**C# Notes**

**Introduction**

* C# is a statically typed, generally-purpose, OOP, programming language
* Microsoft created C# and C# is the main language used with the .NET Framework.
* Before C#, we had C and C++. When we compile C or C++, the code is translated to machine code for the machine it’s running on. However, different machines have different hardware and operating systems. Thus, C or C++ code on one machine might not run on another machine. Thus, Microsoft wanted to solve this issue which is why they created C#. they made it so that C# code is not compiled into machine code. Instead, it is translated into an intermediate language called IL code (Intermediate Language Code). This IL code is independent of the hardware it’s running on. We then translate the IL code into machine code using the CLR via the .NET framework (more on this later).

**Setup:** <https://www.youtube.com/watch?v=CO4BGZOuUkM&t=520s>

**Hello World:**

* Text

  Description automatically generated
* Output: Hello World!
* 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 make the code more readable.
* Line 3: namespace allows us to group our classes. For example, we may have a different project with a class that’s also named Program. But we can have different namespaces so the Program class from this project is distinguishable from the Program class from the other project.
* 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.
* Line 7: Another thing that always appear in a C# program, is the Main method which is the entry point for all C# programs. Any code inside its curly brackets {} will be executed.
  + Description of the things that come before and after the word Main:
  + static: is a keyword which means object is not required to access static members. So it saves memory.
  + void: is the return type of the method. It doesn't return any value. In such case, return statement is not required.
  + Main: is the method name. It is the entry point for any C# program. Whenever we run the C# program, Main() method is invoked first before any other method. It represents start up of the program.
  + string[] args: is used for command line arguments in C#. While running the C# program, we can pass values. These values are known as arguments which we can use in the program.
* 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.

**Output**

* To output values or print text in C#, you can use the WriteLine() and Write() methods. WriteLine() adds a new line at the end while Write() does not.
* Ex:
* Text

  Description automatically generated
* Output:
* A screenshot of a computer screen

  Description automatically generated with low confidence
* Ex:
* Text, letter

  Description automatically generated
* Output: Hello World! I will print on the same line.
* Console.WriteLine actually can take parameters after an initial string, but only when the string is in the form of a format string, with expression(s) in braces where substitutions are to be made, (like in fill-in-the-blanks).
* The remaining parameters, after the initial string, give the values to be substituted. To know which further parameter to substitute, the parameters after the initial string are implicitly numbered, starting from 0. Starting with 0 is consistent with other numbering sequences in C#. So here, where there is just one value to substitute, it gets the index 0, and where it is substituted, the braces get 0 inside, to indicate that parameter 0 is to be substituted.
* Ex: 
* Output: 

**Comments**

* Single-line comments start with two forward slashes (//).
* Multi-line comments start with /\* and ends with \*/.
* Text

  Description automatically generated

**Data Types**

* The variables in C#, are categorized into the following types −
  + Value types
  + Reference types
  + Pointer types
* Diagram

  Description automatically generated
* Graphical user interface, text, application, email

  Description automatically generated
* Value Type
  + Value type variables directly contain data.
  + The value data types are integer-based and floating-point based. C# language supports both signed and unsigned literals.
  + They are derived from the class System.ValueType.
  + There are 2 types of value data type in C# language.
  + 1) Predefined Data Types - such as Integer, Boolean, Float, etc.
  + 2) User defined Data Types - such as Structure, Enumerations, etc.
  + The memory size of data types may change according to 32 or 64 bit operating system.
  + Graphical user interface, text, application, email

    Description automatically generated
* Reference Data Type
  + The reference data types do not contain the actual data stored in a variable, but they refer to a memory location.
  + Using multiple variables, the reference types can refer to a memory location. If the data in the memory location is changed by one of the variables, the other variable automatically reflects this change in value.
  + There are 2 types of reference data type in C# language.
  + 1) Predefined Types - such as Objects, String.
  + 2) User defined Types - such as Classes, Interface.
  + Note: A String is a reference type even though it has most of the characteristics of a value type such as being immutable and having == overloaded to compare the text rather than making sure they reference the same object. But strings aren't value types since they can be huge, and need to be stored on the heap. Value types are (in all implementations of the CLR as of yet) stored on the stack. Stack allocating strings would break all sorts of things: the stack is only 1MB for 32-bit and 4MB for 64-bit, you'd have to box each string, incurring a copy penalty, you couldn't intern strings, and memory usage would balloon, etc.
* Pointer Data Type
  + Pointer type variables store the memory address of another type. Pointers in C# have the same capabilities as the pointers in C or C++.
* Object Type
  + The Object Type is the ultimate base class for all data types in C# Common Type System (CTS). Object is an alias for System.Object class. The object types can be assigned values of any other types, value types, reference types, predefined or user-defined types. However, before assigning values, it needs type conversion.
  + When a value type is converted to object type, it is called boxing and on the other hand, when an object type is converted to a value type, it is called unboxing.
  + 
* Dynamic Type
  + You can store any type of value in the dynamic data type variable. Type checking for these types of variables takes place at run-time.
  + Ex: 
* Numbers
  + Number types are divided into two groups:
  + Integer types stores whole numbers, positive or negative (such as 123 or -456), without decimals. Valid types are int and long. Which type you should use, depends on the numeric value.
    - Int
      * In general, the int data type is the preferred data type when we create variables with a numeric value.
      * Ex:
    - Long
      * This is used when int is not large enough to store the value. Note that you should end the value with an "L":
      * Ex:
  + Floating point types represents numbers with a fractional part, containing one or more decimals. Valid types are float and double.
    - You should use a floating point type whenever you need a number with a decimal, such as 9.99 or 3.14515.
    - The float and double data types can store fractional numbers. Note that you should end the value with an "F" for floats and "D" for doubles:
    - Ex: 
    - Ex: 
    - A floating point number can also be a scientific number with an "e" to indicate the power of 10:
    - Ex: Text

      Description automatically generated
    - Output: 
  + Even though there are many numeric types in C#, the most used for numbers are int (for whole numbers) and double (for floating point numbers).
* Booleans
  + A boolean data type is declared with the bool keyword and can only take the values true or false:
  + Ex: Text

    Description automatically generated
  + Output: 
* Characters
  + The char data type is used to store a single character. The character must be surrounded by single quotes, like 'A' or 'c':
  + Ex: 
* Strings
  + The string data type is used to store a sequence of characters (text). String values must be surrounded by double quotes:
  + Ex: 

**Variables**

* A variable is a name of memory location. It is used to store data. It is a way to represent memory location through symbol so that it can be easily identified.
* All C# variables must be identified with unique names called identifiers
* The general rules for naming variables are:
  + A variable can have alphabets, digits and underscore.
  + A variable name can start with alphabet and underscore only. It can't start with digit.
  + No white space is allowed within variable name.
  + A variable name must not be any reserved word or keyword e.g. char, float etc. However, if we want to use the keywords as identifiers, we may prefix the keyword with @ character.
* To create a variable, you can specify the type and assign it a value
* Ex: 
* You can also declare a variable without assigning the value, and assign the value later:
* Ex: Graphical user interface

  Description automatically generated with medium confidence
* Note that if you assign a new value to an existing variable, it will overwrite the previous value:
* Ex: Text

  Description automatically generated with medium confidence

**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:
* 
* You cannot declare a constant variable without assigning the value. If you do, an error will occur: A const field requires a value to be provided.

Literals

* The constants refer to fixed values that the program may not alter during its execution. These fixed values are also called literals.
* Integer Literals
  + An integer literal can be a decimal, or hexadecimal constant. A prefix specifies the base or radix: 0x or 0X for hexadecimal, and there is no prefix id for decimal.
  + An integer literal can also have a suffix that is a combination of U and L, for unsigned and long, respectively. The suffix can be uppercase or lowercase and can be in any order.
  + Here are some examples of integer literals:
    - Chart, scatter chart

      Description automatically generated
    - Text

      Description automatically generated with medium confidence
* Floating-point Literals
  + A floating-point literal has an integer part, a decimal point, a fractional part, and an exponent part. You can represent floating point literals either in decimal form or exponential form.
  + Here are some examples of floating-point literals:
    - Text

      Description automatically generated
* String Literals
  + String literals or constants are enclosed in double quotes "" or with @"". A string contains characters that are similar to character literals: plain characters, escape sequences, and universal characters.
  + You can break a long line into multiple lines using string literals and separating the parts using whitespaces.
  + Here are some examples of string literals. All the three forms are identical strings.
  + Text

    Description automatically generated

**Type Casting**

* Type casting is when you assign a value of one data type to another type.
* In C#, there are two types of casting, implicit and explicit.
* Implicit Casting (automatically) - converting a smaller type to a larger type size
  + char -> int -> long -> float -> double
  + Implicit casting is done automatically when passing a smaller size type to a larger size type:
  + Text

    Description automatically generated
* Explicit Casting (manually) - converting a larger type to a smaller size type
  + double -> float -> long -> int -> char
  + Explicit casting must be done manually by placing the type in parentheses in front of the value:
  + Text

    Description automatically generated with medium confidence
* Type Conversion Methods
  + It is also possible to convert data types explicitly by using built-in methods, such as Convert.ToBoolean, Convert.ToDouble, Convert.ToString, Convert.ToInt32 (int) and Convert.ToInt64 (long):
  + Text

    Description automatically generated

**User Input**

* Use Console.ReadLine() to get user input. The Console.ReadLine() method returns a string.
* Ex: Text

  Description automatically generated

**Operators**

* Arithmetic Operators:
* A picture containing timeline

  Description automatically generated
* Assignment Operators:
* Table

  Description automatically generated
* Comparison Operators:
* Graphical user interface, text, application

  Description automatically generated
* Logical Operators:
* Graphical user interface, text, application

  Description automatically generated
* The precedence and associativity of C# operators is given below:
* Table

  Description automatically generated

**Math**

* The C# Math class has many methods that allows you to perform mathematical tasks on numbers.
* The Math.Max(x,y) method can be used to find the highest value of x and y
* The Math.Min(x,y) method can be used to find the lowest value of of x and y
* The Math.Sqrt(x) method returns the square root of x
* The Math.Abs(x) method returns the absolute (positive) value of x
* Math.Round() rounds a number to the nearest whole number

**Strings**

* In C#, the string keyword is an alias for String; therefore, String and string are equivalent. It's recommended to use the provided alias string as it works even without using System;
* A string in C# is actually an object, which contain properties and methods that can perform certain operations on strings.
* Immutability of strings
  + String is immutable, i.e. strings cannot be altered. When you alter a string (by adding to it for example), you are actually creating a new string. But StringBuilder is not immutable (rather, it is mutable), so if you have to alter a string many times, such as multiple concatenations, then use StringBuilder.
  + All of the String methods and C# operators that appear to modify a string actually return the results in a new string object
  + Ex:
    - Text

      Description automatically generated
    - When the contents of s1 and s2 are concatenated to form a single string, the two original strings are unmodified. The += operator creates a new string that contains the combined contents. That new object is assigned to the variable s1, and the original object that was assigned to s1 is released for garbage collection because no other variable holds a reference to it.
  + Because a string "modification" is actually a new string creation, you must use caution when you create references to strings. If you create a reference to a string, and then "modify" the original string, the reference will continue to point to the original object instead of the new object that was created when the string was modified. The following code illustrates this behavior:
  + Ex:
  + Text

    Description automatically generated
* String Methods
  + length of a string can be found with the Length property
  + Ex: 
  + There are many string methods available, for example ToUpper() and ToLower(), which returns a copy of the string converted to uppercase or lowercase
  + Ex: A picture containing text

    Description automatically generated
  + The + operator can be used between strings to combine them. This is called concatenation
  + Ex: Text

    Description automatically generated
  + You can also use the string.Concat() method to concatenate two strings:
  + Ex: Graphical user interface, text

    Description automatically generated
  + Another option of string concatenation, is string interpolation, which substitutes values of variables into placeholders in a string. Note that you do not have to worry about spaces, like with concatenation:
  + Ex:Graphical user interface

    Description automatically generated
  + You can access the characters in a string by referring to its index number inside square brackets [].
  + Ex: 
  + You can also find the position of the first occurrent of a specific character in a string, by using the IndexOf() method.
  + Ex: 
  + Another useful method is Substring(), which extracts the characters from a string, starting from the specified character position/index, and returns a new string.
  + Ex: A picture containing chart

    Description automatically generated
  + We can pass a second parameter to Substring which represents the length of the substring.
  + Ex: Text

    Description automatically generated
* Escape characters
  + Graphical user interface, application

    Description automatically generated
  + A picture containing application

    Description automatically generated
  + Ex: 
  + Output: Graphical user interface

    Description automatically generated with low confidence
* Null strings and empty strings
  + An empty string is an instance of a System.String object that contains zero characters. Empty strings are used often in various programming scenarios to represent a blank text field. You can call methods on empty strings because they're valid System.String objects. Empty strings are initialized as follows:
  + 
  + By contrast, a null string doesn't refer to an instance of a System.String object and any attempt to call a method on a null string causes a NullReferenceException. However, you can use null strings in concatenation and comparison operations with other strings. The following examples illustrate some cases in which a reference to a null string does and doesn't cause an exception to be thrown:
  + Text

    Description automatically generated
* Using stringBuilder for fast string creation
  + String operations in .NET are highly optimized and in most cases don't significantly impact performance. However, in some scenarios such as tight loops that are executing many hundreds or thousands of times, string operations can affect performance. The StringBuilder class creates a string buffer that offers better performance if your program performs many string manipulations. The StringBuilder string also enables you to reassign individual characters, something the built-in string data type doesn't support. This code, for example, changes the content of a string without creating a new string:
  + Text

    Description automatically generated
  + Text

    Description automatically generated
* Check if a String represents Number
  + To determine whether a string is a valid representation of a specified numeric type, use the static TryParse method that is implemented by all primitive numeric types and also by types such as DateTime and IPAddress. The following example shows how to determine whether "108" is a valid int.
  + Text

    Description automatically generated
  + If the string contains nonnumeric characters or the numeric value is too large or too small for the particular type you have specified, TryParse returns false and sets the out parameter to zero. Otherwise, it returns true and sets the out parameter to the numeric value of the string.

**If…Else**

* Graphical user interface, text, application

  Description automatically generated
* Note that we can exclude the curly braces if the condition only executes 1 statement.
* Ex:
* A picture containing logo

  Description automatically generated is equivalent to Text

  Description automatically generated with medium confidence
* Ternary Operator: A picture containing logo

  Description automatically generated

**Switch**

* Use the switch statement to select one of many code blocks to be executed.
* When C# reaches a break keyword, it breaks out of the switch block.
* The default keyword is optional and specifies some code to run if there is no case match.
* Graphical user interface, text, application, chat or text message

  Description automatically generated

**Loops**

* While Loop
* The while loop loops through a block of code as long as a specified condition is True
* Ex: Text

  Description automatically generated
* We can also remove curly braces if the while loop only executes 1 statement.
* Ex: 
* Output: 
* Do-While Loop
* The do/while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true. The loop will always be executed at least once, even if the condition is false, because the code block is executed before the condition is tested.
* Ex:
* Graphical user interface, text, application, chat or text message

  Description automatically generated
* For Loop
* When you know exactly how many times you want to loop through a block of code, use the for loop instead of a while loop.
* Text

  Description automatically generated with low confidence
* Statement 1 is executed (one time) before the execution of the code block.
* Statement 2 defines the condition for executing the code block.
* Statement 3 is executed (every time) after the code block has been executed.
* We can also remove curly braces if the for loop only executes 1 statement.
* Ex:
* Output: 
* Foreach Loop
* There is also a foreach loop, which is used exclusively to loop through elements in an array
* Ex: 
* Output: A picture containing text, orange, dark

  Description automatically generated
* Note that we cannot assign to the value of the foreach iteration variable.
* The break statement can also be used to jump out of a loop.
* The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.
* Goto
  + The C# goto statement is also known jump statement. It is used to transfer control to the other part of the program. It unconditionally jumps to the specified label.
  + It can be used to transfer control from deeply nested loop or switch case label.
  + Currently, it is avoided to use goto statement in C# because it makes the program complex.
  + Ex: Graphical user interface, text, application, email

    Description automatically generated
  + Output: Text

    Description automatically generated

**Arrays**

* Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.
* An array stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type stored at contiguous memory locations.
* Array is a reference type
* The Array class is the base class for all the arrays in C#. It is defined in the System namespace. The Array class provides various properties and methods to work with arrays.
* Declaration
  + To declare an array, define the variable type with square brackets:
  + Ex:
  + To insert values to it, we can use an array literal - place the values in a comma-separated list, inside curly braces:
  + Ex: 
  + Ex: 
  + In C#, there are different ways to create an array:
  + Ex: Text

    Description automatically generated
  + However, you should note that if you declare an array and initialize it later, you have to use the new keyword:
  + Ex: Text

    Description automatically generated
  + When you create an array, C# compiler implicitly initializes each array element to a default value depending on the array type. For example, for an int array all elements are initialized to 0.
* Accessing/Changing elements
  + You access an array element by referring to the index number.
  + Ex: A picture containing graphical user interface

    Description automatically generated
  + To change the value of a specific element, refer to the index number:
  + Ex: Text

    Description automatically generated with medium confidence
* Copying Arrays
  + You can copy an array variable into another target array variable. In such case, both the target and source point to the same memory location.
  + Ex:
    - Text

      Description automatically generated
    - Output: 3
* Array Methods:
  + To find out how many elements an array has, use the Length property:
  + Text

    Description automatically generated with medium confidence
  + Sort() sorts an array alphabetically or in an ascending order
  + Ex: Graphical user interface, text, application, chat or text message

    Description automatically generated
  + Ex: Text

    Description automatically generated
  + Other useful array methods, such as Min, Max, and Sum, can be found in the System.Linq namespace:
  + Ex: Text

    Description automatically generated
* Looping through an Array
  + Use a for loop or a foreach loop to loop through an array
  + Ex: 
  + Ex: 
* Multidimensional Arrays
  + If you want to store data as a tabular form, like a table with rows and columns, you need to get familiar with multidimensional arrays.
  + Table

    Description automatically generated
  + Arrays can have any number of dimensions. The most common are two-dimensional arrays (2D).
  + To create a 2D array, add each array within its own set of curly braces, and insert a comma (,) inside the square brackets:
  + 
  + The single comma [,] specifies that the array is two-dimensional. A three-dimensional array would have two commas: int[,,].
  + To access an element of a two-dimensional array, you must specify two indexes: one for the array, and one for the element inside that array.
  + Ex: 
  + Change Elements of a 2D Array:
  + Text

    Description automatically generated with medium confidence
  + You can easily loop through the elements of a two-dimensional array with a foreach loop:
  + Ex: 
  + Output: 
  + You can also use a for loop. For multidimensional arrays, you need one loop for each of the array's dimensions.
  + Also note that we have to use GetLength() instead of Length to specify how many times the loop should run:
  + Text

    Description automatically generated
* Jagged Arrays
  + A Jagged array is an array of arrays. You can declare a jagged array named scores of type int as:
  + 
  + Declaring an array, does not create the array in memory. To create the above array:
  + Text

    Description automatically generated
  + You can initialize a jagged array as:
  + 
  + Notice scores[0] is an array of 3 integers and scores[1] is an array of 4 integers.

File IO

* The File class from the System.IO namespace, allows us to work with files
* Ex: Text

  Description automatically generated with medium confidence
* The File class has many useful methods for creating and getting information about files. For example:
* Text

  Description automatically generated with low confidence
* In the following example, we use the WriteAllText() method to create a file named "filename.txt" and write some content to it. Then we use the ReadAllText() method to read the contents of the file:
* ExText

  Description automatically generated
* Output: 