Cheatography

C# Cheat Sheet

by Sara (lasago) via cheatography.com/70899/cs/20398/

Create, run, debug programs from scratch

View>Terminal	
dotnet new console	Creates new Hello World program from scratch
dotnet restore	resolve .NET build assets
dotnet run	Run the program
Debugging	
Debug> .NET Core Launch (Console)	
Click line, add red dot (breakpoint), F5(run), local variables on left	
Compiling	
	csc hello.cs

New Class

File MyClass.cs	

Inside:	using System;
	namespace SameAsMains{
	classMyClass{
	//...}

Reference MyClass in another class	MyClass c1 = new MyClass();
	c1.function();

C#

C# is an object-oriented language, but C# further includes support for component-oriented programming.

All C# types, including primitive types such as int and double, inherit from a single root object type.

C# supports both user-defined reference types and value types

The "Hello, World" program starts with a using directive that references the System namespace.

Namespaces provide a hierarchical means of organizing C# programs and libraries. Namespaces contain types and other namespaces—for example, the System namespace contains a number of types, such as the Console class referenced in the program, and a number of other namespaces, such as IO and Collections.

A using directive that references a given namespace enables unqualified use of the types that are members of that namespace. Because of the using directive, the program can use Console.WriteLine as shorthand for System.Console.WriteLine.

While instance methods can reference a particular enclosing object instance using the keyword this, static methods operate without reference to a particular object.

short x = 2, y = 3 will make x a short and y an int

Types

string

Length (str.Length)

Enum	The enum keyword is used to declare an enumeration, a distinct type that consists of a set of named constants called the enumerator list.

Types (cont)	
	By default, the first enumerator has the value 0, and the value of each successive enumerator is increased by 1. For example, in the following enumeration, Sat is 0, Sun is 1, Mon is 2, and so forth.
	enum Day {Sat, Sun, Mon, Tue, Wed, Thu, Fri};
	Enumerators can use initializers to override the default values, as shown in the following example.
	enum Day {Sat=1, Sun, Mon, Tue, Wed, Thu, Fri};
	Every enumeration type has an underlying type, which can be any integral numeric type. The char type cannot be an underlying type of an enum. The default underlying type of enumeration elements is int. To declare an enum of another integral type, such as byte, use a colon after the identifier followed by the type, as shown in the following example.
	enum Day : byte {Sat=1, Sun, Mon, Tue, Wed, Thu, Fri};
	an explicit cast is necessary to convert from enum type to an integral type.
	int x = (int)Day.Sun;
Enteros con signo	sbyte, short, int, long
Enteros sin signo	byte, ushort, ulong, uint
FLoating point	float, double
float and double	binary point, not 100% accurate
decimal	decimal, as accurate as you can get
Value types	
Simple	todos los anteriores y bool
Enumer-ación	enum e {}
Estructura	struct S {}
	https://samrueby.com/2016/09/05/when-should-you-use-a-struct-instead-of-a-class/
Reference types	
Class types	object, string, class C{}
Interface types	interface I {}
Tipos de matroz	int[]¸int[,]
Delegate types	delegate int D{}

Input and Output	
Print line	System.Console.WriteLine()
	System.Console.Write()
Format string ($)	Console.WriteLine($"Hello World! {c1.returnMessage()}");
Read line (string)	Console.ReadLine();
Read char	Console.ReadKey().KeyChar;
Read Int	int n = int.Parse(Console.ReadLine());
Read float	

https://docs.microsoft.com/es-es/dotnet/csharp/language-reference/tokens/interpolated

Data Structures and Collections	
Array	
create and initialize	String[] Words = new String[10];
create, initialize and fill	int[] prueba = new int[20]{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20};
int[] vector = {1,2,4}	
access values	
save values	
array (multidimensional)	
create and initialize	
create, initialize and fill	
access values	
save values	
get size	.Length
List	
create and initialize	var names = new List<string> { "<name>", "Ana", "Felipe" };
add to the end of the list	names.Add("Maria");
remove from the end	names.Remove("Ana");
access	Console.WriteLine($"My name is {names[0]}.");
length	Console.WriteLine($"The list has {names.Count} people in it");
indexOf	var index = names.IndexOf("Felipe");
Sort	names.Sort();
Add new values from array to list	n3.AddRange(n1);
Dictionary<TKey,TValue>	Tkey : The type of the keys in the dictionary. The type of the values in the dictionary.
Declare Dictionary with String key and value	Dictionary<string, string> openWith = new Dictionary<string, string>();
Declare and initialize DIctionary	var myDict = new Dictionary<string, string> { { "key1", "value1" }, { "key2", "value2" } };

Data Structures and Collections (cont)	
Add values to Dictionary	openWith.Add("txt", "notepad.exe");
	// The Add method throws an exception if the new key is // already in the dictionary.
Retrieve values from Dictionary	String program = openWith["txt"];
Check if value in Dictionary exists	if (openWith.TryGetValue("tif", out value)) { Console.WriteLine("For key = \"tif\", value = {0}.", value); } else { Console.WriteLine("Key = \"tif\" is not found."); }
List	
Declare and initialize Object List	List<Pet> petsList = new List<Pet>{ new Pet { Name="Barley", Age=8.3 }, new Pet { Name="Boots", Age=4.9 }, new Pet { Name="Whiskers", Age=1.5 }, new Pet { Name="Daisy", Age=4.3 } };

https://docs.microsoft.com/en-us/dotnet/standard/collections/

Scope	
this, ref, out, in, return, attributes, propertoes	
Keywords	
ref	if the caller passes a local variable expression or an array element access expression, and the called method replaces the object to which the ref parameter refers, then the caller's local variable or the array element now refers to the new object when the method returns
	When used in a method's parameter list, the ref keyword indicates that an argument is passed by reference, not by value. any operation on the parameter is made on the argument
	void Method(ref int refArgument) { refArgument = refArgument + 44; } int number = 1; Method(ref number); Console.WriteLine(number); // Output: 45
out	arguments are parsed by reference. any operation on the parameter is made on the argument. It is like the ref keyword, except that ref requires that the variable be initialized before it is passed. It is also like the in keyword, except that in does not allow the called method to modify the argument value.
	It is like the ref keyword, except that ref requires that the variable be initialized before it is passed. It is also like the in keyword, except that in does not allow the called method to modify the argument value

Scope (cont)

in	The in keyword causes arguments to be passed by reference. It makes the formal parameter an alias for the argument, which must be a variable. In other words, any operation on the parameter is made on the argument. It is like the ref or out keywords, except that in arguments cannot be modified by the called method. Whereas ref arguments may be modified, out arguments must be modified by the called method, and those modifications are observable in the calling context.
	the in modifier is usually unnecessary at the call site. It is only required in the method declaration.

A parameter is a variable in a method definition.

When a method is called, the arguments are the data you pass into the method's parameters.

Parameter is variable in the declaration of function.

Argument is the actual value of this variable that gets passed to function.

Hierarchy

Concepts

assembly	An assembly is a file that is automatically generated by the compiler upon successful compilation of every .NET application. It can be either a Dynamic Link Library or an executable file. It is generated only once for an application and upon each subsequent compilation the assembly gets updated.
"Flujo base e intermedio"	
Library	ensemble of functions
Struct and Class	Classes and structs are two of the basic constructs of the common type system in the .NET Framework. Each is essentially a data structure that encapsulates a set of data and behaviors that belong together as a logical unit. The data and behaviors are the members of the class or struct, and they include its methods, properties, and events, and so on.
	A class is a reference type. When an object of the class is created, the variable to which the object is assigned holds only a reference to that memory. When the object reference is assigned to a new variable, the new variable refers to the original object. Changes made through one variable are reflected in the other variable because they both refer to the same data. A struct is a value type. When a struct is created, the variable to which the struct is assigned holds the struct's actual data. When the struct is assigned to a new variable, it is copied. The new variable and the original variable therefore contain two separate copies of the same data. Changes made to one copy do not affect the other copy.

Concepts (cont)

	In general, classes are used to model more complex behavior, or data that is intended to be modified after a class object is created. Structs are best suited for small data structures that contain primarily data that is not intended to be modified after the struct is created.
Delegates	A delegate is a type that represents references to methods with a particular parameter list and return type. When you instantiate a delegate, you can associate its instance with any method with a compatible signature and return type. You can invoke (or call) the method through the delegate instance. Delegates are used to pass methods as arguments to other methods. Event handlers are nothing more than methods that are invoked through delegates. You create a custom method, and a class such as a windows control can call your method when a certain event occurs. The following example shows a delegate declaration: public delegate int PerformCalculation(int x, int y);
return	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. If the return statement is inside a try block, the finally block, if one exists, will be executed before control returns to the calling method.

Type Methods

IndexOf	Reports the zero-based index of the first occurrence of a specified Unicode character or string within this instance. The method returns -1 if the character or string is not found in this instance.	String, Array
string	nameOfString.IndexOf("r", 0, nameOfString.Length);	
array	int posI = Array.IndexOf(nameOfArray, "i");	
Equals	public virtual bool Equals (object obj); true if the specified object is equal to the current object; otherwise, false.	

Type Methods (cont)		
Split	https://docs.microsoft.com/en-us/dotnet/api/system.string.split?view=netframework-4.8	String
.ToLower()	https://docs.microsoft.com/en-us/dotnet/api/system.string.tolower?view=netframework-4.8	String
.Format	https://docs.microsoft.com/en-us/dotnet/api/system.string.format?view=netframework-4.8#code-try-18	String
	Console.WriteLine("Format Decimal: {0:n2}", num);	cambia el punto flotante (n2 = dos)
Linq		
.GroupBy()		
words.Except(banned)	https://docs.microsoft.com/en-us/dotnet/api/system.linq.enumerable.except?view=netframework-4.8	String

File management	
Write to a Text File	
Write list of strings to a file	string[] lines = { "First line", "Second line", "Third line" };
	System.IO.File.WriteAllLines(@"C:\Users\Public\TestFolder\WriteLines.txt", lines);
	// WriteAllLines creates a file, writes a collection of strings to the file, // and then closes the file. You do NOT need to call Flush() or Close().
Write String to a text file	string text = "A class is the most powerful data type in C#. Like a structure, " + "a class defines the data and behavior of the data type. ";
	// WriteAllText creates a file, writes the specified string to the file, // and then closes the file. You do NOT need to call Flush() or Close(). System.IO.File.WriteAllText(@"C:\Users\Public\TestFolder\WriteText.txt", text);
Write only some strings in an array to a file.	

File management (cont)

// The using statement automatically flushes AND CLOSES the stream and calls // IDisposable.Dispose on the stream object. // NOTE: do not use FileStream for text files because it writes bytes, but StreamWriter // encodes the output as text.	using (System.IO.StreamWriter file = new System.IO.StreamWriter(@"C:-\Users\Public\TestFolder\WriteLines2.txt")) { foreach (string line in lines) { // If the line doesn't contain the word 'Second', write the line to the file. if (!line.Contains("Second")) { file.WriteLine(line); } } }
Append new text to an existing file.	// The using statement automatically flushes AND CLOSES the stream and calls // IDisposable.Dispose on the stream object. using (System.IO.StreamWriter file = new System.IO.StreamWriter(@"C:\Users\Public\TestFolder\WriteLines2.txt", true)) { file.WriteLine("Fourth line"); } }

Read Text from a File

Read the file as one string	string text = System.IO.File.ReadAllText(@"C:\Users\Public\TestFolder\WriteText.txt");
	System.Console.WriteLine("Contents of WriteText.txt = {0}", text);
Read each line of the file into a string array. Each element of the array is one line of the file.	string[] lines = System.IO.File.ReadAllLines(@"C:\Users\Public\TestFolder\WriteLines2.txt");
	System.Console.WriteLine("Contents of WriteLines2.txt = "); foreach (string line in lines){ Console.WriteLine("\t" + line); }
	Console.WriteLine("\t" + line);

Variable declaration

const	You use the const keyword to declare a constant field or a constant local. Constant fields and locals aren't variables and may not be modified. Constants can be numbers, Boolean values, strings, or a null reference. Don’t create a constant to represent information that you expect to change at any time.

Variable declaration (cont)

casting	double r = 2.0; int n = 1; n = (int) r
explicit conversion	String cadena =”5”; float variable_flotante = float.Parse(cadena); int numero_entero = int.Parse(cadena); char var_car = char.Parse(cadena);
&&	returns the boolean value TRUE if both operands are TRUE and returns FALSE otherwise. The operands are implicitly converted to type bool prior to evaluation, and the result is of type bool.

Repetition structures

foreach	foreach(int number in numbers){ Console.Write($"\t{number}")}

Inheritance

C# and .NET support single inheritance only. That is, a class can only inherit from a single class.

Not all members of a base class are inherited by derived classes.	Static constructors, which initialize the static data of a class. Instance constructors, which you call to create a new instance of the class. Each class must define its own constructors. Finalizers, which are called by the runtime's garbage collector to destroy instances of a class.
public class C : A	//C inherits from A

Operators

int a = 10, b = 3;
d = a+b++;
e = ++a-b;
	a = 10, b = 3, d = null
	a = 10, b= 3, d = 13
	a = 11, d = 4, e = 7

!	calcula la negación lógica de su operando. Es decir, genera true, si el operando se evalúa como false, y false, si el operando se evalúa como true:
&	El operador & calcula el operador AND lógico de sus operandos. El resultado de x & y es true si x y y se evalúan como true. De lo contrario, el resultado es false.

Operators (cont)

^	The operator computes the logical exclusive OR, also known as the logical XOR, of its operands. The result of x y is true if x evaluates to true and y evaluates to false, or x evaluates to false and y evaluates to true. Otherwise, the result is false. That is, for the bool operands, the ^ operator computes the same result as the inequality operator !=.
|	The | operator computes the logical OR of its operands. The result of x | y is true if either x or y evaluates to true. Otherwise, the result is false. The | operator evaluates both operands even if the left-hand operand evaluates to true, so that the result must be true regardless of the value of the right-hand operand.
&&	computes the logical AND of its operands. The result of x && y is true if both x and y evaluate to true. Otherwise, the result is false. If x evaluates to false, y is not evaluated.

https://docs.microsoft.com/es-es/dotnet/csharp/language-reference/operators/boolean-logical-operators#logical-negation-operator-
https://docs.microsoft.com/es-es/dotnet/csharp/language-reference/operators/arithmetic-operators

Methods

Function	returns a value
Procedure	executes commands
Method	subroutine associated with an object in OO
Subroutine	returns multiple values

Documentation

Comments
//
/ /
/ <*summary*> </*summary*> /

Selection statements

case	int caseSwitch = 1; switch (caseSwitch) { case 1: Console.WriteLine("Case 1"); break; case 2: Console.WriteLine("Case 2"); break; default: Console.WriteLine("Default case"); break; }

Classes

Random	
var rand = new Random();	// Instantiate random number generator using system-supplied value as seed.
// Generate and display 5 random byte (integer) values.	var bytes = new byte[5]; rand.NextBytes(bytes);
// Generate and display 5 random integers between 0 and 100.	for (int ctr = 0; ctr <= 4; ctr++) Console.Write("{0,8:N0}", rand.Next(101)); Console.WriteLine();
// Generate and display 5 random integers.	Console.WriteLine("Five random integer values:"); for (int ctr = 0; ctr <= 4; ctr++) Console.Write("{0,15:N0}", rand.Next()); Console.WriteLine();
Obtener doubles según rango	rand.NextDouble() * 5
https://docs.microsoft.com/en-us/dotnet/api/system.random?view=netframework-4.8	
-	

1,5 will generate 4 random numbers

Compiled and Structured langauges

Tanto compiladores como interpretadores son programas que convierten el código que escribes a lenguaje de máquina.

Lenguaje de máquina son las instrucciones que entiende el computador (el procesador para ser más exactos) en código binario (unos y ceros).

La principal diferencia entre un lenguaje compilado y uno interpretado es que el lenguaje compilado requiere un paso adicional antes de ser ejecutado, la compilación, que convierte el código que escribes a lenguaje de máquina. Un lenguaje interpretado, por otro lado, es convertido a lenguaje de máquina a medida que es ejecutado.

Loops

```
foreach(var c in J)
for(int i =0; i < s.Length; i++)
while(c >10)
```