C# Basic Documentation

Types

Type	Description
byte	8-bit unsigned integer
sbyte	8-bit signed integer
short	16-bit signed integer
ushort	16-bit unsigned integer
int	32-bit signed integer
uint	32-bit unsigned integer
long	64-bit signed integer
ulong	64-bit unsigned integer
float	32-bit Single-precision floating point type
double	64-bit double-precision floating point type
decimal	128-bit decimal type for financial and monetary calculations
char	16-bit single Unicode character
bool	8-bit logical true/false value
object	Base type of all other types.
string	A sequence of Unicode characters

Console Reading/Writing

String Console.ReadLine()

Writes the specified string value, followed by the current line terminator, to the standard output stream.

Returns

String

The next line of characters from the input stream, or null if no more lines are available.

Console.WriteLine(String text)

Writes the specified string value, followed by the current line terminator, to the standard output stream.

Console.WriteLine(String text, Object[] objects)

Writes the text representation of the specified array of objects, followed by the current line terminator, to the standard output stream using the specified format information.

```
static void Main(string[] args)
{
    string world = "World";
    Console.WriteLine("Hello World!");
    Console.WriteLine("Hello " + world + "!");
    Console.WriteLine("Hello {0}!", world);
    Console.WriteLine($"Hello {world}!");
    string sentence = Console.ReadLine();
}
```

Random Class

Represents a pseudo-random number generator. The Random class methods can be used to obtain random numbers.

int Next()

Returns a non-negative random integer.

Returns

Integer

int Next(int maxValue)

Returns a non-negative random integer that is less than the specified maximum.

Returns

Integer

int Next(int minValue, int maxValue)

Returns a random integer that is between the specified range, excluding the maximum number.

Returns

Integer

```
static void Main(string[] args)
{
   Random randomObject = new Random();      //create the random class object
   int number1 = randomObject.Next();       //non-negative number
   int number2 = randomObject.Next(10);     //number from 0 to 9
   int number3 = randomObject.Next(1, 101);//number from 1 to 100
}
```

Conversion from String

Every built-in type has a parsing method that converts a string to the corresponding type.

T T.Parse(String text)

Parses the provided string to the indicated type (T).

Returns

T

Returns the parsed string.

```
static void Main(string[] args)
{
   string text = "0";
    int number = int.Parse(text);
    float numberFloat = float.Parse(text);

    string text2 = "true";
    bool booleanValue = bool.Parse(text2);
}
```

Selection Structures

The **if** statement

Selects a statement to execute based on the value of a Boolean expression.

An if statement without an else part executes its body only if a Boolean expression evaluates to true. An if statement with else/else if parts selects one of the statements to execute based on the Boolean expression. Several if statements can be nested to check multiple conditions.

```
int value = 0;
if (value > 50)
{
    Console.WriteLine("Larger than 50");
}
else if (value > 10)
{
    Console.WriteLine("Larger than 10");
}
else
{
    Console.WriteLine("Something else...");
}
```

The **switch** statement

Selects a statement list to execute based on a pattern match with an expression.

The switch statement selects a statement list to execute based on a pattern match with a match expression. One or more constant cases can be defined to which the expression is compared.

A default case can be defined, which specifies statements to execute when a match expression doesn't match any other case pattern. If a match expression doesn't match any case pattern and there is no default case, control falls through a switch statement.

The statement lists associated to each case must end with a break statement.

```
int number = 2;
switch (number)
{
    case 1:
        Console.WriteLine("Value is one.");
        break;

    case 2:
        Console.WriteLine("Value is two.");
        break;

    default:
        Console.WriteLine($"Value is {number}.");
        break;
}
```

Iteration (repetition) Structures

The **while** statement

The while statement executes a statement or a block of statements while a specified Boolean expression evaluates to true. Because that expression is evaluated before each execution of the loop, a while loop executes zero or more times.

```
int n = 0;
while (n < 5)
{
    Console.Write(n);
    n++;
}
```

The **do** statement

The do statement executes a statement or a block of statements while a specified Boolean expression evaluates to true. Because that expression is evaluated after each execution of the loop, a do loop executes one or more times.

```
int n = 0;
do
{
    Console.Write(n);
    n++;
} while (n < 5);
```

The **for** statement

The for statement executes a statement or a block of statements while a specified Boolean expression evaluates to true.

It is composed of three statements:

- the first statement is executed (one time) before the execution of the code block.
- the second statement defines the condition for executing the code block.
- The third statement is executed (every time) after the code block has been executed.

The following example shows the for statement that executes its body while an integer counter is less than three:

```
for (int i = 0; i < 3; i++)
{
    Console.Write(i);
}
```

The **foreach** statement

The foreach statement executes a statement or a block of statements for each element within a specific collection.

```
int[] array3 = { 1, 2, 3, 4, 5, 6 };

foreach(int number in array3)
{
   Console.WriteLine(number);
}
```

Jump Structures

The **break** statement

The break statement terminates the closest enclosing loop or switch statement in which it appears. Control is passed to the statement that follows the terminated statement, if any.

```
for (int i = 0; i < 3; i++)
{
    if(i == 2)
       break;
    Console.Write(i);
}
```

The **continue** statement

The continue statement passes control to the next iteration of the enclosing iteration statement in which it appears. If there is no following iteration, control is passed to the statement that follows iteration statement, if any.

```
for (int i = 0; i < 3; i++)
{
    if(i == 2)
       break;
    Console.Write(i);
}
```

The **return** statement

The return statement terminates execution of the method in which it appears and returns control to the calling method. It can also return an optional value. If the method is a void type, the return statement can be omitted.

```
static double CalculateArea(int r)
{
    double area = r * r * Math.PI;
    return area;
}

static void Main()
{
    int radius = 5;
    double result = CalculateArea(radius);
    Console.WriteLine("The area is {0:0.00}", result);
}
```

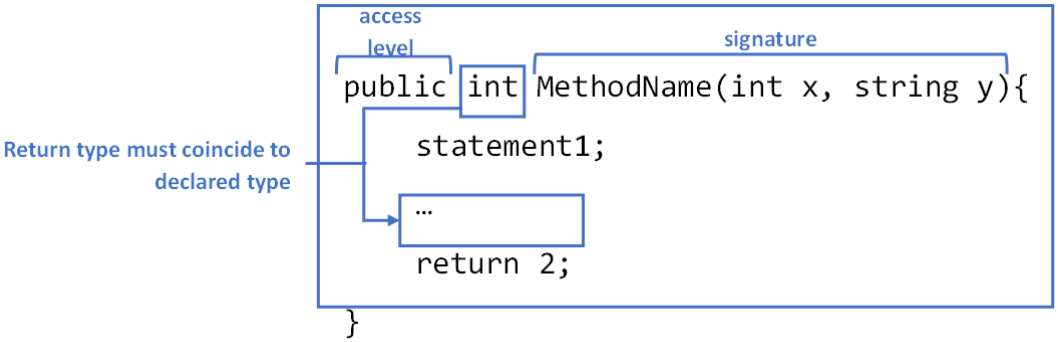
Methods

Methods are code blocks that correspond to a group of statements. The contained statements are executed when the method is called.

One or more parameters may be defined in a method's declaration. Parameters are the list of variables provided externally to a method. When calling a method, the values that are passed to those variables are called arguments.

Methods have a signature, which is their unique identification in the program. In C#, it is composed by the method's name and the parameter types

A method's declaration includes characteristics that are not part of the signature: access level and return type.



Strings

A string is an object of type String whose value is text. Internally, the text is stored as a sequential read-only collection of Char objects.

In C#, the string keyword is an alias for String. Therefore, String and string are equivalent.

Strings can be initialized with null value, string literals or from an array of chars. They are immutable: when a string variable is modified, a new object is created. The Length property of a string represents the number of Char objects it contains.

```
static void Main(string[] args)
{
    string a = "Hello", b = "World";
     string c = "Hello World";
     string d = "He";
     d += "llo";
     string e = a + " " + b;
```

```
    Console.WriteLine(a == d);
    Console.WriteLine(object.ReferenceEquals(a, d));

    char letter = 'H';
    char letter2 = a[0];
    Console.WriteLine(letter == letter2);
}
```

Methods	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(s) is/are the last character(s) 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)
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”); str1.Insert(i, “Thank You”);
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"; String sep = {"and"}; strArray = str1.Split(sep, StringSplitOptions.None);
StartsWith()	Checks whether the first character(s) of string is/are same as specified character(s).	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();

Arrays

An array is a data structure that contains a number of variables which are accessed through computed indices. The variables contained in an array, also called the elements of the array, are all of the same type, and this type is called the element type of the array.

The length of an array cannot be changed after declaration, so arrays need to be initialized when declared.

Similar to the string keyword, array is an alias for the Array class. Values in an array can be accessed and changed by using the indexer operator.

```
static void Main(string[] args)
{
  // Declare a single-dimensional array of 5 integers.
  int[] array1 = new int[5];

  // Declare and set array element values.
  int[] array2 = new int[] { 1, 3, 5, 7, 9 };
```

```
  // Alternative syntax.
  int[] array3 = { 1, 2, 3, 4, 5, 6 };
  array3[0] = 0;

  foreach (int number in array3)
  {
    Console.WriteLine(number);
  }
}
```

Methods	Definitions	Example
Clear()	Clears the contents of an array.	int [] numbers = new int{1,2,3}; numbers.Clear();
IndexOf()	Searches for the specified object and returns the index of its first occurrence in a one-dimensional array.	int [] numbers = new int{1,2,3}; int index=numbers.IndexOf(2);
LastIndexOf()	Searches for the specified object and returns the index of the last occurrence within the entire one-dimensional Array.	int [] numbers = new int{1,2,3}; int index = numbers.LastIndexOf(2);
Reverse()	Reverses the sequence of the elements in the entire one-dimensional Array.	int [] numbers = new int{1,2,3}; numbers.Reverse();
SetValue()	Sets a value to the element at the specified position in the one-dimensional Array. The index is specified as a 32-bit integer.	int [] numbers = new int{1,2,3}; numbers.SetValue(0, 10);
Sort()	Sorts the elements in an entire one-dimensional Array using the IComparable implementation of each element of the Array.	int [] numbers = new int{1,2,3}; numbers.Sort();