CSharp(C#)

**DataTypes**

CSharp Provides many datatypes to define the type of data we are going to store in a particular variable.It mainly having three type of datatype

* + - * ValueDataType
      * PointerDataType
      * ReferenceDataType

**ValueDataType: (Predefined Data type)**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| SL.NO | DATA  TYPE | SIZE | RANGE | TYPE | DEFAULT VALUE | USAGE |
| 1. | bool | 1 bit | true or false | System.Boolean | False | Used to store Boolean  value either true of false |
| 2. | byte | 1 byte | 0-255 | System.byte | 0 | Used to store data from 0- 255 |
| 3. | char | 2 byte | U +0000 to U +ffff | System.Char | Blank character(‘\u’) | Used to store singlecharacter |
| 4. | sbyte | 2 byte | -128-127 | System.SByte | 0 | used to store signed data |
| 5. | short | 2 byte | -32768 to 32767 | System.Int16 | 0 | Used to store signed int data |
| 6. | ushort | 2 byte | 0 to 65535 | System.UInt16 | 0 | Used to store unsigned interger value |
| 5. | int | 4 byte | -2,147,483,648 to 2,147,483,647 | System.Int32 | 0 | Used to store signed data range from -2,147,483,648 to 2,147,483,647 |
| 6. | uint | 4 byte | 0 to 232 | System.UInt32 | 0 | Used to store unsigned integer value |
| 7. | long | 8 byte | -263 to 263-1 | System.Int64 | 0 | Used to store larger interger value more than int |
| 8. | ulong | 8 byte | 0 to 263 | System.UInt64 | 0 | Used to store larger unsigned integer value |
| 9. | float | 4 byte | |  | | --- | | 1.5 \* 10-45 –  3.4 \* 1038,  7-digit prec  ision | |  | | System.Single | 0 | Used to store value with precision |
| 10. | double | 8 byte | 5.0 \* 10-324 - 1.7 \* 10308, 15-digit precision | System.Double | 0 | Used to store larger value than float |

**UserDefined Value Datatype:**

Structure

Enum

Enumerators:

Enumerations (or enums) are a set of integer values which have been assigned names. Used to replace magic numbers and for other purposes.

We can use any integer type to represent the value of an enum, and int is the default. When creating an enum, we can either specify the integer value for each name.

Example:

public enum WeekDays

{

SUNDAY,MONDAY,TUESDAY, WEDNESDAY,THURSDAY, FRIDAY,SATURDAY

}

In this example members of enum weekDays will have default the int value

from 1,2,3,4,..etc.

Structure:

A structure type(or struct) is a userdefined value type that, similarly to classes, encapsulates data and functionality. We use the struct keyword to define a struct.

Example:

public struct SampleStruct

{

public SampleStruct()

{

Console.WriteLine("Constructor from structure");

}

public void display()

{

Console.WriteLine("Method from display method");

}

}

We can access the properties from another class through the reference of the structure as ,

public static void Main(string[] args)

{

Console.WriteLine(WeekDays.SUNDAY);

int a =(int)WeekDays.TUESDAY;

Console.WriteLine("Index value of tuesday in enum : "+a);

SampleStruct s=new SampleStruct();

s.display();

}

**ReferenceDataType**

Reference data type has been classified into two types,that is

* + - * + PreDefinedDataType
        + UserDefinedDataType

Predefined DataType:

\* Object,String

Object and string has been defined by the developers while creating the application itself.so we are accessing the properties of that class using the reference of that type.

UserDefined DataType

\* class,interface

class:

class is reference data type where using reference we can access the properties of the particular class.

Interface:

Interface is the type in where we can declare only abstract methods.

Example:

//Example for class

public class ReferenceDataTypeEx{

int a = 10, b = 20;

public ReferenceDataTypeEx()

{

Console.WriteLine("This is Constructor");

}

public void add()

{

Console.WriteLine("Addition is: " + (a + b));

}}

We can implement the interface by providing the implementation in the derived class .We can access the properties of the class by creating the instance of class.

**KeyWords in CSharp:**

Keywords are predefined, reserved identifiers that have special meanings to the compiler. In c# we are having 78 keywords categorized like,

* + - Modifier keyword
    - Access Modifier keyword
    - Statement keywords
    - Method Parameter keywords

KeyWords

Method parameter Keyword

Access Modifier Keyword

Modifier Keyword

Statement Keyword

|  |
| --- |
| If,else,switch  Goto,continue  Switch,do,for,  Foreach,break,try,  Catch,finally,  Checked,  unchecked |

|  |
| --- |
| abstract,  override,  partial,extern,  new,  static,  sealed,  virtual,  unsafe,  const,readonly |

|  |
| --- |
| private  public  internal  protected internal  protected  sealed  volatile  static  partial  extern  base,this |

|  |
| --- |
| Param  In  Out  ref |

**Access Modifiers**

* **public**: This keyword allows access to the member or type from any code in the assembly or other assemblies that reference it.
* **private**: This keyword restricts access to the member or type to only the containing class.
* **protected**: This keyword allows access to the member or type only within the containing class or derived classes.
* **internal**: This keyword allows access to the member or type only within the same assembly.
* **protected internal**: This keyword allows access to the member or type within the same assembly, as well as from derived classes from other assembly.

**Statement keywords**

* **if**: This keyword is used to start a conditional statement that evaluates a Boolean expression.
* **else**: This keyword is used with **if** to create an alternative branch of execution if the Boolean expression is false.
* **switch**: This keyword is used to evaluate a single expression and select the appropriate branch of code to execute based on its value.
* **case**: This keyword is used with **switch** to specify a particular value to match.
* **default**: This keyword is used with **switch** to specify the branch of code to execute if no other case matches the value.
* **while**: This keyword is used to create a loop that executes while a Boolean expression is true.
* **do**: This keyword is used to create a loop that executes at least once and then continues to execute while a Boolean expression is true.
* **for**: This keyword is used to create a loop that executes a fixed number of times, based on a counter variable.
* **foreach**: This keyword is used to iterate over the elements of a collection or array.
* **break**: This keyword is used to exit a loop or switch statement early.
* **continue**: This keyword is used to skip the rest of the current iteration of a loop and start the next iteration.

**Classes and Objects**

* **class**: This keyword is used to define a new class.
* **new**: This keyword is used to create a new instance of a class or value type.
* **this**: This keyword refers to the current instance of a class or struct.
* **base**: This keyword refers to the base class of the current class.

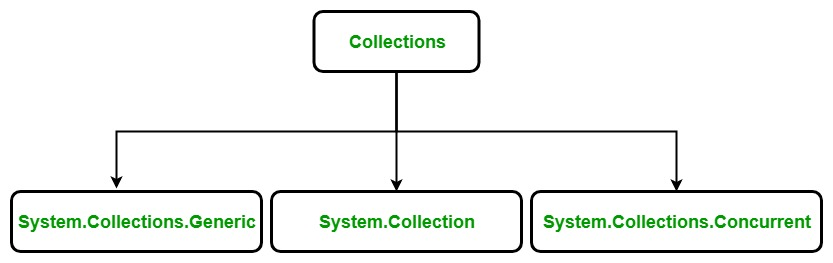
**Data Types and Variables**

* **bool**: This keyword represents a Boolean value that can be either **true** or **false**.
* **byte**: This keyword represents an 8-bit unsigned integer.
* **sbyte**: This keyword represents an 8-bit signed integer.
* **short**: This keyword represents a 16-bit signed integer.
* **ushort**: This keyword represents a 16-bit unsigned integer.
* **int**: This keyword represents a 32-bit signed integer.
* **uint**: This keyword represents a 32-bit unsigned integer.
* **long**: This keyword represents a 64-bit signed integer.
* **ulong**: This keyword represents a 64-bit unsigned integer.
* **float**: This keyword represents a single-precision floating-point number.
* **double**: This keyword represents a double-precision floating-point number.
* **decimal**: This keyword represents a decimal value with up to 28 significant digits.
* **char**: This keyword represents a single Unicode character.
* **string**: This keyword represents a sequence of Unicode characters.
* **object**: This keyword represents any type of object in C#.
* **var**: This keyword is used to implicitly declare a variable with its type inferred from the initialization expression.

**Method Parameters and Return Types**

* **void**: This keyword indicates that a method does not return a value.
* **params**: This keyword is used to allow a variable number of arguments to be passed to a method
* **ref**: The **ref** keyword is used for passing parameters by reference with read and write permission. It allows the called method to modify the original value of the argument passed.
* **out**: The **out** keyword is used for passing parameters by reference with write-only permission. It is used when the method needs to return multiple values. The **out** keyword ensures that the parameter is initialized before leaving the method.
* **in**: The **in** keyword is used for passing parameters by reference with read-only permission.

Collections:



1. [BlockingCollection<T>](https://learn.microsoft.com/en-us/dotnet/api/system.collections.concurrent.blockingcollection-1)
2. [ConcurrentDictionary<TKey,TValue>](https://learn.microsoft.com/en-us/dotnet/api/system.collections.concurrent.concurrentdictionary-2)
3. [ConcurrentQueue<T>](https://learn.microsoft.com/en-us/dotnet/api/system.collections.concurrent.concurrentqueue-1)
4. [ConcurrentStack<T>](https://learn.microsoft.com/en-us/dotnet/api/system.collections.concurrent.concurrentstack-1)
5. HashTable<Tkey,Tvalue>
6. ArrayList<TKey,TValue>
7. Stack<T>
8. Dictionary<Tkey,Tvalue>
9. List<T>
10. Queue<T>
11. SortedList<TKey,TValue>
12. Stack<T>
13. HashSet<T>

Collections.Generic :

|  |  |
| --- | --- |
| 1.Dictionary<Tkey,Tvalue> | It stores key/value pairs and provides functionality similar to that found in the non-generic Hashtable class. |
| 2.List<T> | |  | | --- | | It is a dynamic array that provides functionality similar to that found in the non-generic ArrayList class. | |  | |
| 3.Queue<T> | A first-in, first-out list and provides functionality similar to that found in the non-generic Queue class. |
| 4.Stack<T> | |  | | --- | | It is a first-in, last-out list and provides functionality similar to that found in the non-generic Stack class. | |  | |
| 5.HashSet<T> | |  | | --- | | It is an unordered collection of the unique elements. It prevent duplicates from being inserted in the collection. | |  | |
| 6.SortedList<TKey,TValue> | It is a sorted list of key/value pairs and provides functionality similar to that found in the non-generic SortedList class |

Collections:

|  |  |  |  |
| --- | --- | --- | --- |
| 1.HashTable | |  | | --- | | It represents a collection of key-and-value pairs that are organized based on the hash code of the key. | |  | |
| 2.ArrayList | It is a dynamic array means the size of the array is not fixed, it can increase and decrease at runtime. |
| 3.Stack | It is a linear data structure. It follows LIFO(Last In, First Out) pattern for Input/output. |

Collections.Concurrent:

|  |  |  |  |
| --- | --- | --- | --- |
| BlockingCollection | |  | | --- | | It provides blocking and bounding capabilities for thread-safe collections that implement IProducerConsumerCollection. | |  | |
| ConCurrentDictionary | It represents a thread-safe collection of key/value pairs that can be accessed by multiple threads concurrently. |
| ConcurrentQueue | It represents a thread-safe first in-first out (FIFO) collection |
| ConcurrentStack | It represents a thread-safe Last in-first out (LIFO) collection |

**Selenium with C#**

**IWebDriver Methods:**

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Description** | **Syntax** |
| Url | This command is used to open a specified URL in the browser. | driver.Url=”url”; |
| Title | This command is used to retrieve the page title of the web page that is currently open | String title = driver.Title |
| PageSource | This command is used to retrieve the source code of web page that is currently open. | String pageSource = driver.PageSource |
| Close | This command is used to close the recently opened browser instance. | driver.Close(); |
| Quit | This command is used to close all open browser instances | driver.Quit(); |
| Back | This command is used to navigate to the previous page of browser history. | driver.Navigate().Back(); |
| Forward | This command is used to navigate to the next page of browser history. | driver.Navigate().Forward() |
| Refresh | This command is used to perform browser refresh. | driver.Navigate().Refresh() |

**IWebElement Methods**

|  |  |  |
| --- | --- | --- |
| **Command Name** | **Description** | **Syntax** |
| Click | This command is used to click on a Webelement. For the element to be clickable, the element must be visible on the webpage. This command is used for checkbox and radio button operations as well. | element.Click(); |
| Clear | This command is specifically used for clearing the existing contents of textboxes. | element.Clear(); |
| SendKeys | This command is used to input a value onto text boxes. The value to be entered must be passed as a parameter to | element.SendKeys("value"); |
| Displayed | This command is used to identify if a specific element is displayed on the webpage. This command returns a Boolean value; true or false depending on the visibility of web element. | Boolean status = element.Displayed; |
| Enabled | This command is used to identify if a particular web element is enabled on the web page. This command returns a Boolean value; true or false as a result. | Boolean status = element.Enabled; |
| Selected | This command is used to identify if a particular web element is selected. This command is used for checkboxes,radio buttons, and select operations. | Boolean status = element.Selected; |
| Submit | This command is similar to click command, The difference lies in whether the HTML form has a button with the type Submit. While the click command clicks on any button, submit command clicks on the only the buttons with type submit. | element.submit(); |
| Text | This command returns the innertext of a Webelement. This command returns a string value as a result. | String text=element.Text; |
| TagName | This command returns the HTML tag of a web element. It returns a string value as the result. | String tagName = element.TagName; |
| GetCSSValue | This method is used to return the color of a web element on the form of a rgba string (Red,Green,Blue, and Alpha). | String color = element.getCSSValue; |

**ISearchContext Methods**

|  |  |
| --- | --- |
| **Name** | **Description** |
| [FindElement](https://www.selenium.dev/selenium/docs/api/dotnet/html/M_OpenQA_Selenium_ISearchContext_FindElement.htm) | Finds the first [IWebElement](https://www.selenium.dev/selenium/docs/api/dotnet/html/T_OpenQA_Selenium_IWebElement.htm) using the given method. |
| [FindElements](https://www.selenium.dev/selenium/docs/api/dotnet/html/M_OpenQA_Selenium_ISearchContext_FindElements.htm) | Finds all [IWebElements](https://www.selenium.dev/selenium/docs/api/dotnet/html/T_OpenQA_Selenium_IWebElement.htm) within the current context using the given mechanism. |