## Data types

| | |
|---|---|
| Int | Integer values like 1234, 10000 |
| Double | 64-bit floating-point, 3.145644 |
| Float | Floating point number, 3.1454 |
| String | Set of characters, “Welcome.” |
| Byte | 8bit unsigned integer |
| Char | 16 bit Unicode character, ‘A.’ |
| Long | 64 bit signed integer, -9.0789 |
| Decimal | High precision decimal numbers |
| Bool | True or false Boolean value |
| Enums | Value data type contains its value |
| Struct | value type that is used to represent a record |

### Initialisation of variables

```
int i = 7;
byte b = 255;
String s = “hackr.io”;
char c = ‘h’;
```

### Constant values

```
const String lastDayOfWeek = “Friday”;
```

## String Data type conversion

| Method | Description | Example |
|---|---|---|

| | | |
|---|---|---|
| AsInt(),<br>IsInt() | Convert string into integer<br>Check If the input is int | intVal = str.AsInt();<br>str.IsInt() |
| AsFloat(),<br>IsFloat() | Convert string into float<br>Check if the input is float | floatVal = str.AsFloat();<br>str.IsFloat() |
| AsDecimal()<br>IsDecimal() | Convert string into decimal<br>Check if input is decimal | decVal = str.AsDecimal();<br>str.IsDecimal() |
| AsDateTime()<br>IsDateTime() | Convert string into datetime type<br>Check if input is date-time | dateVal = str.AsDateTime();<br>str.isDateTime(); |
| AsBool()<br>IsBool() | Convert string into Boolean<br>Check if input is Boolean | boolVal = str.AsBool();<br>str.IsBool(); |
| ToString() | Convert another data type like int, array, list etc into String | myVal = 1111;<br>strVal = myVal.ToString(); |

## Operators

| Operator | Description |
|---|---|
| = | Assigns variable value. (i = 10) |
| + | Adds a value or variable. (i + j) or (i + 3) |
| - | Subtracts values or variables. (i – j) |
| * | Multiplies values or variables. (i*j) |
| / | Divides values or variables. (i/j) |
| += | Increments a variable. ( i+=1) |
| -= | Decrements a variable. (i-=1) |

| | |
|---|---|
| == | Equality. Returns true if values are equal. (i==10) |
| != | Inequality. Returns true if values are not equal. (I != 10) |
| < | Less Than (i < 5) |
| > | Greater Than (i > 5) |
| <= | Less Than or Equal to (i <= 5) |
| >= | Greater than equal to (i >= 5) |
| + | String concatenation (“Welcome to ” + websiteName) |
| . | Call methods, constant variables etc..<br>arrVal.ToString() |
| () | Calculations, passing parameters etc…<br>(i+10)*(i-10); multiply(i, j) |
| [] | Access values in arrays or collections. name[i] |
| ! | Reversing Boolean value<br>if (!isMatching) |
| && | Logical AND<br>if (isSingle && isMatching) |
| sizeof() | returns the size of a data type |
| typeof() | returns the type of object – string, integer etc… |

## String Operations

| **String Functions** | **Definitions** | **Example** |
|---|---|---|
| Clone() | Make clone of string. | str2 = str1.Clone() |

| | | |
|---|---|---|
| CompareTo() | Compare two strings and returns integer value as output. It returns 0 for true and 1 for false. | str2.CompareTo(str1) |
| Contains() | checks whether specified character or string is exists or not in the string value. | str2.Contains(“hack”); |
| EndsWith() | checks whether specified character is the last character of string or not. | str2.EndsWith(“io”); |
| Equals() | compares two string and returns Boolean value true as output if they are equal, false if not | str2.Equals(str1) |
| GetHashCode() | returns HashValue of specified string. | str1.GetHashCode() |
| GetType() | returns the System.Type of current instance. | str1.GetType() |
| GetTypeCode() | returns the Stystem.TypeCode for class System.String. | str1.GetTypeCode() |
| IndexOf() | Returns the index position of first occurrence of specified character. | str1.IndexOf(“:”) |
| ToLower() | Converts String into lower case based on rules of the current culture. | str1.ToLower(); |
| ToUpper() | Converts String into Upper case based on rules of the current culture. | str1.ToUpper(); |
| Insert() | Insert the string or character in the string at the specified position. | str1.Insert(0, “Welcome”);<br>str1.Insert(i, “Thank You”); |
| IsNormalized() | Check whether this string is in Unicode normalization form | str1.IsNormalized() |
| LastIndexOf() | Returns the index position of last occurrence of specified character. | str1.LastIndexOf(“T”); |
| Length | returns length of string. | str1.Length; |

| | | |
|---|---|---|
| Remove() | deletes all the characters from beginning to specified index position. | str1.Remove(i); |
| Replace() | replaces the specified character with another | str1.Replace('a', 'e'); |
| Split() | This method splits the string based on specified value. | str1 = "Good morning and Welcome";<br>String sep = {"and"};<br>strArray = str1.Split(sep, StringSplitOptions.None); |
| StartsWith() | Checks whether the first character of string is same as specified character. | str1.StartsWith("H") |
| Substring() | This method returns substring. | str1.Substring(1, 7); |
| ToCharArray() | Converts string into char array. | str1.ToCharArray() |
| Trim() | It removes extra whitespaces from beginning and ending of string. | str1.Trim(); |

## Modifiers

| | |
|---|---|
| public | field or function accessible by any other code in the same assembly or another assembly that references it |
| private | Only available by code in the same class or struct |
| protected | Only accessible by code in the same class or struct or a derived class |
| internal | Accessible by any code in the same assembly, but not from another assembly |
| protected internal | Accessible by any code in the same assembly, or by any derived class in another assembly |
| abstract | to indicate a class that is intended only to be a base class of other classes (has to be extended by other classes) |
| async | Indicates that the modified method, lambda expression, or anonymous method is asynchronous |

| | |
|---|---|
| const | Specifies that the value of the field or the local variable cannot be modified (constant) |
| event | Declares an event |
| extern | Indicates that the method is implemented externally |
| new | Explicitly hides a member inherited from a base class |
| override | Provides a new implementation of a virtual member inherited from a base class |
| partial | Defines partial classes, structs, and methods throughout the same assembly |
| read-only | Declares a field that can only be assigned values as part of the declaration or in a constructor in the same class |
| sealed | Specifies that a class cannot be inherited |
| static | Declares a member that belongs to the type itself instead of to a specific object, e.g., for static class or method, no object needs to be created |
| unsafe | Declares an unsafe context |
| virtual | Declares a method or an accessor whose implementation can be changed by an overriding member in a derived class |
| volatile | Indicates that a field can be modified in the program by something such as the operating system, the hardware, or a concurrently executing thread |

## Date/Time formatting

| | |
|---|---|
| DateTime dt = new DateTime();<br>dt.ToString(); | gives output as –<br>01-01-0001 00:00:00 |
| dt = DateTime.Now; | gives current date and time |
| dt = new DateTime(yyyy, MM, dd); | gives the specified date in yyyy-MM-dd format. Time wi be 00:00:00 |
| dt = new DateTime(yyyy, MM, dd, hh, min, ss); | gives specified date and time in the 24-hour format |

| | |
|---|---|
| dt = new DateTime(yyyy, MM, dd, hh, mm, ss);<br>dt1 = dt.Date; | gives only the date, with the time part set to 00:00:00 |
| DateTime.Now.ToShortDateString() | prints only the date part by completely omitting the time part |
| DateTime.Now.ToLongDateString() | prints the whole date and time based on region, month is printed in letters (JAN, FEB etc.. ) rather than numbe (01, 02) |

## DateTime format specifiers

| Format specifier | Name | Description |
|---|---|---|
| d | Short date pattern | Represents a custom **DateTime** format string defined by the current ShortDatePattern property.<br>For example, the custom format string for the invariant culture is "MM/dd/yyyy." |
| D | Long date pattern | Represents a custom **DateTime** format string defined by the current LongDatePattern property.<br>For example, the custom format string for the invariant culture is "dddd, dd MMMM yyyy." |
| f | Full date/time pattern (short time) | Represents a combination of the long date (D) and short time (t) patterns, separated by a space. |
| F | Full date/time pattern (long time) | Represents a custom **DateTime** format string defined by the current FullDateTimePattern property.<br>For example, the custom format string for the invariant culture is "dddd, dd MMMM yyyy HH:mm: ss." |
| g | General date/time pattern (short time) | Represents a combination of the short date (d) and short time (t) patterns, separated by a space. |

| | | |
|---|---|---|
| G | General date/time pattern (long time) | Represents a combination of the short date (d) and long time (T) patterns, separated by a space. |
| M or m | Month day pattern | Represents a custom **DateTime** format string defined by the current MonthDayPattern property.<br><br>For example, the custom format string for the invariant culture is "MMMM dd." |
| o | Round-trip date/time pattern | Represents a custom **DateTime** format string using a pattern that preserves time zone information. The pattern is designed to round-trip **DateTime** formats, including the **Kind** property, in text. Then th formatted string can be parsed back using Parse or ParseExact wit the correct **Kind** property value.<br><br>The custom format string is "yyyy'-'MM'-'dd'T'HH':' mm': 'ss.fffffffK."<br><br>The pattern for this specifier is a defined standard. Therefore, it is always the same, regardless of the culture used or the format provider supplied. |
| R or r | RFC1123 pattern | Represents a custom **DateTime** format string defined by the current RFC1123Pattern property. The pattern is a defined standard, and the property is read-only. Therefore, it is always the same regardless of the culture used, or the format provider supplied.<br><br>The custom format string is "DDD, dd MMM yyyy HH':' mm': 'ss 'GMT'".<br><br>Formatting does not modify the value of the **DateTime** object that is being formatted. Therefore, the application must convert the value to Coordinated Universal Time (UTC) before using this format specifier. |
| s | Sortable date/time pattern; conforms to ISO 8601 | Represents a custom **DateTime** format string defined by the current SortableDateTimePattern property. This pattern is a defined standard, and the property is read-only. Therefore, it is always the same regardless of the culture used, or the format provider supplied.<br><br>The custom format string is "yyyy'-'MM'-'dd'T'HH':'mm': 'ss." |
| t | Short time pattern | Represents a custom **DateTime** format string defined by the current ShortTimePattern property. |

| | | |
|---|---|---|
| | | For example, the custom format string for the invariant culture is "HH:mm." |
| T | Long time pattern | Represents a custom **DateTime** format string defined by the current LongTimePattern property.<br>For example, the custom format string for the invariant culture is "HH:mm: ss". |
| u | Universal sortable date/time pattern | Represents a custom **DateTime** format string defined by the current UniversalSortableDateTimePattern property. This pattern is a defined standard and the property is read-only. Therefore, it is always the same regardless of the culture used or the format provider supplied.<br>The custom format string is "yyyy'-'MM'-'dd HH':'mm':'ss'Z'".<br>No time zone conversion is done when the date and time is formatted. Therefore, the application must convert a local date and time to Coordinated Universal Time (UTC) before using this format specifier. |
| U | Universal sortable date/time pattern | Represents a custom **DateTime** format string defined by the current **FullDateTimePattern** property.<br>This pattern is the same as the full date/long time (F) pattern. However, formatting operates on the Coordinated Universal Time (UTC) that is equivalent to the **DateTime** object being formatted. |
| Y or y | Year month pattern | Represents a custom **DateTime** format string defined by the current YearMonthPattern property.<br>For example, the custom format string for the invariant culture is "yyyy MMMM". |
| Custom format | Custom patterns –<br>"MM'/'dd yyyy"<br>"dd.MM.yyyy"<br>"MM.dd.yyyy HH:mm"<br>"dddd, MMMM (yyyy): HH:mm:ss" | <br>03/17 2019<br>17.03.2019<br>03.17.2019 06:23<br>Tuesday, march (2019) : 06:23:00 |

| | | |
|---|---|---|
| Any other single character | (Unknown specifier) | An unknown specifier throws a runtime format exception. |

## Arrays

For creating, modifying, sorting and searching arrays.

| PROPERTY | DESCRIPTION | EXAMPLE |
|---|---|---|
| **IsFixedSize** | checks whether the Array has a fixed size. | string[] arrVal = new string[] {“stud1”, “stud2”, “stud3”};<br><br>arrVal.IsFixedSize; |
| **IsReadOnly** | Checks whether the Array is read-only. | arrVal.IsReadOnly; |
| **IsSynchronized** | Checks whether access to the Array is synchronized (thread safe). | arrVal.IsSynchronized; |
| **Length** | Gets the total number of elements in all the dimensions of the Array. | arrVal.Length; |
| **LongLength** | Length in 64-bit integer | arrVal.LongLength; |
| **Rank** | Gets the rank (number of dimensions) of the Array. For example, a one-dimensional array returns 1, a two-dimensional array returns 2, and so on. | arrVal.Rank; |
| **SyncRoot** | Gets an object used to synchronize Array access | arrVal.SyncRoot; |
| **AsReadOnly()** | Returns a read-only wrapper for the specified array. | Array.AsReadOnly(arrVal); |

| | | |
|---|---|---|
| **BinarySearch()** | Searches a value in a one-dimensional sorted array using a binary search algorithm. | Array.BinarySearch(arrVal, obj); where obj is the object to be searched. |
| **Clear()** | Sets a range of elements in an array to the default value of each element type. | Array.Clear(arrVal, 0, 2);<br>If arrVal is an array of integers, the elements at position 0 to 2 will be set to zero after doing Clear(). |
| **Clone()** | Create a shallow copy of the Array. | Array.Clone(arrVal); |
| **ConstrainedCopy()** | Copies a range of elements from an Array starting at the specified source index and pastes them to another Array starting at the specified destination index. Guarantees that all changes are undone if the copy does not succeed completely. | Array.ConstrainedCopy(srcArr, 0, destArr, 3, 5);<br>where srcArr is the source array,<br>0 is the start index from where copy should begin,<br>destArr is the destination array,<br>3 is the place where copy should start in the destination array,<br>5 is the number of elements to copy |
| **ConvertAll()** | Converts an array of one data type to an array of another data type. | conArr = Array.ConvertAll(arrVal, new Converter<dtype1, dtype2> (method)); |
| **Copy()** | Copies a range of elements in one Array to another Array and performs type casting and boxing as required. | Array.Copy(srcArr, destArr, 2);<br>copies first two elements from srcArr to destArr |
| **CopyTo()** | Copies all the elements of the current one-dimensional array to the specified one-dimensional array. | Array.CopyTo(destArr, 4);<br>copy starts from index 4 |
| **CreateInstance()** | Initializes a new instance of the Array class. | Array.CreateInstance(typeof(String), length); |
| **Empty()** | Returns an empty array. | arrVal.Empty() |

| | | |
|---|---|---|
| **Equals()** | Determines whether the specified object is equal to the current object. | arrVal.Equals(arrVal2); |
| **Exists()** | Determines whether the specified array contains elements that match the conditions defined by the specified predicate. | Array.Exists(srcArr, “<elementname>”); |
| **Find()** | Searches for an element that matches the conditions defined by the specified predicate, and returns the first occurrence within the entire Array. | Array.Find(arrVal, <matching pattern>); |
| **FindAll()** | Retrieves all the elements that match the conditions defined by the specified predicate. | Array.FindAll(arrVal, <matching pattern>); |
| **FindIndex()** | Searches for an element that matches the conditions defined by a specified predicate, and returns the zero-based index of the first occurrence within an Array or a portion of it. | Array.FindIndex(arrVal, <matching pattern>); |
| **FindLast()** | Searches for an element that matches the conditions defined by the specified predicate, and returns the last occurrence within the entire Array. | Array.FindLast(arrVal, <matching pattern>); |
| **FindLastIndex()** | Searches for an element that matches the conditions defined by a specified predicate, and returns the zero-based index of the last occurrence within an Array or a portion of it. | Array.FindLastIndex(arrVal, <matching pattern>); |
| **ForEach()** | Loops through each element of the array and performs the specified action | Array.ForEach(arrVal, Action) |

| | | |
|---|---|---|
| **GetEnumerator()** | Returns an IEnumerator for the Array. | arrVal.GetEnumerator() |
| **GetHashCode()** | default hash function. | arrVal.GetHashCode() |
| **GetLength()** | Gets a 32-bit integer that represents the number of elements in the specified dimension of the Array. | arrVal.GetLength(i) where i is an intege |
| **GetLongLength()** | Gets a 64-bit integer that represents the number of elements in the specified dimension of the Array. | arrVal.GetLongLength(i) where i is an integer |
| **GetLowerBound()** | Gets the index of the first element of the specified dimension in the array. | arrVal.GetLowerBound(i) where i is an integer |
| **GetType()** | Gets the Type of the current instance. | arrVal.GetType() |
| **GetUpperBound()** | Gets the index of the last element of the specified dimension in the array. | arrVal.GetUpperBound(i) where i is an integer |
| **GetValue()** | Gets the value of the specified element in the current Array. | |
| **IndexOf()** | Searches for the specified object and returns the index of its first occurrence in a one-dimensional array or in a range of elements in the array. | arrVal.IndexOf(object) |
| **Initialize()** | Initializes every element of the value-type Array by calling the default constructor of the value type. | |
| **LastIndexOf()** | Returns the index of the last occurrence of a value in a one-dimensional Array or in a portion of the Array. | arrVal.LastIndexOf(i) |

| | | |
|---|---|---|
| **MemberwiseClone()** | Creates a shallow copy of the current Object. | |
| **Resize()** | Changes the number of elements of a one-dimensional array to the specified new size. | Array.Resize(ref arrVal, len-2);<br>where len is the original length of the array |
| **Reverse()** | Reverses the order of the elements in a one-dimensional Array or in a portion of the Array. | arrVal.Reverse() |
| **SetValue()** | Sets the specified element in the current Array to the specified value. | Array.SetValue(arrVal[i]) |
| **Sort()** | Sorts the elements in a one-dimensional array. | Array.Sort(arrVal) |
| **ToString()** | Returns a string that represents the current object.<br>(Inherited from Object) | arrVal.ToString() |
| **TrueForAll()** | Determines whether every element in the array matches the conditions defined by the specified predicate. | Array.TrueForAll(arrVal, <matching pattern>) |

## Control Statements

| | |
|---|---|
| if-else | if (true) {...}<br>else if (true) {...}<br>else {...} |
| switch | switch (var)<br>{<br>case 1: break;<br>case 2: break;<br>default: break;<br>} |
| for | for (int i =0; i <=len; i++) {...} |

| | |
|---|---|
| foreach-in | foreach (int item in array) {...} |
| while | while (true) {...} |
| do... while | do {...}<br>while (true); |
| try-catch-finally | try {...}<br>catch (Exception e) {...}<br>catch {...}<br>finally {...} |

## Regular Expressions

| | |
|---|---|
| + | match one or more occurrence |
| * | match any occurrence (zero or more) |
| ? | match 0 or 1 occurrence |
| \d \D | match decimal digit or non-character |
| \w \W | match any word character |
| \s \S | match white space or no white space |
| [] | match any character inside the square brackets |
| [^] | match any character not present in the square brackets |
| a \| b | either a or b |
| \n | new line |
| \r | carriage return |
| \t | tab |

## Collections

### Arraylist

| | |
|---|---|
| Capacity | Gets or sets the number of elements that the ArrayList can contain. |
| Count | Gets the number of elements actually contained in the ArrayList. |
| IsFixedSize | Gets a value indicating whether the ArrayList has a fixed size. |
| IsReadOnly | Returns whether the ArrayList is read-only |
| Item | Gets or sets the element at the specified index. |
| Add(object value) | Adds an object to the end of the ArrayList |
| AddRange(ICollection c); | Adds the elements of an ICollection to the end of the ArrayList. |
| Clear(); | Removes all elements of an ArrayList. |
| Contains(object item); | Checks whether an element is in the ArrayList. |
| GetRange(int index, int count); | Returns an ArrayList which represents a subset of the elements in the source ArrayList. |
| IndexOf(object); | Returns the zero-based index of the first occurrence of a value in the ArrayList or in a portion of it. |
| Insert(int index, object value); | Inserts an element into the ArrayList at the specified index. |
| InsertRange(int index, ICollection c); | Inserts the elements of a collection into the ArrayList at the specified index. |
| Remove(object obj); | Removes the first occurrence of a specific object from the ArrayList. |
| RemoveAt(int index); | Removes the element at the specified index of the ArrayList. |
| RemoveRange(int index, int count); | Removes a range of elements from the ArrayList |
| Reverse(); | Reverses the order of the elements in the ArrayList. |
| SetRange(int index, ICollection c); | Copies the elements of a collection over a range of elements in the ArrayList. |

| | |
|---|---|
| Sort(); | Sorts the elements in the ArrayList. |
| TrimToSize(); | Sets the capacity to the actual number of elements in the ArrayList. |

## Hashtable

| | |
|---|---|
| Count | Gets the number of key-and-value pairs contained in the Hashtable. |
| IsFixedSize | Gets a value indicating whether the Hashtable has a fixed size |
| IsReadOnly | Gets a value indicating whether the Hashtable is read-only. |
| Item | Gets or sets the value associated with the specified key. |
| Keys | Gets an ICollection containing the keys in the Hashtable. |
| Values | Gets an ICollection containing the values in the Hashtable |
| Add(object key, object value); | Adds an element with the specified key and value into the Hashtable |
| Clear(); | Removes all elements from the Hashtable. |
| ContainsKey(object key); | Determines whether the Hashtable contains a specific key. |
| ContainsValue(object value); | Determines whether the Hashtable contains a specific value. |
| Remove(object key); | Removes the element with the specified key from the Hashtable. |

## SortedList

| | |
|---|---|
| Capacity | Gets or sets the capacity of the SortedList. |
| Count | Gets the number of elements in the SortedList. |
| IsFixedSize | Checks if the SortedList is of fixed size. |
| IsReadOnly | Checks if the SortedList is read-only. |

| | |
|---|---|
| Item | Gets and sets the value associated with a specific key in the SortedList. |
| Keys | Gets the keys in the SortedList. |
| Values | Gets the values in the SortedList. |
| Add(object key, object value) | Adds an element with the specified key and value into the SortedList |
| Clear() | Removes all elements from the SortedList. |
| ContainsKey(object key); | Checks if the SortedList contains a specific key. |
| ContainsValue(object value); | Checks if the SortedList contains a specific value. |
| GetByIndex(int index); | Gets the value at the specified index of the SortedList. |
| GetKey(int index); | Gets the key at the specified index of the SortedList. |
| GetKeyList(); | Returns list of keys in the SortedList |
| GetValueList(); | Returns list of values in the SortedList |
| IndexOfKey(object key); | Returns the zero-based index of the specified key in the SortedList. |
| IndexOfValue(object value); | Returns the zero-based index of the first occurrence of the specified value in the SortedList. |
| Remove(object key); | Removes the element with the specified key from the SortedList. |
| RemoveAt(int index); | Removes the element at the specified index of SortedList. |
| TrimToSize(); | Sets the capacity to the actual number of elements in the SortedList. |

## Stack

| | |
|---|---|
| Count | Number of elements in the Stack. |
| Clear(); | Removes all elements from the Stack. |

| | |
|---|---|
| Contains(object obj); | Checks if an element is in the Stack. |
| Peek(); | Returns the object at the top of the Stack without removing it. |
| Pop(); | Removes and returns the object at the top of the Stack. |
| Push(object obj); | Inserts an object at the top of the Stack. |
| ToArray(); | Copies the Stack to a new array. |

## Queue

| | |
|---|---|
| Count | number of elements in the Queue. |
| Clear(); | Removes all elements from the Queue. |
| Contains(object obj); | Checks if the specified object is present in the Queue. |
| Dequeue(); | Removes and returns the object at the beginning of the Queue. |
| Enqueue(object obj); | Adds an object to the end of the Queue. |
| ToArray(); | Copies the Queue to a new array. |
| TrimToSize(); | Sets the capacity to the actual number of elements in the Queue. |

## Dictionary

| | |
|---|---|
| Count | Gets the total number of elements exists in the Dictionary<TKey,TValue>. |
| IsReadOnly | Returns a boolean after checking if the Dictionary<TKey,TValue> is read-only. |
| Item | Gets or sets the element with the specified key in the Dictionary<TKey,TValue>. |
| Keys | Returns collection of keys of Dictionary<TKey,TValue>. |
| Values | Returns collection of values in Dictionary<TKey,TValue>. |
| Add | Add key-value pairs in Dictionary<TKey, TValue> collection. |

| | |
|---|---|
| Remove | Removes the first occurrence of specified item from the Dictionary<TKey, TValue>. |
| ContainsKey | Checks if the specified key exists in Dictionary<TKey, TValue>. |
| ContainsValue | Checks if the specified value exists in Dictionary<TKey, TValue>. |
| Clear | Removes all the elements from Dictionary<TKey, TValue>. |
| TryGetValue | Returns true and assigns the value with specified key, if key does not exists then return false. |

## Exception Handling

```
try{
} catch (Exception e){
throw;
}
```

### Methods

| | |
|---|---|
| No return type | public void MyMethod(){} |
| static method, no object needed to call method | public static void MyMethod(){} |
| with return type | public returnType MyMethod(){<br>return val;<br>} |
| passing parameters | public void MyMethod(String s, int i) {<br>} |

### Classes

```
Class MyClass
{
 /*Class definition*/
}
Object creation –
MyClass ClassObj = new MyClass();
```

### Partial Class

Classes within the same namespace can be split into smaller classes with same name.

```
// PartialClass1.cs
using System;
namespace PartialClasses
{
 public partial class PartialClass
 {
 public void HelloWorld()
 {
 Console.WriteLine("Hello, world!");
 }
 }
}
```

```
// PartialClass2.cs
using System;
namespace PartialClasses
{
 public partial class PartialClass
 {
 public void HelloUser()
 {
 Console.WriteLine("Hello, user!");
 }
 }
}
```

A single instance is enough to call the methods of these partial classes.

```
PartialClass pc = new PartialClass();
pc.HelloWorld();
pc.HelloUser();
```

## File Handling

| | | |
|---|---|---|
| File.Exists | Check the existence of the file in the specified path | File.Exists(path) |
| File.ReadAllLines | Read all the lines from the file specified by the path | File.ReadAllLines(path)<br>Console.WriteLines(File.ReadAllLines(path<br>// Write to console |
| File.ReadAllText | Read all the text from the file and store it as a single string | File.ReadAllText(path) |
| File.Copy | Copy content from one file to another | File.Copy(srcfilepath, destfilepath); |
| File.Delete | Delete an existing file from the specified path | File.Delete(path) |