**C#**

**Variables:**

**Variables** in C# must be declared before they can be used. This means, the name and type of variable must be known before they can be assigned a value. This is why C# is called a [statically-typed language](https://stackoverflow.com/questions/1517582/what-is-the-difference-between-statically-typed-and-dynamically-typed-languages).

**Scope:**

A scope can be thought as a block of code where the variable is visible or available to use.

**var keyword**

we can declare a variable without knowing its type using var keyword. Such variables are called **implicitly typed local variables**.

The compiler determines the type of variable from the value that is assigned to the variable.

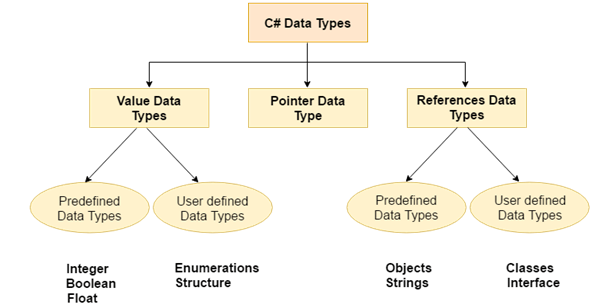
**Rules for Naming Variables in C#**

There are certain rules we need to follow while naming a variable. The rules for naming a variable in C# are:

1. The variable name can contain letters (uppercase and lowercase), underscore(\_ ) and digits only.
2. The variable name must start with either letter, underscore or @ symbol.
3. C# is case sensitive. It means age and Age refers to 2 different variables.
4. A variable name must not be a C# keyword. For example, if, for, using cannot be a variable name.

**Datatypes:**

A data type specifies the type of data that a variable can store such as integer, floating, character etc.



• A variable of value types directly contains only an object with the value.

**1) Predefined Data Types** - such as Integer, Boolean, Float, etc.

**2) User defined Data Types** - such as Structure, Enumerations, etc.

**Enum:**

Enum stands for enumeration, it is a set of named integer constants, their default integer values start with 0, we can also set any other sequence of the values. An enum is defined by using the enum keyword.

// C# example with explicit underlying type (byte)

enum Colors: byte {

Red,

Green,

Blue

}

Structures:

structure refers to the organization and layout of data in memory or within a data type. It involves how data is arranged, stored, and accessed.

example

// Defining a structure in C#

struct Point {

public int X;

public int Y;

}

class Program {

static void Main () {

// Creating instances of the Point struct

Point p1 = new Point {X = 1, Y = 2 };

Point p2 = new Point {X = 3, Y = 4 };

// Copying the entire struct when assigned

Point p3 = p1;

// Modifying p1 does not affect p3

p1.X = 10;

Console. WriteLine ($"p1: ({p1.X}, {p1.Y})"); // Output: p1: (10, 2)

Console. WriteLine ($"p3: ({p3.X}, {p3.Y})"); // Output: p3: (1, 2)

}

}

**Predefined datatypes:**

| **Alias** | **Type Name** | **Type** | **Size(bits)** | **Range** | **Default Value** |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| sbyte | System.Sbyte | signed integer | 8 | -128 to 127 | 0 |
| short | System.Int16 | signed integer | 16 | -32768 to 32767 | 0 |
| Int | System.Int32 | signed integer | 32 | -231 to 231-1 | 0 |
| long | System.Int64 | signed integer | 64 | -263 to 263-1 | 0L |
| byte | System.byte | unsigned integer | 8 | 0 to 255 | 0 |
| ushort | System.UInt16 | unsigned integer | 16 | 0 to 65535 | 0 |
| uint | System.UInt32 | unsigned integer | 32 | 0 to 232 | 0 |
| ulong | System.UInt64 | unsigned integer | 64 | 0 to 263 | 0 |
| float | System. Single | 32 | ±1.5 × 10-45 to ±3.4 × 1038 | | 0.0F |
| double | System. Double | 64 | ±5.0 × 10-324 to ±1.7 × 10308 | | 0.0D |
|  |  |  |  | |  |
| decimal | System. Decimal | 128 | ±1.0 × 10-28 to ±7.9228 × 1028 | | 0.0M |

**Reference types**

• A variable of reference type directly contains a reference to an object. Another variable may contain a reference to the same object.

If the data 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.

String : It represents a sequence of Unicode characters and its type name is Systemsyste.String. So, string and String are equivalent.

Example:

string s1 = "hello"; // creating through string keyword

String s2 = "welcome"; // creating through String class

**Object :** In C#, all types, predefined and user-defined, reference types and value types, inherit directly or indirectly from Object. So basically it is the base class for all the data types in C#.

Before assigning values, it needs type conversion. When a variable of a value type is converted to object, it’s called boxing. When a variable of type object is converted to a value type, it’s called unboxing. Its type name is System. Object.

int intValue = 42;

object boxed Value = intValue; // Boxing

int unboxed Value = (int)boxed Value; // Unboxing

**Command line arguments:**

**Command line arguments** are parameters that are passed to a program or script when it is executed from the command line or terminal. These arguments provide input to the program, allowing it to customize its behavior or process specific data. In most programming languages Command line arguments are parameters that are passed to a program or script when it is executed from the command line or terminal. These arguments provide input to the program, allowing it to customize its behavior or process specific data. In most programming languages, command line arguments are made available to the program through the argv array

**console** is a text-based environment where users can enter commands using a keyboard.

**Pointer Data Type**

The pointer in C# language is a variable, it is also known as locator or indicator that points to an address of a value.

Symbols used in pointer.

* & (ampersand sign) Address operator Determine the address of a variable.
* \* (asterisk sign) Indirection operator Access the value of an address.

Example:

using System;

class Program

{

unsafe static void Main()

{

// Declare a variable

int number = 42;

// Create a pointer to the variable

int\* pointer = &number;

// Print the value through the pointer

Console.WriteLine("Value through pointer: " + \*pointer);

// Modify the value through the pointer

\*pointer = 99;

// Print the updated value

Console.WriteLine("Updated value: " + number);

}

}

**Structures and classes differences:**

* Classes are reference types. When an instance of a class is created, a reference to the object is stored in memory, and variables of class type store this reference.
* Structures are value types. When an instance of a structure is created, the entire data is stored directly in the memory location where the variable is allocated.
* Memory for class instances is allocated on the heap, and objects are garbage-collected when they are no longer referenced.
* Memory for structure instances is typically allocated on the stack or as part of another object, and they are automatically deallocated when they go out of scope.
* Classes have a default constructor automatically generated if you do not provide one explicitly.
* Structures do not have a default constructor automatically generated if you provide any custom constructor. You need to explicitly define one if needed.
* Classes support inheritance structures do not.
* Classes can assign a null value but structures not because they have default values.

**OOPS**

| **Local Variable** | **Instance Variable** | **Static Variable** |
| --- | --- | --- |
| Defined within a method or a code block | Defined outside a method at the class level | Defined outside a method at the class level |
| Is only accessible in the method/code block where it is declared | Is accessible throughout the class | Is accessible throughout the class |
| Remains in memory as long as the method executes | Remains in memory as long as the object is in memory | Remains in memory as long as program executes |
| Does not require any special keyword | Does not require any special keyword but any access specifier (private, protected or public) can be specified. Typically, private or protected is used | Requires the static keyword to be specified. In addition, any access specifier (private, protected or public) can be specified. Typically, public is used |
| Requires to be initialized before it is used and called through object | Is given default value based on its data type, so does not require to be initialized before it is used.it called through object | Is given default value based on its data type, so does not require to be initialized before it is used.it called through class name only. |

**Access Modifiers**

C# supports 4 access modifiers.

private

protected

public

internal

protected internal.

**Public:** The public keyword allows its members to be visible from anywhere inside the project.

**Private:** The private members can only be accessed by the member within the same class.

**Protected:** Protected accessibility allows the member to be accessed from within the class and from another class that inherits this class.

**Internal:** Internal provides accessibility from within the project. Another similar internal accessibility is protected internal. This allows the same as the internal and the only difference is that a child class can inherit this class and reach its members even from another project.

**This keyword:**

When a method or constructor is called within a class, this refers to the current instance of that class. It allows you to distinguish between instance variables and parameters with the same name. this is also used to call other constructors within the same class. This is known as constructor chaining.

**Base keyword:**

In a derived class, base is used to explicitly access members (methods, properties, fields) of the base class.

base is also used in a derived class constructor to explicitly call a constructor of the base class.

It must be used only on first line of method when both this and base are used.

**Example:**

using System;

public class MyBaseClass {

private int baseValue;

public MyBaseClass(int value) {

this. base Value = value;

Console.WriteLine($"Base class constructor called with value: {this. base Value}");

}

public void DisplayBaseValue () {

Console.WriteLine($"Base class value: {this. baseValue}");

}

}

public class MyDerivedClass: MyBaseClass {

private int derived Value;

public MyDerivedClass (int base Value, int derived Value) : base(base Value) {

this. derived Value = derived Value;

Console.WriteLine($"Derived class constructor called with derived value: {this. derived Value}");

}

public void DisplayDerivedValue () {

Console.WriteLine($"Derived class value: {this. derived Value}");

}

public void DisplayCombinedValues() {

// Using 'base' to call the DisplayBaseValue method from the base class

base.DisplayBaseValue();

// Using 'this' to call the DisplayDerivedValue method within the derived class

this.DisplayDerivedValue();

}

}

class Program {

static void Main() {

// Creating an instance of the derived class

MyDerivedClass myObject = new MyDerivedClass(10, 20);

// Calling a method that uses both 'base' and 'this'

myObject.DisplayCombinedValues();

}

}

**POCO rule:**

In C#, "POCO" stands for Plain Old CLR Object. It's used to describe a simple, lightweight object in the context of object-oriented programming and the Common Language Runtime (CLR) in .NET.

POCO classes are commonly used in various data access scenarios, such as with Object-Relational Mapping (ORM) frameworks like Entity Framework, where you can define your data model using simple classes without having to inherit from specific framework classes. This allows for better separation of concerns and more flexibility in your codebase.

**Multiple inheritance**

C# provides a mechanism for achieving some level of multiple inheritance through interfaces. An interface defines a contract of methods and properties that a class can implement. A class can implement multiple interfaces, allowing it to inherit behavior from multiple sources.

// Define two interfaces

public interface IDisplayable

{

void Display();

}

public interface ILoggable

{

void Log();

}

// Implement the interfaces in a class

public class MyClass : IDisplayable, ILoggable

{

public void Display()

{

Console.WriteLine("Displaying...");

}

public void Log()

{

Console.WriteLine("Logging...");

}

}

class Program

{

static void Main()

{

// Create an instance of MyClass

MyClass myObject = new MyClass();

// Use methods from both interfaces

myObject.Display();

myObject.Log();

}

}

**Upcasting:**

Definition: Upcasting involves casting an object of a derived class to its base class.

Use: It is implicitly done by the compiler, and it's generally safe. Upcasting is used when you want to treat an object of a derived class as an object of its base class.

**Example:**

class Animal { }

class Dog : Animal { }

// Upcasting

Dog myDog = new Dog();

Animal myAnimal = myDog; // Implicit upcasting

**Downcasting:**

Definition: Downcasting involves casting an object of a base class to its derived class.

Use: It needs an explicit cast and is not always safe. Downcasting is used when you have an object of a base class, but you know it's actually an object of a derived class, and you want to access the specific features of that derived class.

**Example:**

class Animal { }

class Dog : Animal { }

// Upcasting

Animal my Animal = new Dog();

// Down casting (with explicit cast)

Dog my Dog = (Dog)my Animal;

Down casting can lead to runtime exceptions if the object being down casted is not actually an instance of the derived class. To avoid runtime errors, you can use the as operator and check for null

Dog my Dog = my Animal as Dog;

if (my Dog != null)

{

// Downcast successful

}

else

{

// Object is not an instance of Dog

}

Console object represents a terminal that uses system namespace

Writeline function represents a function or method in console and it print string to the console

A screenshot of a computer

Description automatically generatedDifference Between abstract class and Interface

| **S.No.** | **Git** | | **Github** |
| --- | --- | --- | --- |
|  | |  | |
| 1. | | Git is a software. | | GitHub is a service. |
| 2. | | Git is a command-line tool | | GitHub is a graphical user interface |
| 3. | | Git is installed locally on the system | | GitHub is hosted on the web |
| 4. | | Git is maintained by linux. | | GitHub is maintained by Microsoft. |
| 5. | | Git is focused on version control and code sharing. | | GitHub is focused on centralized source code hosting. |
| 6. | | Git is a version control system to manage source code history. | | GitHub is a hosting service for Git repositories. |
| 7. | | Git was first released in 2005. | | 8. | Git has no user management feature. | GitHub has a built-in user management feature. |
|  | |  | |  |
|  | |  | |  |

**.NET definition**

.NET is a free, open-source, cross-platform framework developed by Microsoft. It provides a runtime environment for managing and executing applications, as well as a comprehensive set of libraries and tools for building various types of software, including web, desktop, mobile, gaming, and IoT applications. .NET supports multiple programming languages, including C#, F#, and Visual Basic, allowing developers to choose the language that best suits their needs while still benefiting from the underlying framework's features and interoperability.

.NET is an abbreviation for the term "Network Enabled Technologies," but in the context of Microsoft's software framework, it does not directly stand for anything. It is a brand and framework developed by Microsoft for building and running applications.

**Framework:**

A framework is a software platform or structure that provides a foundation for developing and building applications. It includes a set of pre-written code, libraries, and tools that developers can use to streamline the process of creating software. Frameworks provide a standardized way of building and organizing code, enforcing best practices, and often include reusable components to help developers avoid repetitive tasks

**Git**

Open source free control version system

Version control means we can track our changes in the code by time to time and track down the bugs and also use previously code

**Terminal or command line:**

The application that runs on computer where u can type in texts

cli-command line interface

**Git and Github:**

[Yesterday 10:38 PM] Nikhita Palla

**Git and Github**

Version control system : tracks changes in code.

Git is popular, free,open source, fast and scalable

Features of git:

* To track the history of the project

Ex : while building a website, if we think to add extra features but that extra features have some issues, so we want to implement in the later version but the changes made in this version should be preserved and the new change which we left for next version should be build on top of the changes made in this version.

* To collobarate with other people so that it doesnt create any conflicts when two or more people work on the same module.

Github : website where developers store and manage the code.

We upload project in the form of folder. These folders are called as reposiotory.(repos)

Clone: git clone <link>

Status : git status

Change or modify file (untracked file) ---->  add file ( staged file) ----> commitg file (unchanged)

Add ( stage)

To add new or changed files into working directory in the staging area

Git add <file name>

Git add . ( to add all the files from the folder

Commit

To record the change

Git commit –m ‘message’

Push

To add files from the local computer to the repository

Local->repo

Git push origin <branch name>

Init

To create new git repository

Git init

Git remote add origin <link>

Git remote –v ( to verify origin)

Git branch ( to check branch)

Git branch –m rename\_branch\_name ( to rename branch name)

Git push origin rename\_branch\_name

Branches

Git branch ( to check in what branch we are)

Git branch –m new\_name

Git checkout another\_branch ( to go from one branch to another)

Git checkout –b new\_branch ( to move into new branch)

Git branch –d branch\_name ( to delete branch)

Merge

1)Git diff branch\_name  ( to compare commits,branches and everything that is changed in the branch)

2)Git merge main ( add the compared branch to main

Another method is to use pull request

Pull

Used to fetch and download content from the remote repository and immediately update the local repository to match the content with the remote repositpory

Git pull origin main