**AR/VR Developer Roadmap**

**1. Foundation Stage**

**1.1 Programming Basics (C#) :**

**Definition**

C# is a simple & powerful object-oriented programming language developed by Microsoft. C# can be used to create various types of applications, such as web, windows, console applications, or other types of applications using Visual studio.

**First Program**

****

**Image 1.1.1: HelloWorld Program in C#**

**Explanation**

* using System; imports the System namespace which contains fundamental classes like Console.
* namespace HelloWorldApp groups related classes.
* class Program defines a class named Program.
* static void Main(string[] args) is the entry point of the program where execution starts.
* Console.WriteLine("Hello World!"); prints "Hello World!" to the console.
* Console.ReadKey(); waits for a key press to prevent the console window from closing immediately after printing the message.

**Keywords in C#**

C# has a set of reserved words known as **keywords** that have special meanings in the language and cannot be used as identifiers (such as variable names or method names). There are about **79 keywords** in C#, all written in lowercase.

**Common C# Keywords**

* **Value types:** bool, byte, char, decimal, double, float, int, long, sbyte, short, uint, ulong, ushort
* **Control flow:** if, else, switch, case, for, foreach, while, do, break, continue, goto, return, yield, throw, try, catch, finally
* **Modifiers and access:** public, private, protected, internal, static, readonly, sealed, abstract, virtual, override, const, unsafe
* **Others:** class, struct, interface, enum, event, delegate, null, true, false, sizeof, typeof, using, namespace, new, this, base

**Contextual Keywords**

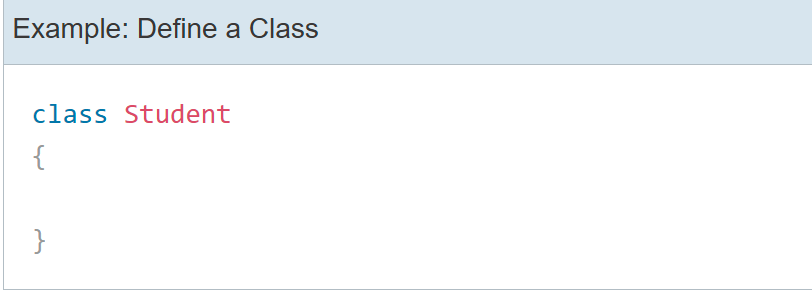
There are also **25 contextual keywords** that have special meanings only in certain contexts but can be used as identifiers otherwise. Examples include async, await, var, dynamic, partial, yield, when, etc.

These keywords are fundamental to C# programming and help define the language's syntax and structure. If needed, keywords can sometimes be used as identifiers by prefixing them with '@' (e.g., @class).

**Class in C#**

**Definition:** Class is a blueprint or template for creating objects. It defines the data (fields or properties) and behavior (methods) that the objects created from the class will have. A class acts as a logical structure and doesn't occupy memory by itself.

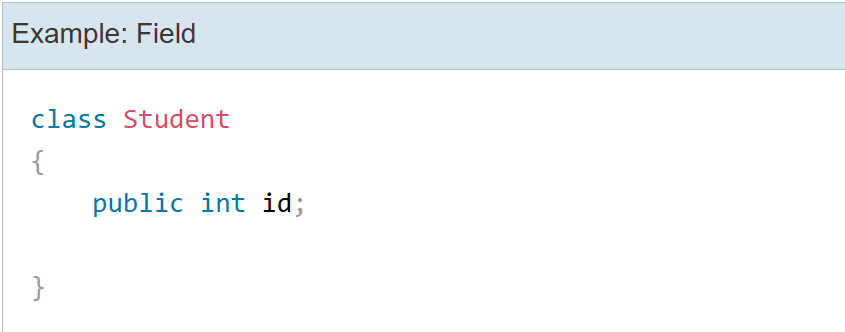
**Define a Class in c#:** In C#, a class can be defined by using the class keyword. Let's define a class named 'Student'.



**Image 1.1.1: defining Class in C#**

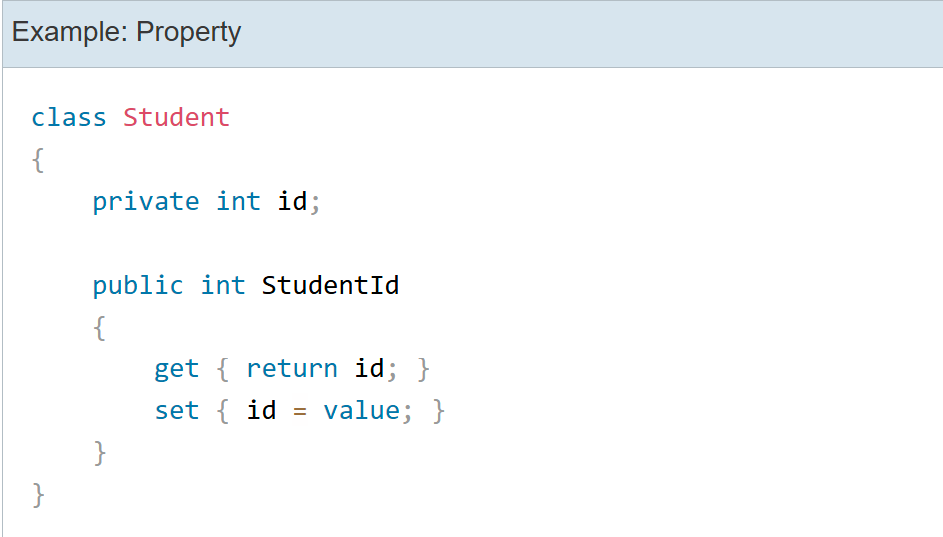
A class can contain one or more constructors, fields, methods, properties, delegates, and events. They are called class members. A class and its members can have access modifiers such as public, private, protected, and internal, to restrict access from other parts of the program.

**Field:** A class can have one or more fields. It is a class-level variable that holds a value. Generally, field members should have a private access modifier used with property.

****

**Image 1.1.2: defining Field in C#**

**Property:** In C#, a property is a member of a class that provides a flexible mechanism to read, write, or compute the value of a private field. Properties appear like public data members from outside the class but are implemented using special methods called **accessors** — specifically, a **get** accessor to retrieve a value and a **set** accessor to assign a value.

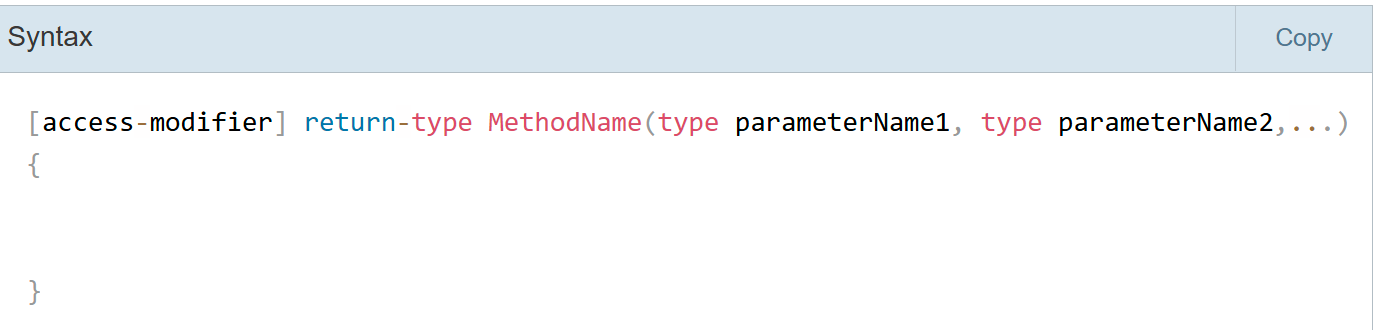
****

**Image 1.1.3: defining Property in C#**

In the above example, the id is a private field that cannot be accessed directly. It will only be accessed using the StudentId property. The get returns the value of the underlying field and set assigns the value to the underlying field id.

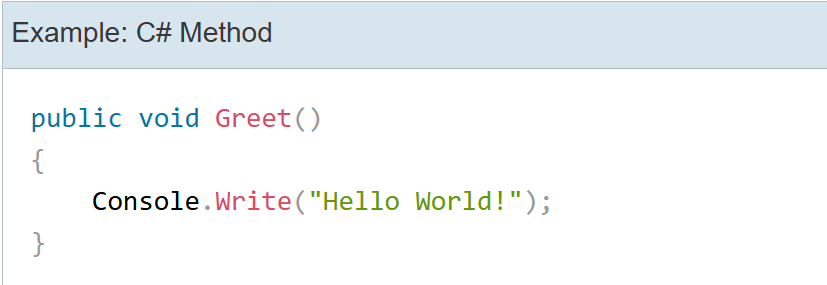
**Method:** A method can contain one or more statements to be executed as a single unit. A method may or may not return a value. A method can have one or more input parameters.

**Syntax:**

****

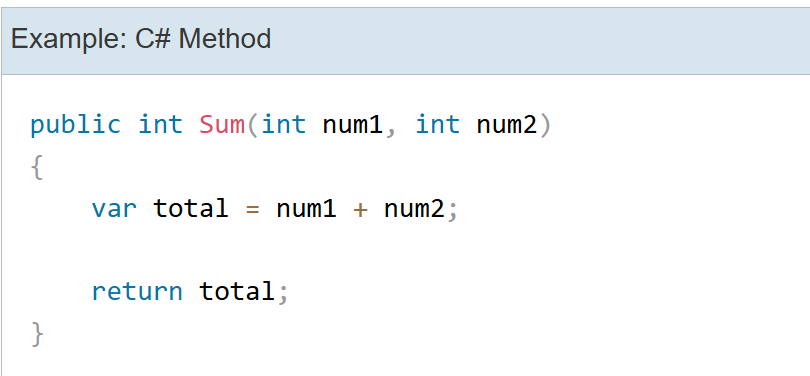
**Image 1.1.4: Syntax of Method in C#**

**Method without returning value:** The following method doesn't return anything and doesn't have any parameters. The return type is void.



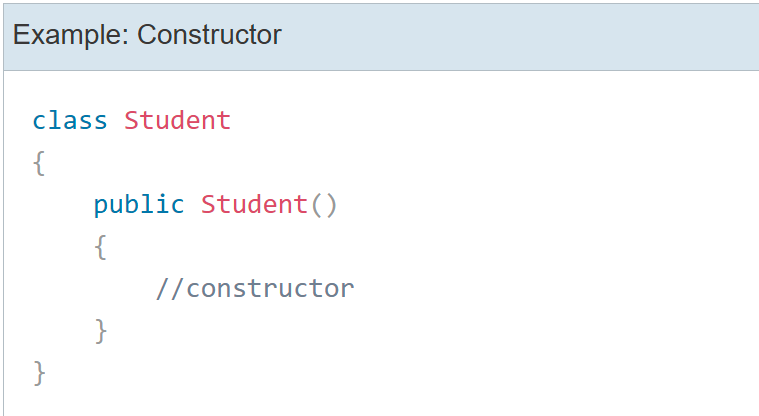
**Image 1.1.5: Method defined without returning any value in C#**

**Method with returning value:** The following defines the Sum method that returns the sum of two numbers.



**Image 1.1.6: Method defined with returning any value in C#**

**Constructor:** A constructor is a special type of method which will be called automatically when you create an Object of a class. A constructor is defined by using an access modifier and class name <access-modifier> <class-name>().



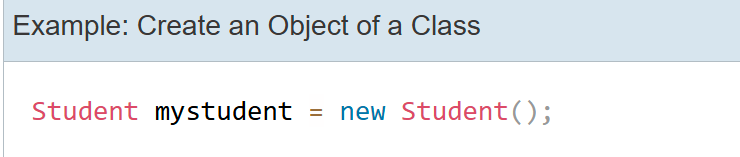
**Image 1.1.7 Example of constructor in C#**

**Note:**

* A constructor name must be the same as a class name.
* A constructor can be public, private, or protected.
* The constructor cannot return any value so cannot have a return type.
* A class can have multiple constructors with different parameters but can only have one parameterless constructor.
* If no constructor is defined, the C# compiler would create it internally.

**Object of a Class:** An **Object of class** in C# is a specific **instance** created based on the structure and behavior defined by that class. When an object is created, it represents a concrete entity with its own unique state and can perform behaviours (methods) defined by the class.

In C#, an object of a class can be created using the **new** keyword and assign that object to a variable of a class type. For example, the following creates an object of the **Student** class and assign it to a variable of the **Student** type.

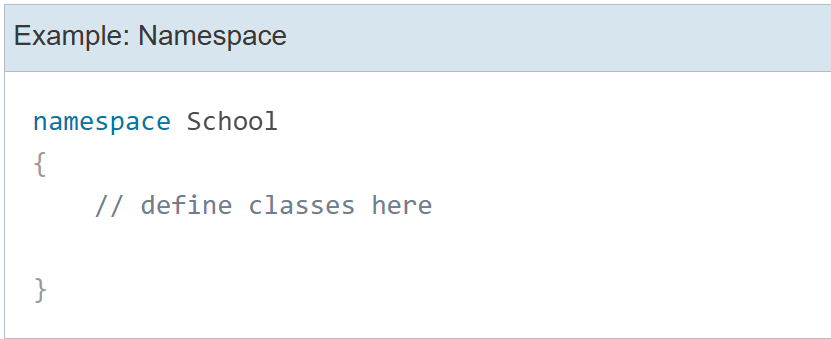


**Image 1.1.8: defining Object of a Class in C#**

**Namespace in C#**

**Definition:** A **namespace** is a **container that holds classes, structs, interfaces, enums, and other namespaces**. It is used to **organize code and avoid naming conflicts** in large programs.

In C#, a namespace can be defined using the namespace keyword.

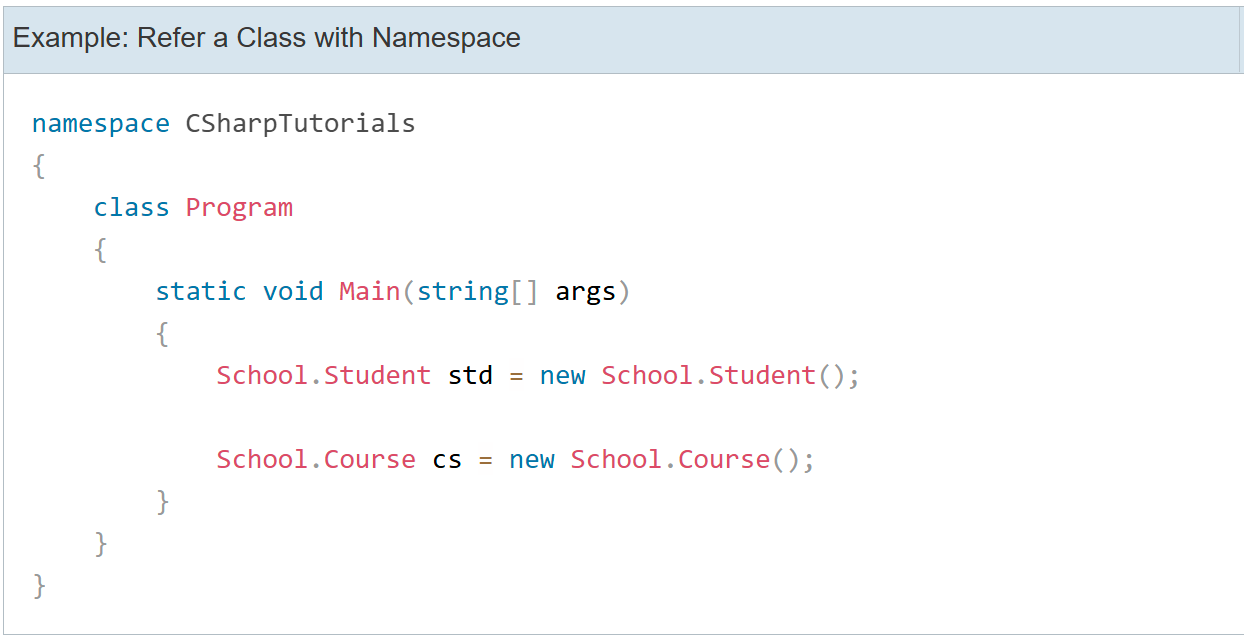


**Image 1.1.9 Syntax of namespace in C#**



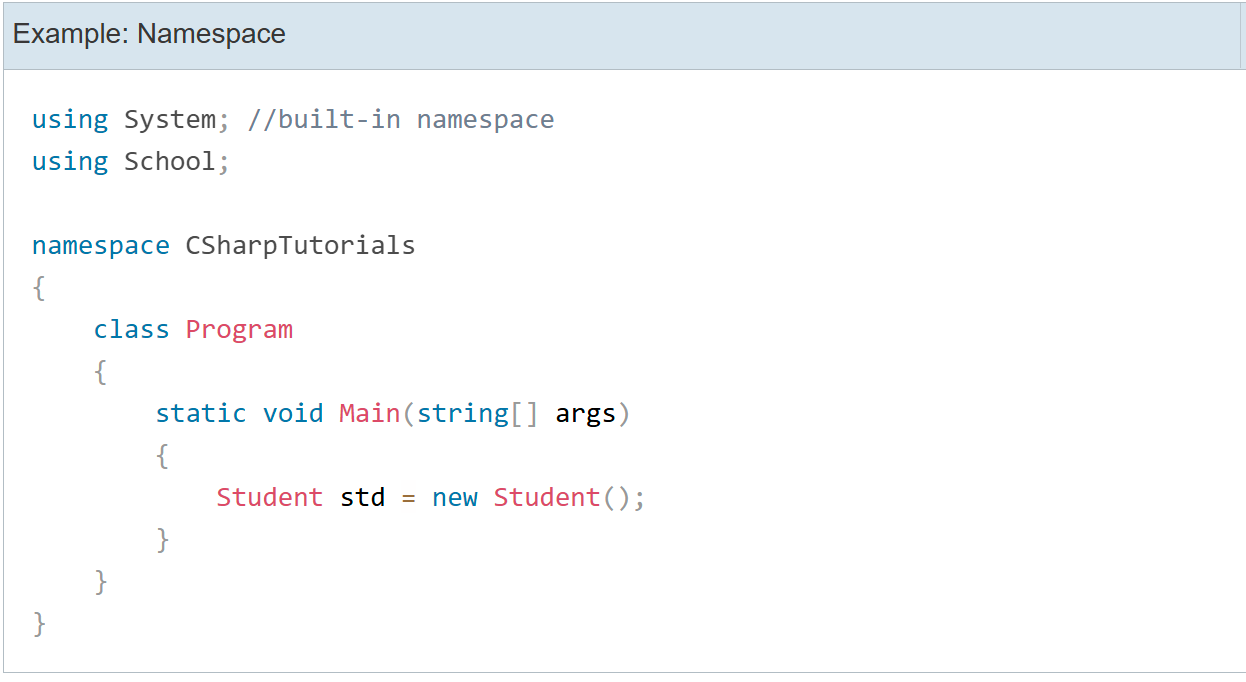
**Image 1.1.10 Example of Namespace in C#**

Classes under the same namespace can be referred to as namespace.classname syntax. For example, the Student class can be accessed as School.Student.

****

**Image 1.1.11 Example: Refer a class with namespace.**

To use classes under a namespace without the fully qualified name, import the namespace with the using keyword at the top of C# class file.

****

**Image 1.1.12 Example: Namespace with using keyword.**

**Variables in C#**

**Definition:** A variable is a container used to store data values during program execution. Each variable has a type, which determines the kind of data it can hold, such as numbers, text, or boolean values.

**Syntax:**

****

**Image 1.1.13 Example: Syntax of variable declaration.**

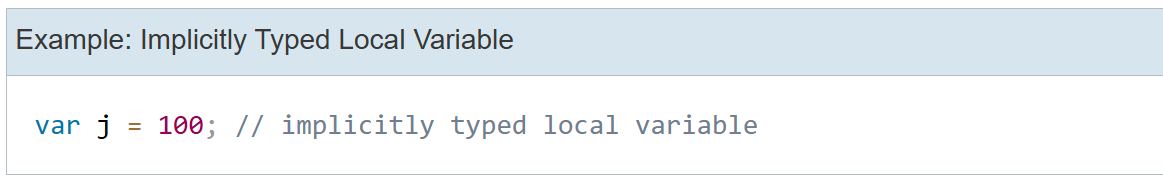
**The followings are naming conventions for declaring variables in C#:**

* Variable names must be unique.
* Variable names can contain letters, digits, and the underscore \_ only.
* Variable names must start with a letter.
* Variable names are case-sensitive,  num and Num are considered different names.
* Variable names cannot contain reserved keywords. Must prefix @ before keyword if want reserve keywords as identifiers.

**var in C#**

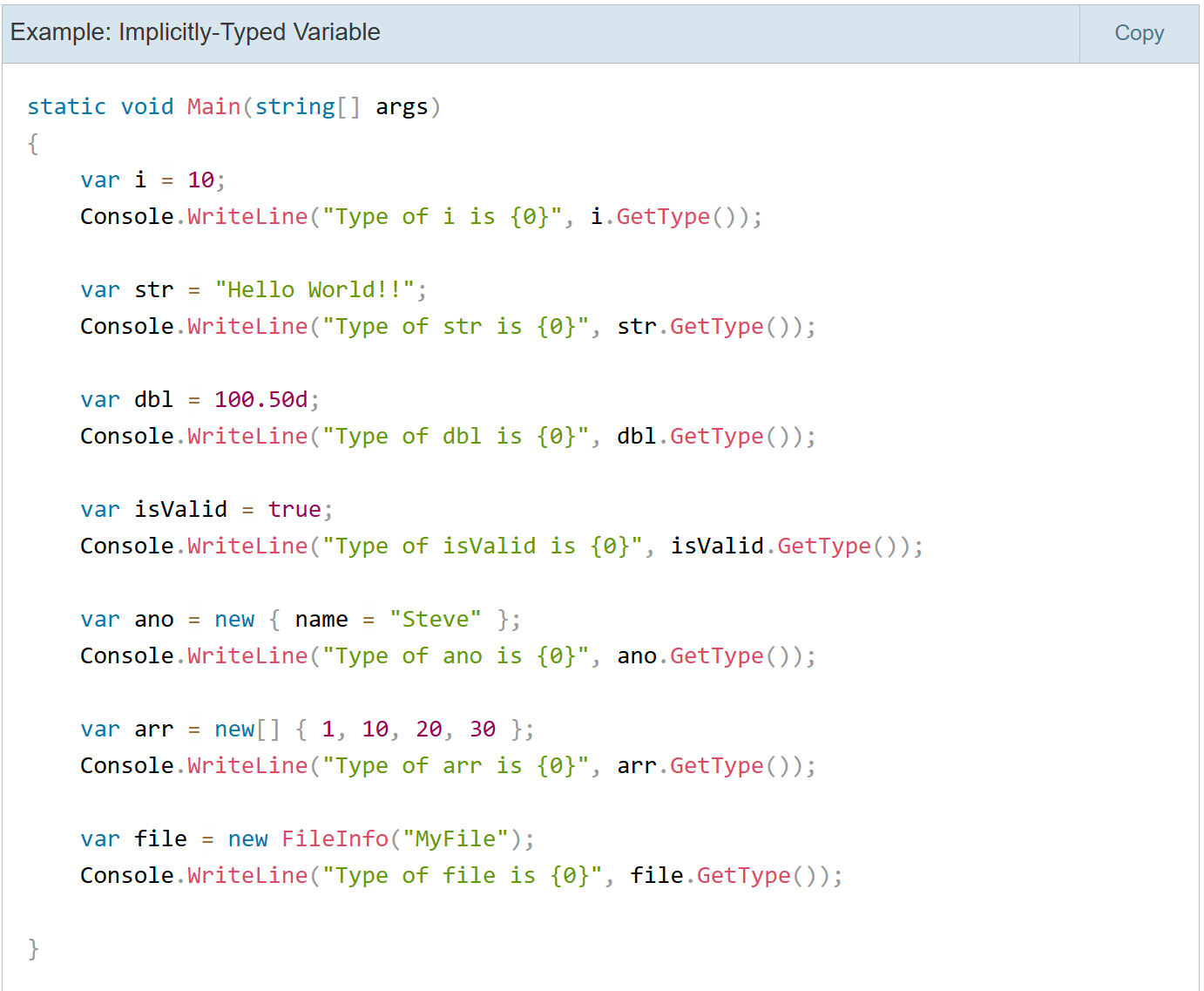
In C#, variables must be declared with the data type. These are called explicitly typed variables.

C# 3.0 introduced var keyword to declare method level variables without specifying a data type explicitly.

****

**Image 1.1.14 Example: declaration and initialization of var.**

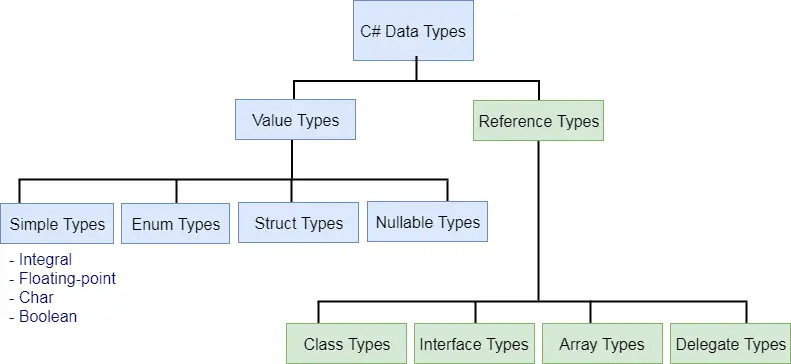
var can be used to declare any built-in data type or a user-defined type or an anonymous type variable. The following example shows C# compiler infers type based on the value:

****

**Image 1.1.15 Example: implicitly-typed variable**

**Data Types in C#**

**Definition:** A data types specifies the kind of data a variable can hold and determines the size and layout of the data in memory. C# supports a variety of built-in data types broadly categorized into a value types and reference types.

****

**Image 1.1.16 Types of data types**

**StringBuilder in C#**

In C#, **StringBuilder** is a class in the System.Text namespace designed for efficient manipulation of strings, especially when performing multiple modifications like appending, inserting, replacing, or removing text. Unlike regular strings in C# which are immutable (creating a new string with each modification), StringBuilder dynamically modifies the string content without creating new string objects, improving performance and reducing memory usage.

**Nullable Types in C#**

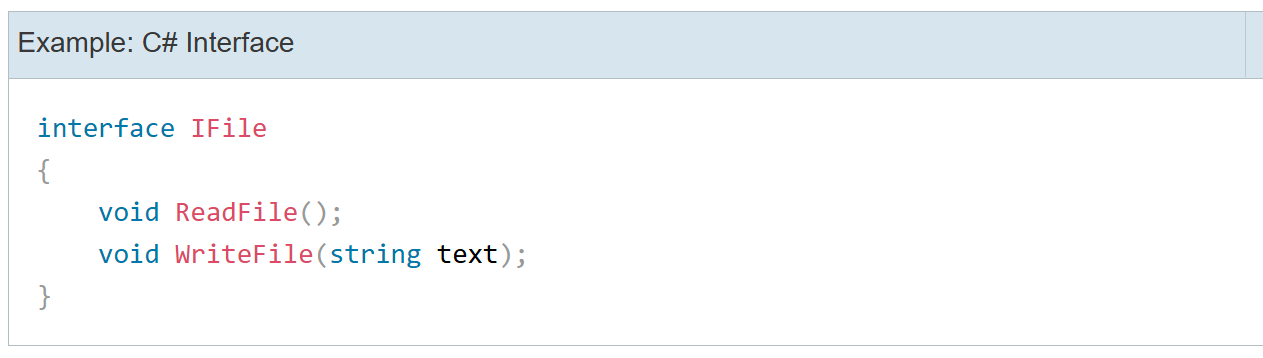
As you know, a value type cannot be assigned a null value. For example, int i = null will give you a compile time error.

C# 2.0 introduced nullable types that allow you to assign null to value type variables. You can declare nullable types using Nullable<t>where T is a type

**Interface in C#**

In C#, an interface is a contract or blueprint that defines a set of methods, properties, events, or indexers that a class or struct must implement. It specifies what a class must do, but not how to do it. Interfaces enforce a consistent API across different classes and enable multiple inheritance and polymorphism.

an interface can be defined using the interface keyword.

****

**Image 1.1.17 Declaration of Interface**

# if, else if, else Statements in C#

## C# if Statement: The if statement contains a boolean condition followed by a single or multi-line code block to be executed. At runtime, if a boolean condition evaluates to true, then the code block will be executed, otherwise not.

## Syntax:

## 

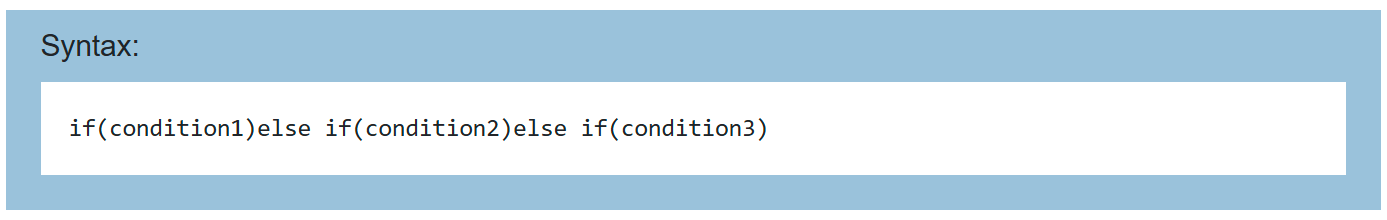
## Example:

## 

## Image 1.1.19 Example: if statement

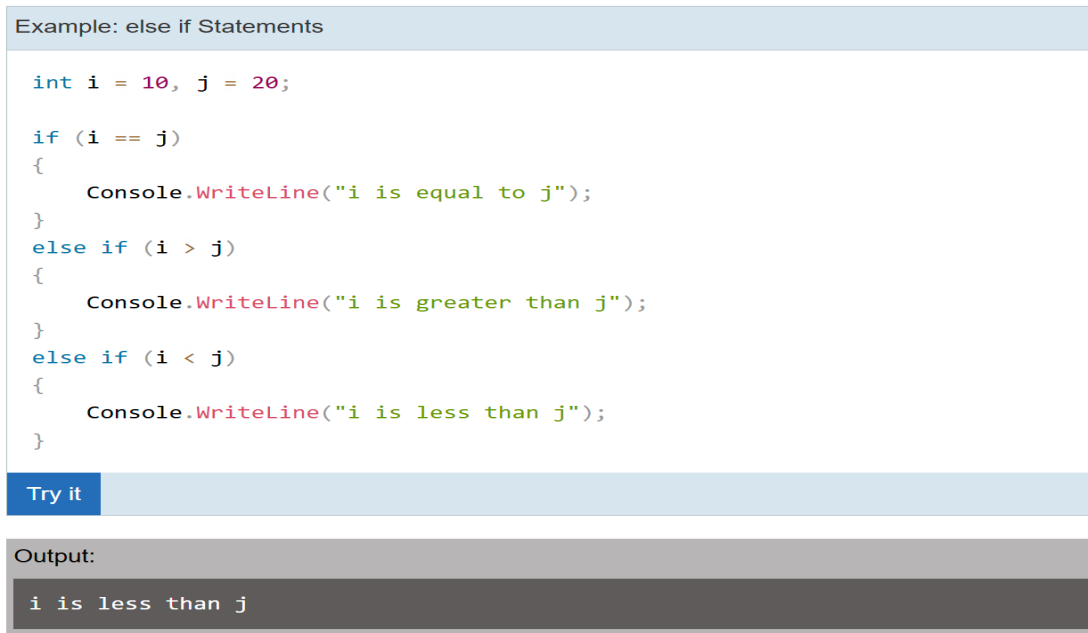
## else if Statement: Multiple else if statements can be used after an if statement. It will only be executed when the if condition evaluates to false. So, either if or one of the else if statements can be executed, but not both.

**Syntax:**



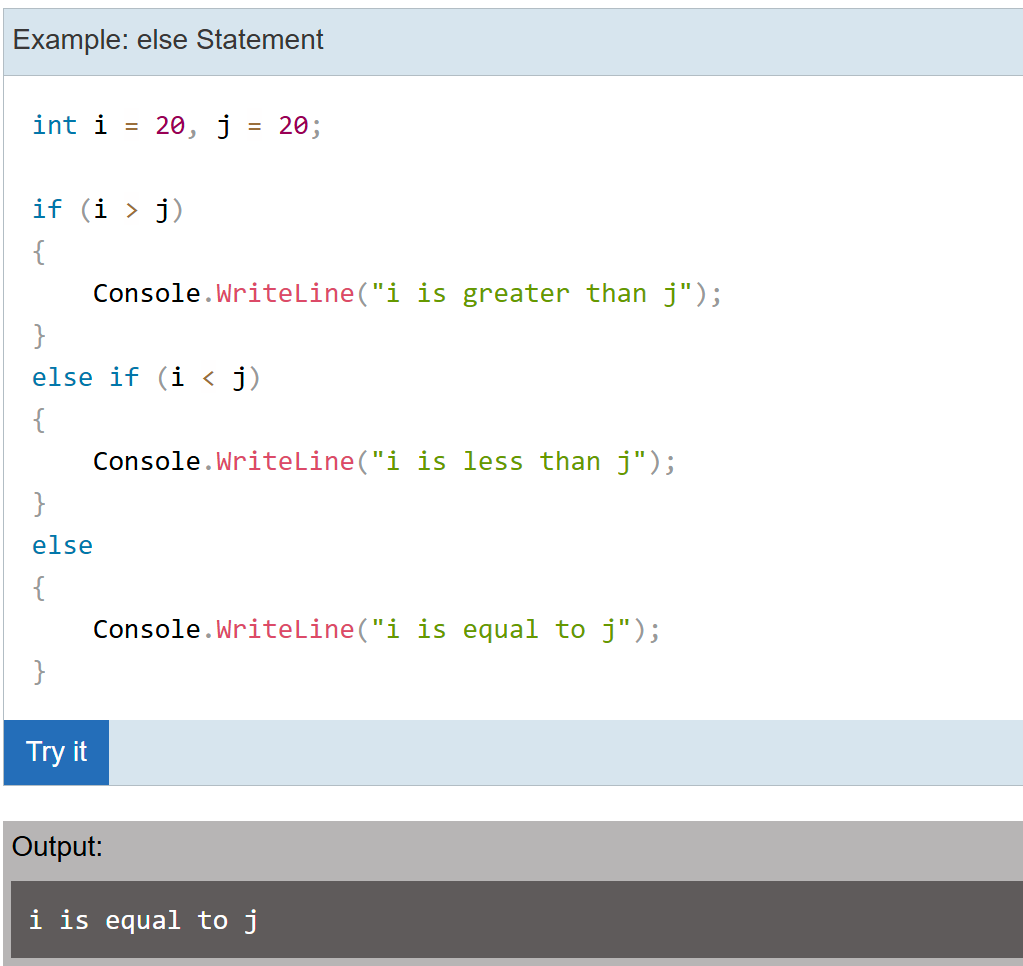
**Image 1.1.20 Syntax of else if Statement**

**Example:**

****

**Else Statement:** The else statement can come only after if or else if statement and can be used only once in the if-else statements. The else statement cannot contain any condition and will be executed when all the previous if and else if conditions evaluate to false.

**Example:**

****

**Image 1.1.22 Example of else statement**

# Ternary Operator in C#

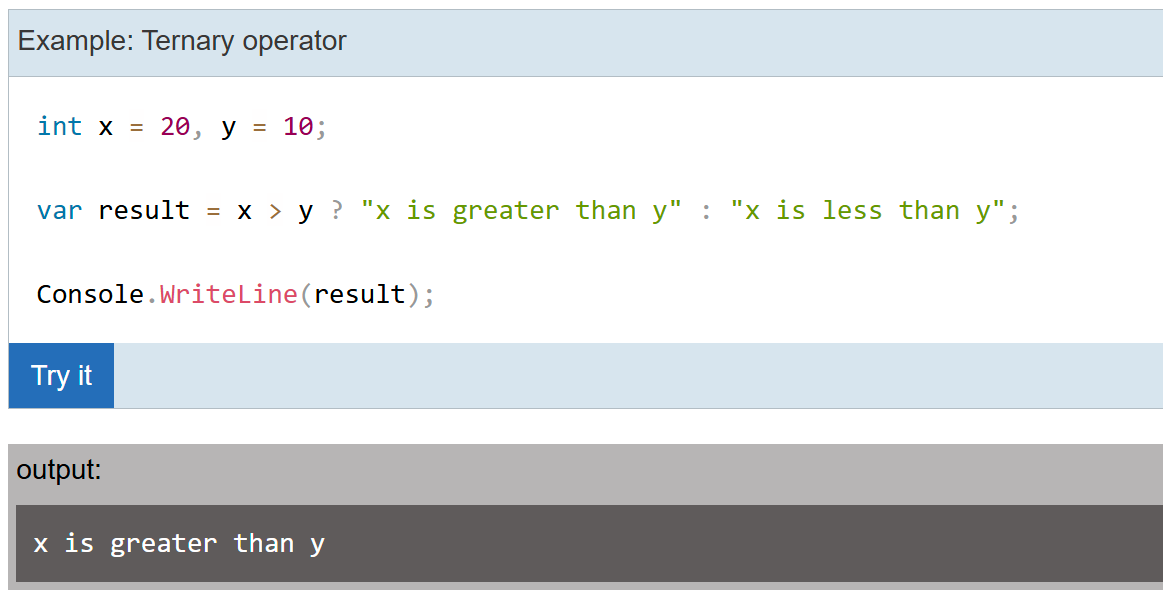
**Definition:** The Ternary Operator in C# is a concise way to write simple conditional statements. It is a shorthand for the if-else statement and is also known as the conditional operator. The ternary operator uses three operands and has the following syntax:

****

**Image 1.1.23 Syntax of Ternary Operator**

The ternary operator starts with a boolean condition. If this condition evaluates to true then it will execute the first statement after ?, otherwise the second statement after : will be executed.

**Example:**

****

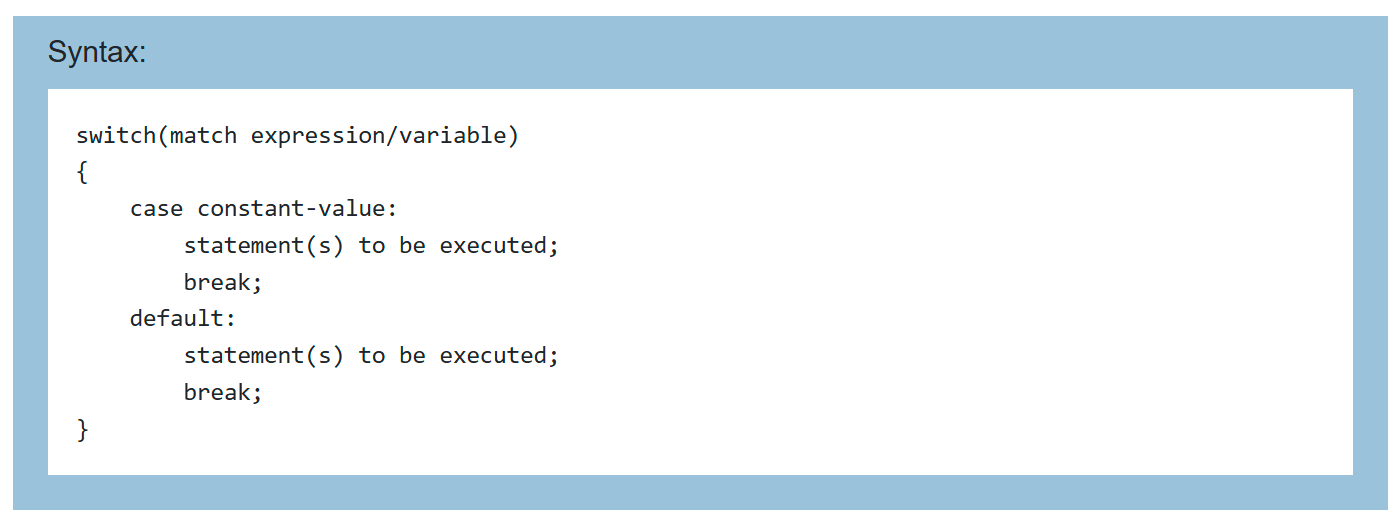
**Image 1.1.24 Example for Ternary Operator**

Above, a conditional expression x > y returns true, so the first statement after ? will be execute.

# Switch Statement in C#

**Definition:** The switch statement in C# is a control statement that allows selecting one of many code blocks to execute based on the value of an expression. It provides a cleaner alternative to multiple if-else statements when checking a variable against different constant values.

**Syntax:**



**Image 1.1.25 Syntax of switch statement**

The switch statement starts with the switch keyword that contains a match expression or a variable in the bracket switch(match expression). The result of this match expression or a variable will be tested against conditions specified as cases, inside the curly braces. A case must be specified with the unique constant value and ends with the colon :. Each case includes one or more statements to be executed. The case will be executed if a constant value and the value of a match expression/variable are equal. The switch statement can also contain an optional default label. The default label will be executed if no cases executed. The break, return, or goto keyword is used to exit the program control from a switch case.

**Example:**

****

Above, the switch(x) statement includes a variable x whose value will be matched with the value of each case value. The above switch statement contains three cases with constant values 5, 10, and 15. It also contains the default label, which will be executed if none of the case value match with the switch variable/expression. Each case starts after : and includes one statement to be executed. The value of x matches with the second case case 10:, so the output would be Value of x is 10.

# For Loop in C#

**Definition:** The **For loop** in C# is a control flow statement that allows you to execute a block of code repeatedly for a specified number of times. It is useful when the number of iterations is known beforehand.

**Syntax:**

****

**Image 1.1.27 Syntax of for loop**

The for loop contains the following three optional sections, separated by a semicolon:

**Initializer:** The initializer section is used to initialize a variable that will be local to a for loop and cannot be accessed outside loop. It can also be zero or more assignment statements, method call, increment, or decrement expression e.g., ++i or i++, and await expression.

**Condition**: The condition is a boolean expression that will return either true or false. If an expression evaluates to true, then it will execute the loop again; otherwise, the loop is exited.

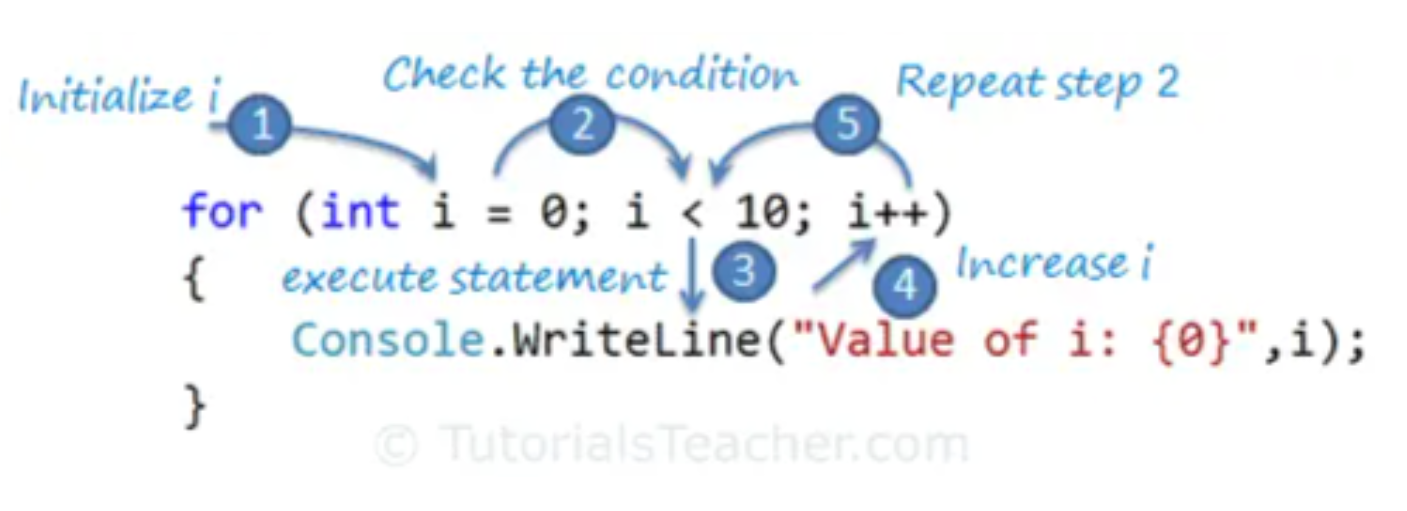
**Iterator**: The iterator defines the incremental or decremental of the loop variable.

**Example:**

In the example, int i = 0 is an initializer where we define an int variable i and initialize it with 0. The second section is the condition expression i < 10, if this condition returns true then it will execute a code block. After executing the code block, it will go to the third section, iterator. The i++ is an incremental statement that increases the value of a loop variable i by 1. Now, it will check the conditional expression again and repeat the same thing until conditional expression returns false.

****

**Image 1.1.28 Example of for loop**

****

**Image 1.1.29 Execution steps of for loop**

Practice object-oriented programming (classes, inheritance, events).