C# Documentation

Variables:

- Int ->(Stores an integer value)
- float -> (Stores decimal point values)
- double -> (similar to float but it has more precision)
- long -> (Used for storing large integers)
- char -> (Used to store single characters)
- bool -> (Stores Boolean value)
- string -> (Stores a sequence of characters)

Examples:

```
int a = 5;
float b = 6.98;
double c = 8.976543;
long d =6748382119427;
char e = ‘A’ (Single inverted commas are used)
bool flag =true or flag=false;
string s = “CSI”;
```

Simple Hello world program:

```
using System;

namespace HelloWorldApplication {
  class HelloWorld {
    static void Main(string[] args) {
      /* my first program in C# */
      Console.WriteLine("Hello World");
      Console.ReadKey();
    }
  }
}
```

WriteLine is a method of the Console class defined in the System namespace. This statement causes the message "Hello, World!" to be displayed on the screen.

The last line Console.ReadKey(); is for the VS.NET Users. This makes the program wait for a key press and it prevents the screen from running and closing quickly when the program is launched from Visual Studio .NET.

BASIC PRINTING STATEMENTS AND SCANNING STATEMENTS

```
Console.WriteLine("n = {0}", n);
```

(This statement writes the value stored in variable n)

Console.WriteLine("a = {0}, b={1}, c={2}", a,b,c);
(This statement writes the value stored in variables a,b,c in the same order)

Console.ReadKey();
(This statement reads a key pressed by the user)

Loops in C#:

While loop:
Example code:

```
using System;

namespace Loops {
  class Program {
    static void Main(string[] args) {
      /* local variable definition */
      int a = 10;

      /* while loop execution */
      while (a < 20) {
        Console.WriteLine("value of a: {0}", a);
        a++;
      }
      Console.ReadLine();
    }
  }
}
```

Output:

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

For loop:

```
using System;
```

```
namespace Loops {
  class Program {
    static void Main(string[] args) {

      /* for loop execution */
      for (int a = 10; a < 20; a = a + 1) {
        Console.WriteLine("value of a: {0}", a);
      }
      Console.ReadLine();
    }
  }
}
```

Output:

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

Do… While Loop:

```
using System;

namespace Loops {
  class Program {
    static void Main(string[] args) {
      /* local variable definition */
      int a = 10;

      /* do loop execution */
      do {
        Console.WriteLine("value of a: {0}", a);
        a = a + 1;
      }
      while (a < 20);
      Console.ReadLine();
    }
  }
}
```

Output:
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19

Nested loops:
Syntax for Nested **FOR LOOP** in C#:

```
for ( init; condition; increment ) {
   for ( init; condition; increment ) {
      statement(s);
   }
   statement(s);
}
```

Syntax for nested **WHILE LOOP** in C#

```
while(condition) {
   while(condition) {
      statement(s);
   }
   statement(s);
}
```

Syntax for nested **do ... while loop:**

```
do {
   statement(s);
   do {
      statement(s);
   }
   while( condition );
}
while( condition );
```

Example using nested for loop:

```
using System;

namespace Loops {
   class Program {
```

```
    static void Main(string[] args) {
      /* local variable definition */
      int i, j;

      for (i = 2; i < 100; i++) {
        for (j = 2; j <= (i / j); j++)
        if ((i % j) == 0) break; // if factor found, not prime
        if (j > (i / j)) Console.WriteLine("{0} is prime", i);
      }
      Console.ReadLine();
    }
  }
}
```

Output:

```
2 is prime
3 is prime
5 is prime
7 is prime
11 is prime
13 is prime
17 is prime
19 is prime
23 is prime
29 is prime
31 is prime
37 is prime
41 is prime
43 is prime
47 is prime
53 is prime
59 is prime
61 is prime
67 is prime
71 is prime
73 is prime
79 is prime
83 is prime
89 is prime
97 is prime
```

Arrays in C#:

To declare an array in C#, you can use the following syntax −

datatype[] arrayName;

where,

- datatype is used to specify the type of elements in the array.
- [ ] specifies the rank of the array. The rank specifies the size of the array.
- arrayName specifies the name of the array.

For example,

double[] balance;

Initializing an Array:
double[] balance = new double[10];

You can assign values to individual array elements, by using the index number, like −
double[] balance = new double[10];
balance[0] = 4500.0;

You can assign values to the array at the time of declaration, as shown −
double[] balance = { 2340.0, 4523.69, 3421.0};

You can also create and initialize an array, as shown −
int [] marks = new int[5] { 99, 98, 92, 97, 95};

You may also omit the size of the array, as shown −
int [] marks = new int[] { 99, 98, 92, 97, 95};

You can copy an array variable into another target array variable. In such case, both the target and source point to the same memory location −
int [] marks = new int[] { 99, 98, 92, 97, 95};
int[] score = marks;

When you create an array, C# compiler implicitly initializes each array element to a default value depending on the array type. For example, for an int array all elements are initialized to 0.

Accessing Array Elements:

double salary = balance[9];

Example for Implementation of array:
using System;

```
namespace ArrayApplication {
  class MyArray {
     static void Main(string[] args) {
```

```
      int []  n = new int[10]; /* n is an array of 10 integers */
      int i,j;

      /* initialize elements of array n */
      for ( i = 0; i < 10; i++ ) {
         n[ i ] = i + 100;
      }

      /* output each array element's value */
      for (j = 0; j < 10; j++ ) {
         Console.WriteLine("Element[{0}] = {1}", j, n[j]);
      }
      Console.ReadKey();
    }
  }
}
```

Output:

```
Element[0] = 100
Element[1] = 101
Element[2] = 102
Element[3] = 103
Element[4] = 104
Element[5] = 105
Element[6] = 106
Element[7] = 107
Element[8] = 108
Element[9] = 109
```

Functions in C# are called Methods:
When you define a method, you basically declare the elements of its structure. The syntax for defining a method in C# is as follows −

```
<Access Specifier> <Return Type> <Method Name>(Parameter List) {
   Method Body
}
```

Following are the various elements of a method −

- Access Specifier − This determines the visibility of a variable or a method from another class.

- Return type − A method may return a value. The return type is the data type of the value the method returns. If the method is not returning any values, then the return type is void.
- Method name − Method name is a unique identifier and it is case sensitive. It cannot be same as any other identifier declared in the class.
- Parameter list − Enclosed between parentheses, the parameters are used to pass and receive data from a method. The parameter list refers to the type, order, and number of the parameters of a method. Parameters are optional; that is, a method may contain no parameters.
- Method body − This contains the set of instructions needed to complete the required activity.

Example:

```
class NumberManipulator {

   public int FindMax(int num1, int num2) {
      /* local variable declaration */
      int result;

      if (num1 > num2)
         result = num1;
      else
         result = num2;

      return result;
   }
   ...
}
```

Calling methods:

```
using System;

namespace CalculatorApplication {
   class NumberManipulator {
      public int FindMax(int num1, int num2) {
         /* local variable declaration */
         int result;

         if (num1 > num2)
            result = num1;
         else
            result = num2;
         return result;
```

```
    }

    static void Main(string[] args) {
      /* local variable definition */
      int a = 100;
      int b = 200;
      int ret;
      NumberManipulator n = new NumberManipulator();

      //calling the FindMax method
      ret = n.FindMax(a, b);
      Console.WriteLine("Max value is : {0}", ret );
      Console.ReadLine();
    }
  }
}
```

Output:
Max value is : 200