**C# Basics**

**-----------------------------------------------------------------------------------**

**Section 1: Introduction  
1.1 Check Later (Quick Links to Learning path)**

<https://hexacorp.udemy.com/course/csharp-tutorial-for-beginners/learn/lecture/10803848#overview>

-------------------------------------------------------------------------------------------------

**Section 2: Introduction C# vs .NET Framework**

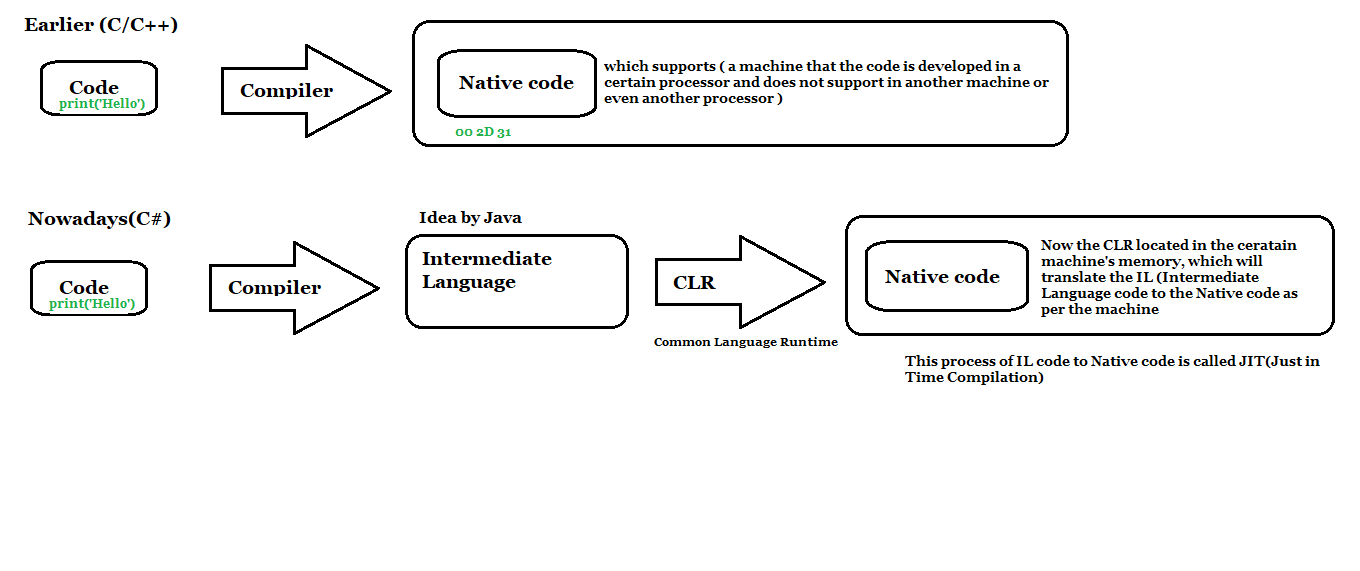
**2.1 C# vs .NET**

* C# is a programming language.
* .NET is a framework to build windows applications. Languages that use .NET framework are C#, F#, VB .NET.

**2.2 .NET Framework components**

* CLR (Common Language Runtime)
* Class Library

**CLR (Common Language Runtime):**

****

[**https://drive.google.com/file/d/1YhOBWdFQnPJCbZ-xjALGhJjHaseCtMcC/view**](https://drive.google.com/file/d/1YhOBWdFQnPJCbZ-xjALGhJjHaseCtMcC/view)

**Architecture of .NET Applications**

**A screenshot of a computer screen

Description automatically generatedhttps://drive.google.com/file/d/1giTJnMcDhIyWZNTDkTIK8Bg\_VRTqX0vt/view**

**-------------------------------------------------------------------------------------------------------------**

**Section 3: Primitive Types & Expressions:  
  
3.1 Variables and Constants**

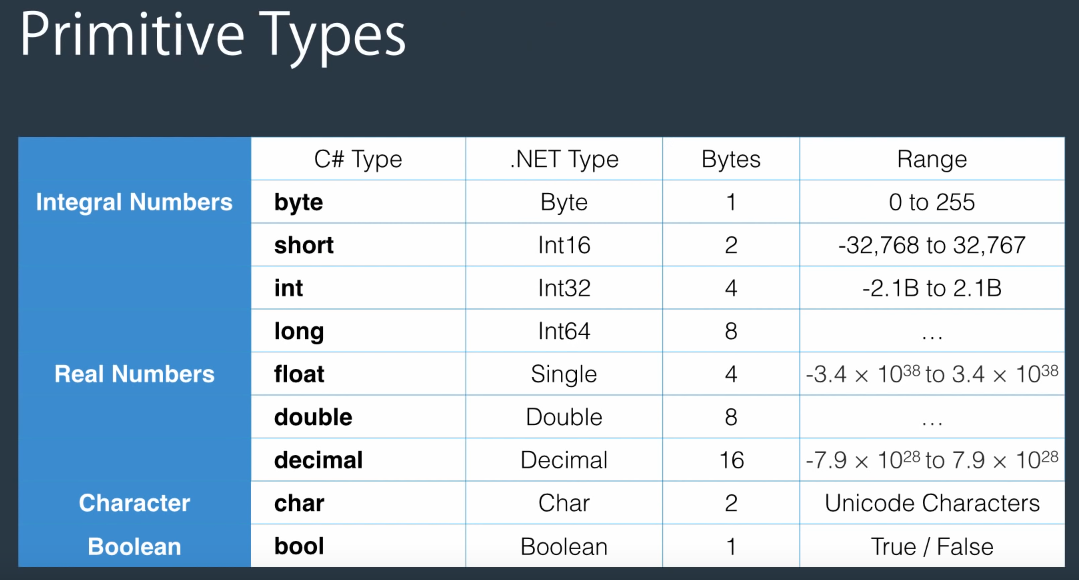
* **Variables :** a name given to a storage location in memory
* **Constant :** an **immutable** value
* Constants are used for safety in applications. Example Pi value : 3.14.
* Need to use const keyword for constants
* Variables are declared. Cannot be used before initialization.
* Identifiers cannot be started with numbers.
* Identifiers must be single word, no white space, use meaning names.

**Naming conventions:**

* **Camel Case:** firstName
* **Pascal Case:** FirstName
* **Hungarian Notation(C/C++) :** strFirstName

**Variable use Camel Case and Constant must be Pascal Case**

**3.2 Primitive Types**

****

**3.2.1 Real Numbers**

* By default the point values considered itself as double. So, we denote suffix **“f”** for float and **“m”** for decimal.

**Example:**

**float:** float number = 1.2f

**double:** double number = 1.2

**decimal:** decimal number = 1.2m

**3.3 Non-Primitive Types**

* String
* Array
* Enum
* Class

**3.4 Scope:**

* Where a variable/ constant has meaning

**3.5 Keywords:**

* Keywords are reserved words in C# (indicated by blue color in VSC and of lowercase)

**3.6 Format String (placeholders)**

* Console.WriteLine("{0} {1}", byte.MinValue, byte.MaxValue);

**3.7 Type Conversion**

* Implicit Type Conversion
* Explicit Type Conversion

**3.7.1 Implicit Type Conversion**

byte b= 1 00000001

int i = b 00000000 00000000 00000000 00000001

**3.7.2 Explicit Type Conversion (Also called Casting)**

int i = 1; 00000000 00000000 000000000 00000001

byte b = (byte) i; 00000001

**3.7.3 Converting Non-compatible types.**

string s = “1”;

int i = Convert.ToInt32(s); (To Byte(), ToInt16(), ToInt32(), ToInt64())

int j = int.Parse(s);

**3.8 Operators:**

* Arithmetic Operators – (+, -, \*, /, %, ++, --)
* Comparison Operators – (==, !=, >, >=, <, <=)
* Assignment Operators – (=, +=, -=, \*=, /=)
* Logical Operators – (&&, ||, !)
* Bitwise Operators - (&,, |)

Operator Precedence – a + b \* c (multiply, division has higher precedence)

**3.9 Comments:**

* Single-line comment - //
* Multi-line /\* \*/

**Section – 4 : Non-Primitive Types  
  
4.1 Classes:**

* Classes are building blocks of an application.
* It contains Fields and methods.
* It is blueprint for an object.
* Objects are instance of a class.

**Creating Objects:**

* Person person = new Person(); //new Person() to allocate memory
* person.Name = “Renith”;
* person.Introduce();
* CLR handles Garbage collection by which the objects are not in use automatically.

**Static Modifier:**

* For static classes, the object no need to be created to multiple memories can be allocated.
* The static modifier can be called by Class itself.  
    
  Example :
  + static void Main(string[] args) – only one Entry point
  + Console.WriteLine where Console is a class under System namespace.

Consider Person class. We can access by:  
Person.Introduce();

ClassName.Method

**Go to Basic\_Classes and Objects soln to see example**

**4.2 Structs (Structure are similar to classes):**

* Same as objects used for small, light-weight objects.
* It is value type where classes are reference types.
* It allocated memory in stack, where classes are in heap.
* Mostly it is memory efficient.

**Declaration:**

public **struct** RGB

{

public int Red;

public int Green;

public int Blue;  
}

**4.3 Arrays**

* A data structure to store collection of variables of the same type.
* Instead of declaring

int number1;

int number2;

int number3;

**int[] numbers = new int[3];**

By scenario, Array is a class. **new int[3]**  is creating instance of that class. So, we need to allocate memory size at the initialization itself.

Example **: int[] number = new int[3] {1,2,3}**

If the value is not assigned in array of integers. By default, is assigned to 0.

For boolean, the default is false.

**4.4 Strings**

* String is a sequence of characters.
* Strings are immutable.
* string firstName = “Renith” //string literal
* string name = string,Format(“{0} {1}”, firstName, lastName)
* **var numbers = new int[3] {1,2,3};  
  string list = string.Join(“,”, numbers);**

**Verbatim Strings**

* instead of string path = “c:\\projects\\project1\\folder1”;
* we use string path = @”c:\projects\project1\folder1”;

**4.5 Enums:**

* Enum is a datatype that represents set of name\value pairs (constants).

**Example**

Instead of

const int RegularAirMail=1

const int RegisteredAirMail =2

const int Express =3

we use

public enum ShippingMethod : byte

//by default enum in int,

//if want to convert data use :byte

{

RegularAirMail =1,

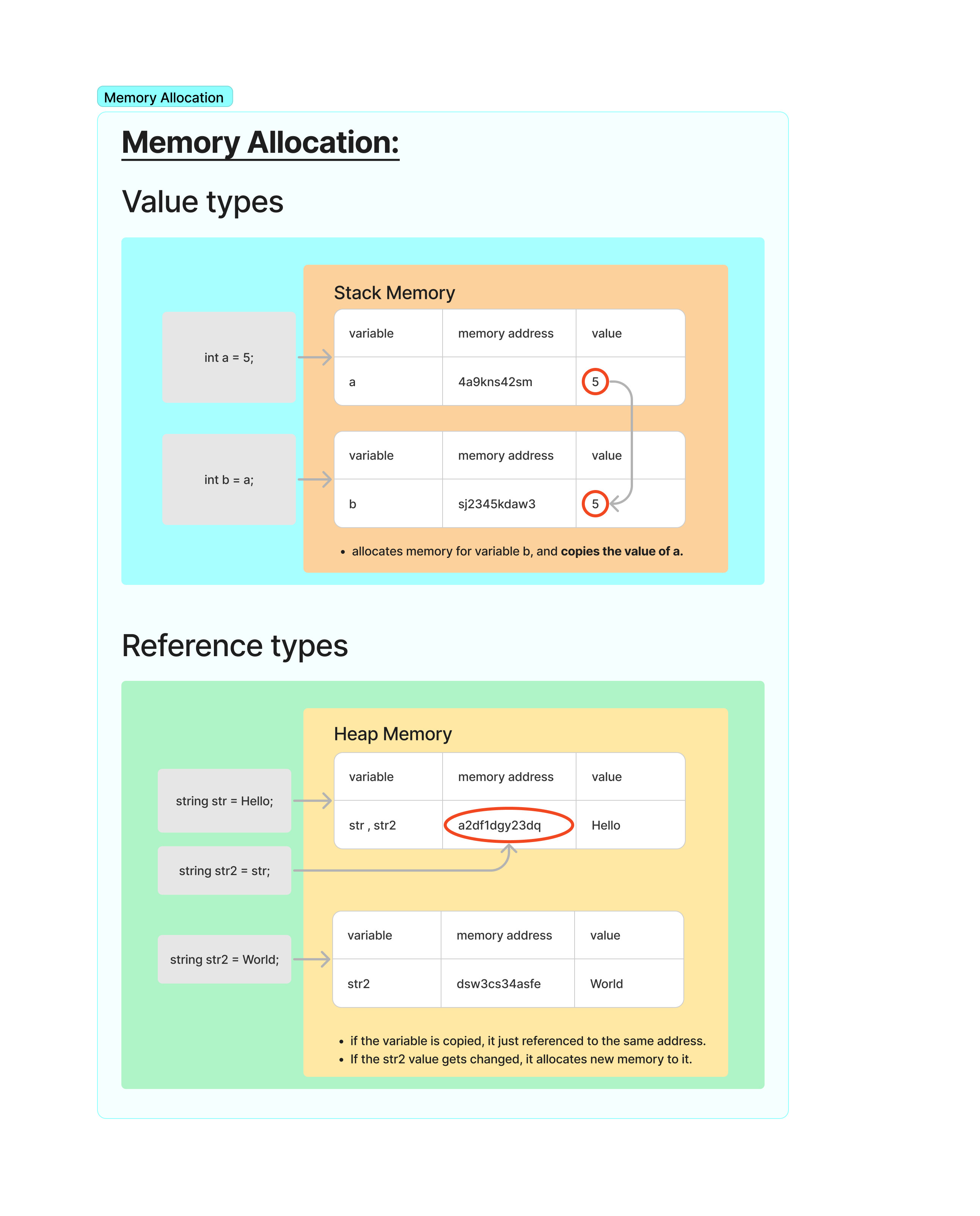
RegisteredAirMail=2,

Express=3;  
}

var method = ShippingMethod.Express;

**4.6 Value types vs Reference types**By default,

* All Primitive types like int, char, bool, float are structures. These are lesser than 8 bytes.
* Some Non-primitive types like array, strings are Classes.



**Structures**

* All primitive types (int, char, bool, float) are structures.
* Memory allocated in stack.
* Memory allocation done automatically.
* While run time, the CLR removes the variable memory, when it is goes out of scope.
* These are **Value types.**
* While assigning the value initialized value to another variable. The data gets copied.
* There is no relation between both variables.
* Use structures when small, light-weight objects need to be created.
* By Default, the structures that are predefined in C# are smaller than 16 bytes.
* Creating complex objects using structures results in obtaining larger memory and get lags.
* **Example:**
  + int a = 5; int b = a;
  + Now, the value 5 will be copied to b.
  + Changing the value of b does not make effect on a.

**Classes**

* Some Non-primitive types (Strings, Arrays) are classes.
* Memory allocated in heap.
* User need to allocate memory using **new** keyword.
* Garbage collection done automatically by run time or CLR. But it takes some time. CLR needs to confirm that the objects are no longer get used.
* These are **Reference types.**
* While assigning the value initialized to another variable. The address of that memory location (reference) will be stored in a heap.
* While using that variable the value stored in that address will be shown as output.
* The reference is stored in each variable gets assigned.
* **Example:**
  + string text = “Hello”;
  + text2 = text;
  + text2 will be referenced to same address where the “Hello” is stored.
  + If value for text2 is changed then it CLR allocates new memory to it.

**4.7 Mutable vs Immutable.**