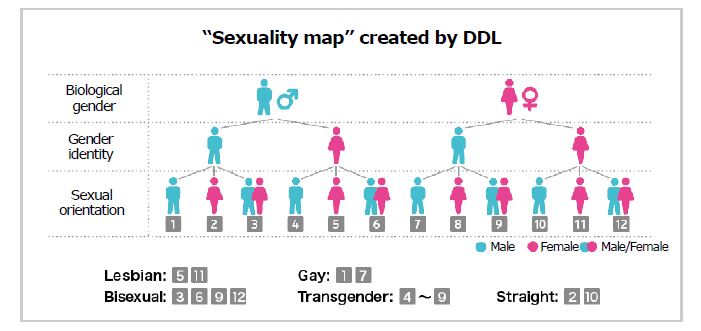
# Exercise : LGBTQ+ Classifier

Write a program that gets the following info from the user:

* Biological gender
* Gender identity (What gender they think they are)
* Sexual orientation (Whether they are interested in men, women or both or none.)

and figure out which category the person belongs to. The catetoreis are:

* Lesbian
* Gay
* Bisexual
* Transgender
* Straight



[resource](http://www.dentsu.com/news/sp/release/2019/0110-009756.html)

**Sample Output**

*What is your biological gender?*

*(enter 'm' for male and 'f' for female.)*

***m***

*What do you identify as (what gender you thing you are)?*

*(enter 'm' for male and 'f' for female.)*

***f***

*What is your sexual orientation (interested in 'm'ales, 'f'emales or 'b'oth)?*

***m***

*You are transexual*

**Solution**

static void Main(string[] args)

{

    //- Biological gender

    //- Gender identity (What gender they think they are)

    //- Sexual orientation (Whether they are interested in men, women or both or none.)

    Console.WriteLine("What is your biological gender?\n(enter 'm' for male and 'f' for female.)");

    string biologicalGender = Console.ReadLine();

    Console.WriteLine("What do you identify as (what gender you thing you are)?\n(enter 'm' for male and 'f' for female.)");

    string genderIdentity = Console.ReadLine();

    Console.WriteLine("What is your sexual orientation (interested in 'm'ales, 'f'emales or 'b'oth)?");

    string sexualOrientation = Console.ReadLine();

    bool lesbian = (biologicalGender == "f" && genderIdentity == "f" && sexualOrientation == "f") ||

                   (biologicalGender == "m" && genderIdentity == "f" && sexualOrientation == "f");

    bool gay     = (biologicalGender == "m" && genderIdentity == "m" && sexualOrientation == "m") ||

                   (biologicalGender == "f" && genderIdentity == "m" && sexualOrientation == "m");

    bool bisexual = sexualOrientation == "b";

    bool transgender = (biologicalGender == "f" && genderIdentity == "m") ||

                       (biologicalGender == "m" && genderIdentity == "f");

    bool straight = (biologicalGender == "f" && genderIdentity == "f" && sexualOrientation == "m") ||

                    (biologicalGender == "m" && genderIdentity == "m" && sexualOrientation == "f");

    if (lesbian)

    {

        Console.WriteLine("You are lesbian.");

    }

    if (gay)

    {

        Console.WriteLine("You are gay");

    }

    if (bisexual)

    {

        Console.WriteLine("You are bisexual");

    }

    if (transgender)

    {

        Console.WriteLine("You are transexual");

    }

    if (straight)

    {

        Console.WriteLine("You are straight");

    }

}

# Increment operator ++

Study this example:

double a = 3;

++a;

Console.WriteLine(a);   // output: 4.0

As you may have noticed, ++ operator increases the value of a by 1.

# Exercise: Adding by one

What’s the difference between the these:

1. x = x + 1;
2. x += 1
3. ++x;

Answer: No difference! All of them are exactly the same!

# Exercise: Increase a variable by 8

Answer:

u = u + 8;

or

u += 8;

# Exercise: Decrease a variable by 8

Answer

u = u - 8 ;

or

u -= 8;

# Exercise: Multipy a variable by 13

Anwer:

t = t \* 13

or

t \*= 13

# Loops

# While-loop

As a programmer, sometimes you want a piece of code to be executed seveal times and not just once. Study this:

while (true)

{

    Console.WriteLine("Hello World!");

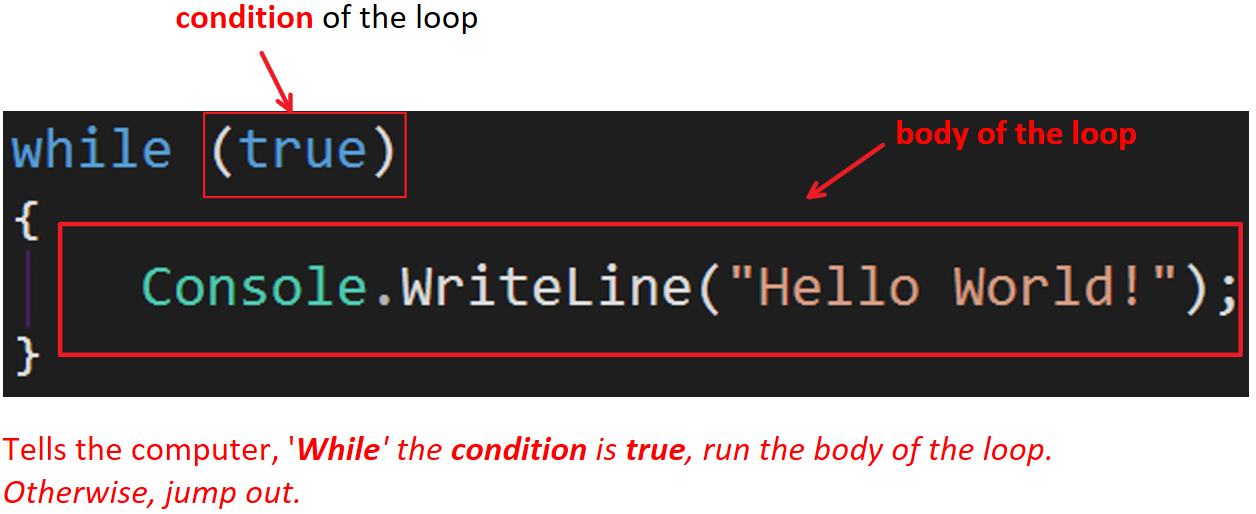
}

Guess what would be the output!

Now run this and see what is the output!

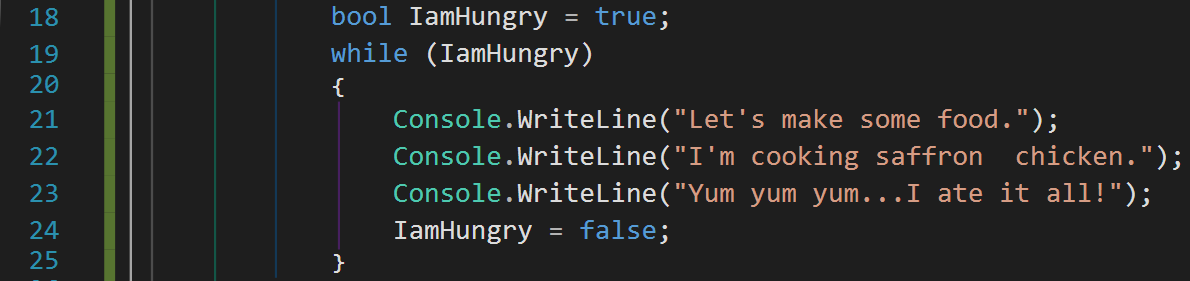
Answer:

If you run this code, you will get infinite number of “Hello World!”. Why? Because this is an infinite loop! Let’s dive in:



This program runs forever because the condition never becomes false. So it continousely prints out hello world till the user closes the console.

Let’s go through another example:



At line 19, ‘IamHungry’ is true so the program gets into the loop body. It runs lines 21, 22, 23 and then 24.

At line 24, the boolean variable becomes false. Then computer goes back to line 19. It evaluates ‘IamHungry’. Its value is false. Consequently, the computer jumps out of the loop to line 26.

As you may have understood at this point, while loop structure is like this:

while (boolean condition)

{

body…

}

It first check the ‘boolean condition’ and if it was true, it goes and run the ‘body’. Then it goes back to the boolean condition and checks it. If it is true, it runs the body again. It repeats this process till the boolean condition becomes false. It this case, it jumps out of the loop without running it anymore.

# Exercise

What is the output of the following program:

int i = 1;

while (i > 2)

{

    Console.WriteLine("i is still larger than 2");

}

**Answer**:

The output is nothing! When it gets to the boolean condition, it evalutes to false (beause is smaller than 2). Thus, it does not enter the loop body at all!

# Exercise:

What is the output of the following program:

int i = 3;

while (i > 2)

{

    Console.WriteLine("i is still larger than 2");

}

**Solution:**

It prints out “i is still larger than 2” infintiely!

# Exercise:

Write a program than prints out the integer numbers from 1 to infinity.

**Sample Output**

1

2

3

4

…

**Solution**

int i = 1;

while (true)

{

    Console.WriteLine(i);

    ++i;

}

# Exercise:

Write a program that outputs numbers from 1 to 10.

int i = 1;

while (i <= 10)

{

    Console.WriteLine(i);

    ++i;

}

# Exercise:

Write a program that outputs even numbers between 1 to 20 (including)

int i = 1;

while (i <= 20)

{

    if (i % 2 == 0)

        Console.WriteLine(i);

    ++i;

}

# Exercise:

Write a program that calculates the sum of numbers from 1 to 100 (including). In other words, it should calculates 1+2+3+ …+ 100.

**Solution**

int i = 1;

int sum = 0;

while (i <= 100)

{

    sum = sum + i;

    ++i;

}

Console.WriteLine(sum);

# Exercise

Write a program that repeatedly recieves a number from the user till the entered number is negative. Then it should print out the average of all entered numbers excluding the negative one.

For example, when the user enters

4

6

3

8

-2

Then the program should stop asking for a new number and also should prints out the average of 4,6,3 and 8 which is

5.25

**Solution**

static void Main(string[] args)

{

    // Write a program that repeatedly recieves a number from the user

    // till the entered number is negative. Then it should print out

    // the average of all entered numbers excluding the negative one.

    bool ongoing = true;

    double sum = 0;

    int i = 0;

    double userInput;

    while (ongoing)

    {

        userInput = double.Parse(Console.ReadLine());

        if (userInput > 0)

        {

            ++i;

            sum = sum + userInput;

        }

        else

        {

            ongoing = false;

        }

    }

    // Calculate the average.

    if (i != 0)

    {

        double average = sum / i;

        Console.WriteLine(average);

    }

    else

    {

        Console.WriteLine("No average!");

    }

}

# Exercise

Write a program in C# Sharp to display the first n odd numbers. At the end, it prints their sum.

**Sample Output**

*5*

*The odd numbers are:*

*1*

*3*

*5*

*7*

*9*

*Their sum is 25*

**Solution**

int number = int.Parse(Console.ReadLine());

Console.WriteLine("The odd numbers are: ");

int sum = 0;

int nOddsFound = 0;

int counter = 0;

while (nOddsFound < number)

{

    ++ counter;

    if (counter % 2 != 0)

    {

        ++ nOddsFound;

        Console.WriteLine(counter);

        sum += counter;

    }

}

Console.WriteLine("Their sum is " +  sum);

# Do-while loop:

There is a similar loop in C# called while-loop. It is written like this:

do

{

// Body of loop

} while (condition)**;**

Let’s solve an example to learn about it:

# Exercise: Print out 1 to 10 using while-loop

**Solution**

int i = 0;

do

{

    ++i;

    Console.WriteLine(i);

} while (i < 10);

# Exercise:

Write a program that adds numbers until user enters 0.

**Solution**

double userInput;

double sum = 0;

do

{

    userInput = double.Parse(Console.ReadLine());

    sum += userInput;

} while (userInput != 0);

Console.WriteLine($"The sum of all numbers is {sum}");

# for-loop:

In C#, there is another type of loop called “for-loop”. You use it when you know the number of iterations in advance (if you don’t know that, use while or do-while loops). For example, assume you want to print out numbers from 0 to 10. We can use a for-loop like this:

for (int i = 0; i < 10; i++)

{

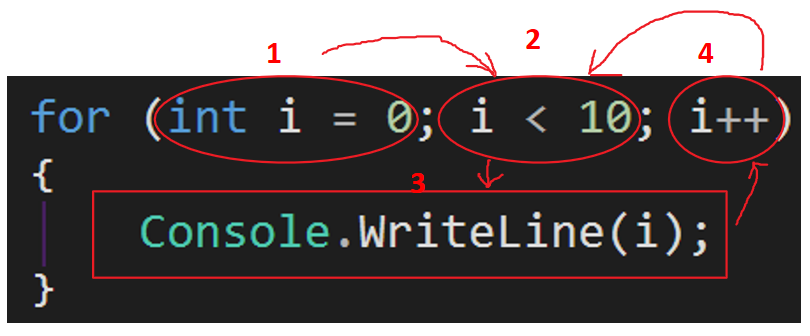
    Console.WriteLine(i);

}

// ----output------

0  
1  
2  
3  
4  
5  
6  
7  
8  
9

The computer goes through the code with this oder:



# Exercise:

Write a program that prints out numbers between 0 to 8.

**Solution**

for (int i = 0; i <= 8; i++)

{

    Console.WriteLine(i);

}

# Exercise:

Write a program that prints out all numbers that are divisible to 3 and 7 between 1 to 100.

**Solution**

for (int i = 1; i < 100; i++)

{

    if (i % 3 == 0 && i % 7 == 0)

        Console.WriteLine(i);

}

# Exercise: Write a program that calculates factorial of a number (any number below or equal 16)

**Solution**

// Write a program that calculates factorial of a number

// (any number below or equal 16)

Console.WriteLine("Enter a number below or equal 16.");

int x = int.Parse(Console.ReadLine());

int factorial = 1;

for (int i = 1; i <= x; i++)

{

    factorial = factorial \* i;

}

Console.WriteLine($"Its factorial is {factorial}");

# Exercise: FizzBuzz

Write a short program that prints each number from 1 to 100 on a new line.

For each multiple of 3, print "Fizz" instead of the number.

For each multiple of 5, print "Buzz" instead of the number.

For numbers which are multiples of both 3 and 5, print "FizzBuzz" instead of the number.

**Solution**

for (int i = 1; i <= 100; i++)

{

    if (i % 3 == 0 && i % 5 == 0)

    {

        Console.WriteLine("FizzBuzz");

    }

    else if (i % 3 == 0)

    {

        Console.WriteLine("Fizz");

    }

    else if (i % 5 == 0)

    {

        Console.WriteLine("Buzz");

    }

    else

    {

        Console.WriteLine(i);

    }

}

# Exercise

Given an integer, , print its first  multiples. Each multiple  (where ) should be printed on a new line in the form: N x i = result.

**Input Format**

A single integer

**Output Format**

Print  lines of output; each line  (where ) contains the of  in the form:   
N x i = result.

**Sample Input**2

**Sample Output**

2 x 1 = 2

2 x 2 = 4

2 x 3 = 6

2 x 4 = 8

2 x 5 = 10

2 x 6 = 12

2 x 7 = 14

2 x 8 = 16

2 x 9 = 18

2 x 10 = 20

Source: HackerRank

**Solution**

int number = int.Parse(Console.ReadLine());

for (int i = 1; i <= 10; i++)

{

    Console.WriteLine($"{number} \* {i} = {number\*i}");

}

# Exercise

Write a program in C# Sharp to display the pattern like right angle triangle using an asterisk as follows:

**Sample Output**

**7**

**\***

**\*\***

**\*\*\***

**\*\*\*\***

**\*\*\*\*\***

**\*\*\*\*\*\***

**\*\*\*\*\*\*\***

**Another Sample Output:**

**3**

**\***

**\*\***

**\*\*\***

**Note:** In this exercise, you need to use this command:

Console.**Write**(“You Text Here”);

In addition to

Console.**WriteLine**(“You Text Here”);

The command

Console.**Write**(…);

is slightly different from Console.WriteLine(“You Text Here”) in that Console.Write(…) prints out the text but **DOES NOT GO THE NEXT LINE**.

So if you have

Console.Write(“first”);

Console.Write(“second”);

The output becomes,

***firstsecond***

As you see, Console.Write(“first”) prints out in the same line but does not move the cursor to the new line. As a result, when the computer prints out ‘second’, it prints it out on the same line.

In contrast, if we do this:

Conssole.WriteLine(“first”) ;

Console.WriteLine(“second”);

The output becomes

**first**

**second**

This means Console.WriteLine() moves the cursor to the new line after it finished printing out its text.

# Loop Exercises:

# Exercise:

Write a program that prints out the Fibonacci numbers (only the first 20 of them).

Note: The Fibonacci sequence is a series where the next term is the sum of pervious two terms. The first two terms of the Fibonacci sequence is 0 followed by 1.

The Fibonacci sequence: 0, 1, 1, 2, 3, 5, 8, 13, 21

**Solution**

            int length = 20;

    int first = 0;

    int second = 1;

    Console.WriteLine(first);

    Console.WriteLine(second);

    int next;

    for (int i = 0; i < length - 2; i++)

    {

        next = second + first;

        Console.WriteLine(next);

        first = second;

        second = next;

    }

}

**break and continue**

**break:** The break statement terminates the closest enclosing loop. Study this example:

for (int i = 1; i <= 100; i++)

{

if (i == 5)

{

break;

}

Console.WriteLine(i);

}

// Keep the console open in debug mode.

Console.WriteLine("Press any key to exit.");

Console.ReadKey();

}

}

/\*

Output:

1

2

3

4

\*/

**Continue:**

The continue statement passes control to the next iteration of the enclosing loop statement in which it appears.

Example:

for (int i = 1; i <= 10; i++)

{

if (i < 9)

{

continue;

}

Console.WriteLine(i);

}

// Keep the console open in debug mode.

Console.WriteLine("Press any key to exit.");

Console.ReadKey();

}

}

/\*

Output:

9

10

\*/

**Arrays**

At this piont, we know how to get a data from user, do some calculations and print out the result. However, when it comes **to hold large amount of data**, we don’t know what to do!

Assume you want to write a software for a busy Japanese restaurnat. This program analyses customers orders during the last month and figures out which menu item (like Salmon sashimi) was the most popular item. To achieve that, the program has to go through thousands of orders. Obvisousely we cannot manually createa thousands variables. Even if we do, we don’t know how many variables we define because we don’t know exactly how many orders we are going to read from the receipts (or a database).

Another example is when you want to write a program to keep students’ data (first name, last names, their grades in various courses, etc) from a University. A typical universtiy has around 30000 students. Each year the number of students is different. So we would not know how many variables we should define in advance.

To solve this problem (having to define so many variables or not knowing how many variables to define), computer scientits has made a special variables called arrays. Array can hold any number of items. They can hold huge amont of data at once. Let’s see how to define and use them:

// Declare an array that holds 5 elements.

int[] numbers = new int[5];

// Populate the array

numbers[0] = 12;

numbers[1] = -8;

numbers[2] = 6;

numbers[3] = 20;

numbers[4] = 23;

// Print out the array's content.

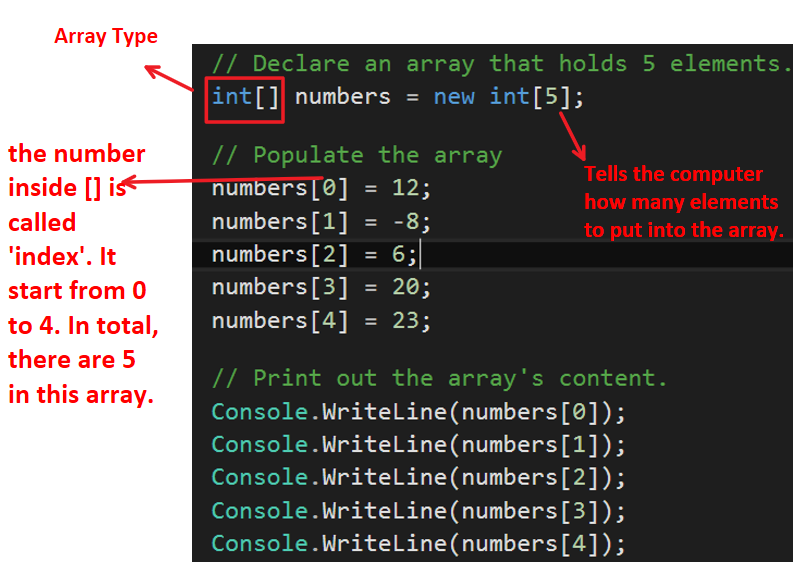
Console.WriteLine(numbers[0]);

Console.WriteLine(numbers[1]);

Console.WriteLine(numbers[2]);

Console.WriteLine(numbers[3]);

Console.WriteLine(numbers[4]);



Exercise:

Solution:

// Declare an array that holds 3 elements.

double[] numbers = new double[3] { 12, -8, 6 };

// Print out the array's content.

Console.WriteLine(numbers[0]);

Console.WriteLine(numbers[1]);

Console.WriteLine(numbers[2]);

Exercise:

Solution:

string[] names = new string[3];

names[0] = Console.ReadLine();

names[1] = Console.ReadLine();

names[2] = Console.ReadLine();

Console.WriteLine($"The names are {names[0]}, {names[1]} and {names[2]}.");

# Exercise: Write a program that find the largest number in an int array.

# Exercise: Bubble sort

Write a program that sort an int (or double) array such that the first item is the smallest and the last item is the largest item.

Example :

If you have this array:

6,2,8,3,2,5,55

it sorts it as:

2,2,3,5,6,8,55

**Solution**

static void Main(string[] args)

{

// Sort an array from smallest to largest.

int[] num = new int[] { 1, 2, 7, 1, 4, 3};

bool flag = true; // Tells whether at least one swapping action has happened.

do

{

flag = true;

// Traverse the array and swap any two adjacent elements that are not in the increasing order.

for (int i = 0; i < num.Length - 1; i++)

{

if (num[i + 1] < num[i]) // if in decreasing order

{

// Swap them

int temp = num[i + 1];

num[i + 1] = num[i];

num[i] = temp;

flag = false; // a swap has happended.

}

}

} while (!flag); // continue the loop as long as one swapping action has happened. Otherwise, jump out.

// Print out the array.

for (int j = 0; j < num.Length; j++)

{

Console.WriteLine(num[j]);

}

}

# Exercise: Palindrome

Write a program that tells you whether a name is a palindrome or not. Assume all letters are lower-case.

A 'palindrome' name is a name that reads the same backward as forward, e.g., "hannah" or "kayak".

**Solution**

static void Main(string[] args)

{

// Write a program that tells you whether a name is a palindrome or not. Assume all letters are lower-case.

// A 'palindrome' name is a name that reads the same backward as forward, e.g., "hannah" or "kayak".

string name = Console.ReadLine();

bool flag = true; // 'false' if we have found a character that is not equal to its corresponding character.'true' otherwise.

for (int i = 0; i < name.Length / 2; i++) // we only need to iterate through half of the string.(that means we have traversed all of the string).

{

if (name[i] != name[name.Length - 1 - i]) // if the character and its corresponding one are not equal.

{

flag = false;

// We found a character that is not the same as it corresponding one. Jump out.

break;

}

}

if (flag == false)

Console.WriteLine("It is Not a palindrome.");

else

Console.WriteLine("It is a palindrome.");

}

# Method

A method is block of code that does something. Here is a method:

void PrintInfo()   
{  
 Console.WriteLine(“Carrot juice is delicousd and healthy.”);  
 Console.WriteLine(“’The selfish gene’ is a great book.”);  
}

The method above prints out some info.

# Exercise: Method that calculates product of two numbers.

Write a method that receives two numbers and returns their product.

**Solution**

    static void Main(string[] args)

    {

        Console.WriteLine(product(3, 4));

    }

    /// <summary>

    /// Returns the product of two numbers.

    /// </summary>

    /// <param name="x"></param>

    /// <param name="y"></param>

    /// <returns></returns>

    static public double product(double x, double y)

    {

        return x \* y;

    }

}

# Exercise: Average of a number array.

Write a method that calculates the average of a given double array and returns it.

**Solution**

static void Main(string[] args)

{

// Write a method that calculates the average of

// a given double array and returns it.

double[] myNums = new double[] { 1, 2, 3, 4 };

Console.WriteLine($"The average is {GetAverage(myNums)}");

}

/// <summary>

/// Returns the average of a double array.

/// </summary>

/// <param name="nums"></param>

/// <returns></returns>

static public double GetAverage(double[] nums)

{

double sum = 0;

for (int i = 0; i < nums.Length; i++)

{

sum += nums[i];

}

return sum / nums.Length;

}

# Exercise: Reverse an array

The following program should reverses an array. Complete it.

static void Main(string[] args)

{

int[] nums = new int[] { 1, 2, 3, 4, 5, 6};

// TODO: Reveres the array such that it becomes

// {6,5,4,3,2,1}

// Print out the array.

for (int j = 0; j < nums.Length; j++)

{

Console.WriteLine(nums[j]);

}

}

**Solution**

static void Main(string[] args)

{

int[] nums = new int[] { 1, 2, 3, 4, 5, 6};

// Reveres the array such that it becomes

// {6,5,4,3,2,1}

for (int i = 0; i < nums.Length/2; i++)

{

int temp = nums[i];

nums[i] = nums[nums.Length - 1 - i];

nums[nums.Length - 1 - i] = temp;

}

// Print out the array.

for (int j = 0; j < nums.Length; j++)

{

Console.WriteLine(nums[j]);

}

}

# Exercise: check if array contains a duplicate integer number.

Write a method that returns true if it finds at least a duplicate value in a given array.

**Solution**

// Write a method that returns true if it finds at least a duplicate value in a given array.

static void Main(string[] args)

{

int[] nums = new int[] { 1, 2, 3, 4, 5};

Console.WriteLine(AnyDuplicate(nums));

}

/// <summary>

/// Returns true if there is at least a duplicate in the array.

/// </summary>

/// <returns></returns>

static bool AnyDuplicate(int[] numbers)

{

for (int i = 0; i < numbers.Length; i++)

{

for (int j = i + 1; j < numbers.Length; j++)

{

if (numbers[i] == numbers[j])

{

return true;

}

}

}

return false;

}

# Exercise:

Write a program that receives numbers from the user untill she enters -1. Then the program prints out all numbers she has entered (plus how many numbers she has entered).

**Sample output**

**4**

**6**

**3**

**4**

**-1**

**4 6 3 4**

**The length of the array is 4.**

# Struct

Sometimes we need to group related variables together. As as example, we want to record coordiantes of a single point in a 3d space. To do so, we need three variables:

double x;

double y;

double z;

The problem with this definition is that these variables are separate. They are not grouped together. Thus if we decide to send them to a method we have to send three of them individually to a method:

calculateLength(x,y,z);

This method call is OK but it’s wordy and not elegant. To write a cleaner code, we can group these variabls together in a struct:

class Program

{

struct Point

{

public double x;

public double y;

public double z;

}

static void Main(string[] args)

{

Point p = new Point();

p.x = 1;

p.y = 2;

p.z = 3;

Console.WriteLine(p.x);

double length = calculateLength(p);

Console.WriteLine(length);

}

static double calculateLength(Point p)

{

return Math.Sqrt(p.x \* p.x + p.y \* p.y + p.z \* p.z);

}

}