Code Clock

Day 1: Basic Input/Output

Learn.to.code

Coding with C#

@ QUB

Introduction to C#

Welcome to the exciting world of C# programming!

C# is a fun and powerful programming language that's perfect for beginners, especially kids like you. It's a modern, object-oriented language developed by Microsoft and is widely used for creating desktop applications, websites, and most famously, games with the Unity engine. Whether you want to create your own games, solve puzzles, or build cool projects, C# makes it easy to turn your ideas into reality.

What is C#?

C# is a type of language that computers understand. Just like we use English to talk to each other, we use C# to tell computers what to do. It is one of the most popular programming languages in the world because it's a powerful and versatile language.

Why Learn C#?

1. **Strongly Typed:** C# requires you to be specific about the type of data you are using. This helps you write correct code and catch mistakes early.
2. **Versatile:** You can use C# to make games (with Unity), create desktop applications, build websites, and much more!
3. **Powerful:** C# is used by many large companies to build professional, high-performance applications.

What Can You Do with C#?

- **Create Games:** Imagine making your own video game from scratch using the Unity game engine!
- **Build Applications:** C# is a great language for building all sorts of applications for Windows.
- **Solve Problems:** C# can help you solve maths problems, puzzles, and even real-world challenges.

Getting Started

You will be using Visual Studio to learn C#. Visual Studio is a powerful and free-to-download C# Code Editor.

Your First Program

Let's write your very first C# program together. It's called "Hello, World!" and it's super simple. This program will make the computer say "Hello, World!" on the screen. Ready? Let's go!

Open up Visual Studio and create a new **Console App** project. The main file will be called Program.cs.

Enter the following code inside the Main method:

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

What Happens Here?

- Console.WriteLine: This is a command that tells the computer to display something on the screen. The text you want to display goes inside the parentheses () and quotation marks "".
- "Hello, World!": This is the message we want to show.

When you run this program, you'll see the words "Hello, World!" appear on your screen. Congratulations, you just wrote your first C# program!

C# Code Structure

In C#, the code is organised into classes and methods. A basic console application will look like this:

```
using System; // This line imports a library that we need

class Program
{
    static void Main(string[] args)
    {
        // This is where we write our code!
    }
}
```

C# Indentation

Indentation refers to the spaces at the beginning of a code line. In C#, indentation is used to make the code more readable and easier to follow. The curly braces {} are used to define a block of code.

For example, a loop uses indentation to show which lines of code are part of the loop's block:

```
int counter = 1;

while (counter <= 5)
{
    Console.WriteLine(2 * counter);
    // increments counter by 1 each iteration of the loop
    counter = counter + 1;
}

Console.WriteLine($"The value of the counter is {counter}");
Console.WriteLine("The loop has now terminated");
```

C# Comments

Comments can be used to explain C# code.

- Comments can be used to make the code more readable.
- Comments can be used to prevent execution when testing code.

A single-line comment starts with //.

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

You can also use multi-line comments. These start with /* and end with */.

```
/*
This is a comment
written in
more than just one line
*/
Console.WriteLine("Hello, World!");
```

Variables

Variables are containers for storing data values.

Creating Variables

In C#, you must declare the data type and the name of the variable before you assign a value to it.

```
int x = 5;
string y = "John";
Console.WriteLine(x);
Console.WriteLine(y);
```

You can change the value of a variable, but you cannot change its data type after it has been declared.

```
int x = 4; // x is of type int
// x = "Sally"; // This would cause an error because "Sally" is a string
```

Variable Names

A variable can have a short name (like x and y) or a more descriptive name (age, carName, totalVolume). Rules for C# variables:

- A variable name must start with a letter or the underscore character.
- A variable name cannot start with a number.
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _).
- Variable names are case-sensitive (age, Age and AGE are three different variables).
- A variable name cannot be any of the C# keywords (e.g., int, string, if).

Here are some examples of legal variable names:

```
string myvar = "Code Clock";
string my_var = "Code Clock";
string _my_var = "Code Clock";
string myVar = "Code Clock";
string MYVAR = "Code Clock";
string myvar2 = "Code Clock";
```

Now try entering the following illegal variable names:

```
// int 2myvar = "John";  // Cannot start with a number
// string my-var = "John"; // Cannot contain a hyphen
// string my var = "John"; // Cannot contain a space
```

Casting

If you want to specify the data type of a variable, this can be done with casting. This is often necessary when you get input from the user, which is always a string.

```
string x = Convert.ToString(3); // x will be "3"
int y = Convert.ToInt32(3); // y will be 3
double z = Convert.ToDouble(3); // z will be 3.0
```

Get the Type

You can get the data type of a variable with the GetType() method.

```
int x = 5;
string y = "John";
Console.WriteLine(x.GetType());
Console.WriteLine(y.GetType());
```

Case-Sensitive

Variable names in C# are case-sensitive.

```
int a = 4;
string A = "Sally";

Console.WriteLine(a);
Console.WriteLine(A); // A will not overwrite a
```

Output Variables

The C# Console.WriteLine() method is used to output variables.

```
string x = "Code Clock is awesome";
Console.WriteLine(x);
```

You can output multiple variables by using string concatenation with the + operator, or with modern string interpolation using a $ symbol:

```
string x = "C#";
string y = "is";
string z = "awesome";

// Using string interpolation
Console.WriteLine($"{x} {y} {z}");
```

Built-in Data Types

In programming, a data type is an important concept. Variables can store data of different types, and different types can do different things.

C# has the following data types built-in by default:

- **Integer Types:** int, long
- **Floating-Point Types:** float, double, decimal
- **Character Types:** char, string
- **Boolean Type:** bool

C# Numbers

There are two common numeric types in C#:

- int (for whole numbers)
- double (for floating-point numbers with decimals)

Variables of numeric types are created when you assign a value to them.

```
int x = 1; // int
double y = 2.8; // double
```

Random Number

To generate a random number in C#, you need to use the Random class. You must first create an instance of the Random class and then call its Next() method.

The using System; line at the top of your file is important as it imports the necessary code to use the Random class.

```
Random rnd = new Random();
int randomNumber = rnd.Next(1, 11); // Generates a number from 1 to 10 (the second number is exclusive)
Console.WriteLine(randomNumber);
```

C# Conditions and if statements

C# supports the usual logical conditions from mathematics:

- Equals: a == b
- Not Equals: a != b
- Less than: a < b
- Less than or equal to: a <= b
- Greater than: a > b
- Greater than or equal to: a >= b

These conditions can be used in several ways, most commonly in if statements and loops. An if statement is written by using the if keyword.

```
int a = 33;
int b = 200;
if (b > a)
{
    Console.WriteLine("b is greater than a");
}
```

else if

The else if keyword is C#'s way of saying "if the previous conditions were not true, then try this condition."

```
int a = 33;
int b = 33;
if (b > a)
{
    Console.WriteLine("b is greater than a");
}
else if (a == b)
{
    Console.WriteLine("a and b are equal");
}
```

else

The else keyword catches anything which isn't caught by the preceding conditions.

```
int a = 200;
int b = 33;
if (b > a)
{
    Console.WriteLine("b is greater than a");
}
else if (a == b)
{
    Console.WriteLine("a and b are equal");
}
else
{
    Console.WriteLine("a is greater than b");
}
```

In this example, a is greater than b, so the first condition is not true. The else if condition is also not true, so we go to the else condition and print to the screen that "a is greater than b".

You can also have an else without the else if:

```
int a = 200;
int b = 33;
if (b > a)
{
    Console.WriteLine("b is greater than a");
}
else
{
    Console.WriteLine("b is not greater than a");
}
```

Logical Operators

The && (and), || (or), and ! (not) operators are used to combine conditional statements.

&& (And)

Test if a is greater than b AND if c is greater than a:

```
int a = 200;
int b = 33;
int c = 500;
if (a > b && c > a)
{
    Console.WriteLine("Both conditions are True");
}
```

|| (Or)

Test if a is greater than b OR if a is greater than c:

```
int a = 200;
int b = 33;
int c = 500;
if (a > b || a > c)
{
    Console.WriteLine("At least one of the conditions is True");
}
```

! (Not)

The ! keyword is a logical operator that is used to reverse the result of the conditional statement.

Test if a is NOT greater than b:

```
int a = 33;
int b = 200;
if (!(a > b))
{
    Console.WriteLine("a is NOT greater than b");
}
```

Nested If

You can have if statements inside if statements. This is called a nested if statement.

```
int x = 41;
if (x > 10)
{
    Console.WriteLine("Above ten,");
    if (x > 20)
    {
        Console.WriteLine("and also above 20!");
    }
    else
    {
        Console.WriteLine("but not above 20.");
    }
}
```

Challenges

Using basic input/output commands, write the code which does the following:

1. Prints today's date.
2. Asks the user to enter their name, gender, and age and prints it out.
3. Using the user's age, checks if they are under 12, in which case, it should say "You are too young to be here!"
4. Checks the user's age against the following age boundaries for each year group and prints their year out. i.e., Year 8 (12), Year 9 (13), Year 10 (14), Year 11 (15), etc.

Solution

```
using System;

public class Challenge
{
    public static void Main(string[] args)
    {
        // 1. Prints today's date
        Console.WriteLine($"Today's date is: {DateTime.Now.ToShortDateString()}");

        // 2. Asks the user to enter their name, gender, and age and prints it out.
        Console.Write("Please enter your name: ");
        string name = Console.ReadLine();

        Console.Write("Please enter your gender: ");
        string gender = Console.ReadLine();

        Console.Write("Please enter your age: ");
        int age = Convert.ToInt32(Console.ReadLine());

        Console.WriteLine($"\nHello, {name}!");
        Console.WriteLine($"Gender: {gender}");
        Console.WriteLine($"Age: {age}");

        // 3. Checks if they are under 12
        if (age < 12)
        {
            Console.WriteLine("You are too young to be here!");
        }

        // 4. Checks the user's age against year group boundaries
        if (age == 12)
        {
            Console.WriteLine("You are in Year 8.");
        }
        else if (age == 13)
        {
            Console.WriteLine("You are in Year 9.");
        }
        else if (age == 14)
        {
            Console.WriteLine("You are in Year 10.");
        }
        else if (age == 15)
        {
            Console.WriteLine("You are in Year 11.");
        }
        // You can add more conditions for higher ages here
        else
        {
            Console.WriteLine("Your year group is outside of the specified range.");
        }
    }
}
```