# C# Cheat sheet for Beginners

**Program Structure**

using System;  
  
namespace HelloWorld  
{  
 class Program  
 {  
 static void Main(string[] args)  
 {  
 Console.WriteLine("Hello World!");   
 }  
 }  
}

**Console Output**

To print/show a text in console, use the follow statement.

Console.WriteLine("Hello World!");

“Hello World” is a text that will be printed in user console.

**Console Input**

To read a input from the user, use the follow statement.

String x = Conso­le.R­ea­dLi­ne();

The code above read input from the user and save it in variable **x**.

**Variable and Data Types**

Variable are space in memory that store some information.

To create a variable, you should use the follow code

*DataType variableName = value;*

**Declaration examples**

// create a variable with name x to save a integer number  
int x;// create a variable with name age to save a integer number  
// and give it value 18  
int age = 18;// create a variable with name age to save a string (text data type)  
string name = "John";

**Assign value**

We can assign or change the value of the variable any time we want, after it been created.

x = 1000;age = 25; // change the value of variable to 25name = “John Snow”;

**Data Types**

C# support many data types, but initially you can focus in those.

**int** — store integer number

int weight = 56;

**string** — store text, text should be always inside “ ” (quotes) .

string address = “Cape Vert, Assomada”

**double**— store a decimal number (to separate integer from decimal places we use . )

double note = 9.5;

**bool** — store a boolean value, just assume true or false

bool approved = true;

**char** — store a single character, should be always inside ‘ ‘

char letter = ‘A’;

**Casting / Type Conversion**

Casting or type conversion is the process to convert one data type to another.

To convert a string to a integer number, you can do one of the follow codes.

// example 1  
int num = int.Parse("3");// example 2  
int num2 = Convert.ToInt32("5");

To convert a decimal number to a integer number, you can just do as code below.

// example 1  
double x = 9.5;  
int num = (int) x;// example 2  
double x2 = 9.5;  
int num2 = Convert.ToInt32(x2);

**Operators**

**Arithmetic**

This type of operators are used to perform mathematical operation.

**Addiction** (+), the + operator is used to sum two values.

int a = 7;  
int b = 3;  
int c = a + b;

**Subtraction** (-), the - operator is used to subtract one value to another.

int d = a - b;

**Multiplication** (\*), the \* operator is used to multiple two values.

int e = a \* b;

**Division** (/), the / operator is used to divide one value to another.

int f = a / b;

**Modulus** (%), the % operator is used to get division remainder.

int g = a % b;

**Increment**(++), the ++ operator is used to increase value of a variable by 1.

int h= 2;  
h++;  
// the h now will take the value 3

**Decrement** (--), the -- operator is used to decrease value of a variable by 1.

int i = 10;  
i--;  
// the i now will have the value 9

**Attribution**

This type of operators are used to assign value to variables.

**Assignment** (=), the operator = is used to assign value to a variable.

int a = 15;

**Addition assignment**(+=), the operator += is used to add value to a variable.

int b = 8;  
b += 8;  
// the code above is the same thing as  
// b = b + 8;

**Subtraction assignment**(-=), the operator -= is used to subtract a value in a variable.

**Multiplication assignment**(\*=), the operator \*= is used to multiple a value to a variable.

**Division assignment**(/=), the operator /= is used to divide a value in a variable.

**Comparison**

The comparison operators are used to compare two values.

**Equal to**(==), the operator == is used to verify if two variable has the same value.

int x = 3;  
int y = 4;  
bool isEqual = x == y;// the isEqual variable will take the value false because those two variables dont have the same value

**Is different**(!=), the operator != is used to verify if two variable has different value.

**Greater than**(>), the operator > is used to verify if one variable has greater value than another variable.

**Greater than or equal** (>=), the operator > = is used to verify if one variable has greater or equal value to another variable.

**Less than**(<), the operator < is used to verify if one variable has less value than another variable.

**Less than or equal** (<=), the operator <= is used to verify if one variable has less or equal value to another variable.

**Conditions**

**if**

The **if**is used to specify a block of code that will be executed if some is true.

int age = 20;if (age > 18)   
{  
 Console.WriteLine("The age is greater than 18");  
}

**… else**

The **else**statement is used to specify a block of code that will be executed if some is false.

if (age > 18)   
{  
 Console.WriteLine("The age is greater than 18");  
}  
else  
{  
 Console.WriteLine("UPS!!! The age is not greater than 18");  
}

**… else if …**

The **else if** statement is used to specify new condition to be verified is the before condition is false.

if (age > 18)   
{  
 Console.WriteLine("The age is greater than 18");  
}  
else if (age > 14)   
{  
 Console.WriteLine("The age is greater than 14");  
}  
else  
{  
 Console.WriteLine("UPS!!! The age is not greater than 14");  
}

**switch case**

The **switch**statement is used to select a code block to execute if some variable value match.

int day = 4;switch (day)   
{  
 case 1:  
 Console.WriteLine("Monday");  
 break; case 2:  
 Console.WriteLine("Tuesday");  
 break; case 3:  
 Console.WriteLine("Wednesday");  
 break; case 4:  
 Console.WriteLine("Thursday");  
 break; case 5:  
 Console.WriteLine("Friday");  
 break; case 6:  
 Console.WriteLine("Saturday");  
 break; case 7:  
 Console.WriteLine("Sunday");  
 break;  
}

**Loop**

With loop we can execute a block of code as long some condition is reached.

**while**

The **while**loop will execute some block repeatedly of code as long a condition is been true.

int i = 0;  
while (i < 5)   
{  
 Console.WriteLine(i);  
 i++;  
}

**do/while**

The **do/while**loop is a variant of while loop, the main difference is that in **do/while** the block of code will be executed at least one time, and it will repeat as long the condition has been true.

int i = 0;  
do   
{  
 Console.WriteLine(i);  
 i++;  
}  
while (i < 5);

**for**

The **for**loop is used when you know the number of times it will repeat.

for (int i = 0; i < 5; i++)   
{  
 Console.WriteLine(i);  
}

**break/continue**

You can use **break**statement to jump out of the loop.

for (int i = 0; i < 10; i++)   
{  
 if (i == 4)   
 {  
 break;  
 }  
 Console.WriteLine(i);  
}

You can use the **continue**statement to jump to the next loop iteration,

for (int i = 0; i < 10; i++)   
{  
 if (i == 4)   
 {  
 continue;  
 }  
 Console.WriteLine(i);  
}

**Function / Methods**

Methods is a block of code that do something and it only run when it is called.

It can receive data (parameters) and can return some value.

You can create your on function, doing the follow code

ReturnDatatype functionName(Parameter1Datatype parameter1Name)  
{  
 // some code  
 return;  
}

Below is a example to create a function to sum two numbers.

// function declaration  
int sum(int x, int y)  
{  
 return x + y;  
}// call the function  
int c = sum(4, 5);// it will return the value of the sum 4 + 5

**Array**

Array is a type of variable that can store more than one values.

To declare a array, we should do as thecode below

DataType[] variableName;

**Examples**

string[] students;// or declare and assign values  
int[] age = {18, 25, 30};

To access or assign value we should refer in which position, the position always start referred in zero and got until n -1, where n is the size of the array.

**Examples**

string[] students = { “John”, “Katty”, “Robert” };// access value in the first position  
Console.WriteLine(students[0]);// change value in the second position  
students[1] = "Kathy";

**Class**

Class is one way to create many objects with same attributes.

Car is class and Volvo is object of the class Car.

Person is a class and John is a object of class Person.

**See how to create a class.**

class Person  
{  
 string name;  
 int age;  
}

**To create a object of a class we can do.**

Person person1 = new Person();  
person1.name = "John";  
person1.age = 25;

# Cheat sheet - C#

#### [BY ADAFACE TEAM](https://www.adaface.com/author/customer/" \t "_blank)

1. Data Types
2. Primitive Size Example
4. String 2 bytes/char s = "reference";
5. bool b = true;
6. char 2 bytes ch = 'a';
7. byte 1 byte b = 0x78;
8. short 2 bytes val = 70;
9. int 4 bytes val = 700;
10. long 8 bytes val = 70;
11. float 4 bytes val = 70.0F;
12. double 8 bytes val = 70.0D;
13. decimal 16 bytes val = 70.0M;
14. Arrays
15. 2.1 Declaration
16. //Initiliazed using a list defined with curly braces
17. int[] nameArray = {100, 101, 102};
18. //Define an empty array
19. int[] nameArray = new int[3]; // 3 rows and 2 columns
20. //To access a specific item in the array
21. int[] nameArray = new int[10];
22. int firstNumber = nameArray[0];
23. nameArray[1] = 20;
24. //Multidimensional arrays
25. int [,] matrix = new int [2,2]
26. matrix[0,0] = 1;
27. matrix[0,1] = 2;
28. matrix[1,0] = 3;
29. matrix[1,1] = 4;
30. int[,] predefinedMatrix = new int[2,2] { { 1, 2 }, { 3, 4 } };
31. 2.2 Array Operations
32. //Sort ascending
33. Array.Sort(nameArray);
34. //Sort begins at element 6 and sorts 20 elements
35. Array.Sort(nameArray,6,20);
36. //Use 1 array as a key & sort 2 arrays
37. string[] values = {"Juan", "Victor", "Elena"};
38. string[] keys = {"Jimenez", "Martin", "Ortiz"};
39. Array.Sort(keys, values);
40. //Clear elements in array (array, first element, # elements)
41. Array.Clear(nameArray, 0, nameArray.Length);
42. //Copy elements from one array to another
43. Array.Copy(scr, target, numOfElements);
44. String Operations
45. //To concatenate between strings, use the plus operator:
46. string firstName = "Erin";
47. string lastName = "Roger";
48. string fullName = firstName + " " + lastName;
49. //To add one string to another, use the += operator:
50. string secondLastName = "Green";
51. string fullName += secondLastName;
53. //ToString function
54. //It converts an object to its string representation so that it is suitable for display
55. Object.ToString();
57. //String formatting
58. //Each additional argument to the function can be referred to in the string using the brackets operator with the index number.
59. String.Format(String format, Object arg0);
60. format - A composite format string that includes one or more format items
61. arg0 - The first or only object to format
62. //Substring
63. //Returns a part of the string, beginning from the index specified as the argument. Substring also accepts a maximum length for the substring
64. String.Substring(beginAt);
65. String.Substring(beginAt, maximum);
67. //Replace
68. string newStr = oldStr.Replace("old","new");
69. //IndexOf
70. //Finds the first ocurrence of a string in a larger string
71. //Returns -1 if the string is not found
72. String.IndexOf(val, start, num)
73. val - string to search for
74. start - where to begin in string
75. //LastIndexOf
76. //Search from end of string
77. //Split
78. //Split is used to break delimited string into substrings
79. String.Split(Char[]);
80. //ToCharArray
81. //Places selected characteres in a string in a char array
82. String str = "AaBbCcDd";
83. //create array of 8 vowels
84. var chars = str.ToCharArray();
85. //create array of 'B' and 'C'
86. var chars = str.ToCharArray(2,2);
87. System.Text.StringBuilder
88. 4.1 Constructor
89. StringBuilder sb = new StringBuilder();
90. StringBuilder sb = new StringBuilder(myString);
91. StringBuilder sb = new StringBuilder(myString, capacity);
92. myString - Initial value of StringBuilder object
93. capacity - Initial size of buffer
94. DateTime
95. 5.1 DateTime Constructor
96. DateTime(year, month, day)
97. DateTime(year, month, day, hour, minute, second)
98. DateTime newYear = DateTime.Parse("1/1/2018"):
99. DateTime currentDate = DateTime.Now;
100. DateTime nextYear = DateTime.AddYears(1);
101. TimeSpan
102. 6.1 TimeSpan Constructor
104. TimpeSpan(hour, minute, sec)
106. TimeSpan timeS = new TimeSpan(10, 14, 50);
107. TimeSpan timeS\_Hours = TimeSpan.FromDays(3640);
108. Formatting Values
109. Format item syntax: {index[,alignment][:format string]}
110. index - Specifies element in list of values to which format is applied
111. aligment - Indicates minimun width (in characters) to display value
112. format string - Contains the code which specififes the format of the displayed value
113. 7.1 Numeric
114. Format Name Pattern Value Result
115. C or c Currency {0:C2}, 1000.1 $ 1000.1 A currency value
116. D or d Decimal {0:D5}, 30 00030 Integer digits with optional negative sign
117. E or e Exponential {0,9:E2}, 120.2 1.20+E002 Exponential notation
118. F or f Fixed-point {0,9:F2}, 120.2 120.2 Integral and decimal digits with optional negative sign
119. G or g General {0,9:G2}, 120.2 120.2 The more compact of either fixed-point or scientific notation
120. N or n Number {0,9:N1}, 1300.5 1,300,5 Integral and decimal digits, group separators, and a decimal separator with optional negative sign
121. P or p Percent {0,9:P3}, .0903 9.03% Number multiplied by 100 and displayed with a percent symbol
122. R or r Round-trip {0,9:R}, 3.1416 3.1316 A string that can round-trip to an identical number
123. X or x Hexadecimal {0,9:X4}, 31 001f A hexadecimal string
124. C# compiler at the Command Line
125. csc File.cs -> Compiles Files.cs producing File.exe
126. csc -target:library File.cs -> Compiles File.cs producing File.dll
127. csc -out:My.exe File.cs -> Compiles File.cs and creates My.exe
128. csc -define:DEBUG -optimize -out:File2.exe \*.cs -> Compiles all the C# files in the current directory with optimizations enabled and defines the DEBUG symbol. The output is File2.exe
129. csc -target:library -out:File2.dll -warn:0 -nologo -debug \*.cs -> Compiles all the C# files in the current directory producing a debug version of File2.dll. No logo and no warnings are displayed
130. csc -target:library -out:Something.xyz \*.cs -> Compiles all the C# files in the current directory to Something.xyz (a DLL)
132. 8.1 Compiler Options Listed
133. Option Purpose
134. @ Reads a response file for more options.
135. -? Displays a usage message to stdout.
136. -additionalfile Names additional files that don't directly affect code generation but may be used by analyzers for producing errors or warnings.
137. -addmodule Links the specified modules into this assembly
138. -analyzer Run the analyzers from this assembly (Short form: -a)
139. -appconfig Specifies the location of app.config at assembly binding time.
140. -baseaddress Specifies the base address for the library to be built.
141. -bugreport Creates a 'Bug Report' file. This file will be sent together with any crash information if it is used with -errorreport:prompt or -errorreport:send.
142. -checked Causes the compiler to generate overflow checks.
143. -checksumalgorithm:<alg> Specifies the algorithm for calculating the source file checksum stored in PDB. Supported values are: SHA1 (default) or SHA256.
144. -codepage Specifies the codepage to use when opening source files.
145. -debug Emits debugging information.
146. -define Defines conditional compilation symbols.
147. -delaysign Delay-signs the assembly by using only the public part of the strong name key.
148. -deterministic Causes the compiler to output an assembly whose binary content is identical across compilations if inputs are identical.
149. -doc Specifies an XML Documentation file to generate.
150. -errorreport Specifies how to handle internal compiler errors: prompt, send, or none. The default is none.
151. -filealign Specifies the alignment used for output file sections.
152. -fullpaths Causes the compiler to generate fully qualified paths.
153. -help Displays a usage message to stdout.
154. -highentropyva Specifies that high entropy ASLR is supported.
155. -incremental Enables incremental compilation [obsolete].
156. -keycontainer Specifies a strong name key container.
157. -keyfile Specifies a strong name key file.
158. -langversion:<string> Specify language version: Default, ISO-1, ISO-2, 3, 4, 5, 6, 7, 7.1, 7.2, 7.3, or Latest
159. -lib Specifies additional directories in which to search for references.
160. -link Makes COM type information in specified assemblies available to the project.
161. -linkresource Links the specified resource to this assembly.
162. -main Specifies the type that contains the entry point (ignore all other possible entry points).
163. -moduleassemblyname Specifies an assembly whose non-public types a .netmodule can access.
164. -modulename:<string> Specify the name of the source module.
165. -noconfig Instructs the compiler not to auto include CSC.RSP file.
166. -nologo Suppresses compiler copyright message.
167. -nostdlib Instructs the compiler not to reference standard library (mscorlib.dll).
168. -nowarn Disables specific warning messages
169. -nowin32manifest Instructs the compiler not to embed an application manifest in the executable file.
170. -optimize Enables/disables optimizations.
171. -out Specifies the output file name (default: base name of file with main class or first file).
172. -parallel[+|-] Specifies whether to use concurrent build (+).
173. -pathmap Specifies a mapping for source path names output by the compiler.
174. -pdb Specifies the file name and location of the .pdb file.
175. -platform Limits which platforms this code can run on: x86, Itanium, x64, anycpu, or anycpu32bitpreferred. The default is anycpu.
176. -preferreduilang Specifies the language to be used for compiler output.
177. -publicsign Apply a public key without signing the assembly, but set the bit in the assembly indicating the assembly is signed.
178. -recurse Includes all files in the current directory and subdirectories according to the wildcard specifications.
179. -reference References metadata from the specified assembly files.
180. -refout Generate a reference assembly in addition to the primary assembly.
181. -refonly Generate a reference assembly instead of a primary assembly.
182. -resource Embeds the specified resource.
183. -ruleset:<file> Specify a ruleset file that disables specific diagnostics.
184. -subsystemversion Specifies the minimum version of the subsystem that the executable file can use.
185. -target Specifies the format of the output file by using one of four options: -target:appcontainerexe, -target:exe, -target:library, -target:module, -target:winexe, -target:winmdobj.
186. -unsafe Allows unsafe code.
187. -utf8output Outputs compiler messages in UTF-8 encoding.
188. -warn Sets the warning level (0-4).
189. -warnaserror Reports specific warnings as errors.
190. -win32icon Uses this icon for the output.
191. -win32manifest Specifies a custom win32 manifest file.
192. -win32res Specifies the win32 resource file (.res).
193. Control flow statements
194. 9.1 Switch
195. switch (expression) {
196. //expression may be integer, string or enum
197. case expression:
198. //statements
199. break/ goto / return()
200. case ..
201. default
202. //statements
203. break/ goto / return()
204. }
205. 9.2 If
206. if (condition) {
207. //statements
208. } else {
209. //statements
210. }
211. Loop
212. 10.1 While
213. while (condition) {body}
214. 10.2 Do while
215. do {body} while condition;
216. 10.3 For
217. for (initializer; termination condition; iteration;) {
218. //statements
219. }
220. 10.4 For each
221. foreach (type identifier in collection) {
222. //statements
223. }
224. Class Definition
225. 11.1 Class
226. public | protected | internal | private
227. abstract | sealed | static
228. class className [:class/interfaces inherited from]
229. 11.2 Constructor
230. [access modifier] className (parameters) [:initializer]
231. initializer -base calls constructor in base class.
232. this calls constuctor within class.
233. public class nameClass : Initializer {
234. public className(dataType param1 , dataType param2, ...) : base(param1, param2)
235. { constructor body }
236. }
237. 11.3 Method
238. [access modifier]
239. static | virtual | override | new | sealed | abstract
240. methodName (parameter list) { body }
241. virtual – method can be overridden in subclass
242. override – overrides virtual method in base class
243. new – hides non-virtual method in base class
244. sealed – prevents derived class from inheriting
245. abstract – must be implemented by subclass
246. Passing parameters:
247. 1. By default, parametres are passed by value
248. 2. Passing by reference: ref, in and out modifers
249. To pass a parameter by reference with the intent of changing the value, use the ref, or out keyword. To pass by reference with the intent of avoiding copying but not changing the value, use the in modifier
250. 11.4 Property
251. [modifier] <dataType> property name{
252. public string BrandName
253. {
254. get { return brandName; }
255. set { brandName = value; }
256. }
257. }
258. Struct
259. 12.1 Defining a structure
260. [attribute][modifier] struct name [:interfaces] { struct-body }
261. 12.2 Class vs Structure
262. -> Classes are reference types and structs are value types
263. -> Structures do not support inheritance
264. -> Structures cannot have default constructor
265. Enum
266. 13.1 Declaring enum variable
267. enum <enumName> {
268. enumeration list
269. };
270. enumName - Specifies the enumeration type name
271. enumeration list is a comma-separated list of identifiers
272. //Each of the symbols in the enumeration list stands for an integer value, one greater than the symbol that precedes it.
273. Delegates
274. //A delegate is a reference type variable that holds the reference to a method. The reference can be changed at runtime.
275. 14.1 Declaring delegates
276. //Delegate declaration determines the methods that can be referenced by the delegate.
277. delegate <return type> <delegate-name> <parameter list>
278. 14.2 Instantiating delegates
279. //When creating a delegate, the argument passed to the new expression is written similar to a method call, but without the arguments to the method
280. public delegate void printString(string s);
281. printString ps1 = new printString(WriteToScreen);
282. printString ps2 = new printString(WriteToFile);
283. Events
284. 15.1 Declaring events
285. //To declare an event inside a class, first a delegate type for the event must be declared.
286. public delegate string MyDelegate(string str);
287. //The event itself is declared by using the event keyword
288. event MyDelegate MyEvent;
289. 15.2 Commonly used Control Events
290. Event Delegate
291. Click, MouseEnter, DoubleClick, MouseLeave EventHandler( object sender, EventArgs e)
292. MouseDown, Mouseup, MouseMove MouseEventHandler(object sender, MouseEventArgs e)
293. e.X, e.Y – x and y coordinates
294. e.Button – MouseButton.Left, Middle, Right
295. KeyUp, KeyDown KeyEventHandler(object sndr, KeyEventArgs e)
296. e.Handled – Indicates whether event is handled.
297. e.KeyCode – Keys enumeration, e.g., Keys.V
298. e.Modifiers – Indicates if Alt, Ctrl, or Shift key.

KeyPress KeyPressEventHandler(object sender, KeyPressEventArgs e)

# C# Best Practices CheatSheet for Developers

## Standard Practices

* D.R.Y - Don't Repeat Yourself
* K.I.S.S - Keep It Simple Stupid
* Always, Always check for Null
* when possible use a USING() statement

### **Naming and Handling Classes**

* Namespace : Company.ProgramLayer
  + Class Naming
* Classes = PascalCase, Nouns
* Constuctors
* Methods = PascalCase, Verbs
* Fields = camelCase, Private
* Properties = PascalCase, at start
* Constants = PascalCase
* Vars = camelCase
  + Access Modifiers
* Keeps all in Private, unless you are desinging otherwise
  + Avoid -Prefixes, Underscores, abbreviations

### **Constructors**

(method of a class that gets created on intialization)

* Provide a Default Constructor
* Consider paramaterized constructors
* Name the parameters the same as related properties
* Avoid - Don't make them do too much work, Simple is key

### **Namespaces**

organizes classes in a heirarchy

* Follow Format:
  + [Comany].[Technology].[Feature]
  + example Microsoft.Media.Design
* PascalCasing
* use the import/using instead of calling directly to a function
* Avoid naming the Namespace the same as the class

### **Static Classes**

a static class cannot be instantiated, good for services (logging, notifications)

* Use sparingly
* use for common utilites
* Avoid as a random bucket

### **Singletons**

* Use when you only need one instance
* Use when you need to create 'child objects'
* Use to Support OOP features
* Avoid if you won't leverage any of the aforementioned features

### **Method Overloading**

* Include comments for the methd and parameters (xml for intellisense)
* Use few Parameters
* Order the parameters consistently
* Avoid overloads with the same name but different purpose, and duplicate Code

### **Method chaining**

when one method calls another method to get most of the work done

* Implement to reduce repeated code
* Avoid if it's overkill and will overcomplicate things

### **Constants vs. Read-only fields**

#### **Constants**

* PascalCasing
* Compile-time constant
* assigned on declaration
* only number, boolean or string
* Always Static

#### **Read-Only**

* PascalCasing
* Runtime constant
* assigned on declaration or constructor
* any data type
* optionally static

### **Property**

flexible mechanism that allows you to, Read, Write Compute it's value on a Private Field

* Lazy-Loading
* Validate incoming values
* Use Relevant Names
* use 'getter' for simple protection, formatting and initilalizing
* use 'setter' for simple protection, formatting and validation
* Avoid abbreviation, complex logic

#### **Test Driven Development**

1. Arrange (create the test expectations)
2. Act (run the desired test / method)
3. Assert (Run the test against the expected and the actual)

**Download C# Cheat Sheet PDF for Your Quick Reference**

Posted in [C#](https://hackr.io/blog/category/c-sharp), [Cheat Sheet](https://hackr.io/blog/category/cheat-sheet)

[Ramya Shankar](https://hackr.io/blog/author/ramyashankar)

Last Updated 23 Feb, 2022

Table of Contents

**C# Introduction**

* Object-oriented language, with syntax similar to C++ and Java.
* Type safe
* Component oriented, structured language
* Automatic garbage collection
* Rich set of libraries
* Conditional compilation

**Syntax**

* Case sensitive
* Comments are typed within // (single-line) or /\*\*/ (multi-line)
* Code is typed inside code blocks {}
* Line termination is done using semicolon ;
* Supports comment task highlighters like TODO: , NOTE: , WARN: etc…

**Variables**

<datatype> <variablename> = <initialvalue>;

* Variables should start with underscore and cannot contain white spaces.
* It can contain numbers but should always start with a capital letter.
* It cannot contain any symbols (other than underscore).

**Naming Conventions**

|  |  |
| --- | --- |
| Class | **S**tudent**C**lass |
| Method | **G**et**M**arks |
| Local variable | **f**irst**N**ame |
| Private variable | **a**vg**M**arks |
| Constant | **P**ercentile |

**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(), IsInt() | Convert string into integer  Check If the input is int | intVal = str.AsInt();  str.IsInt() |
| AsFloat(),  IsFloat() | Convert string into float  Check if the input is float | floatVal = str.AsFloat();  str.IsFloat() |
| AsDecimal()  IsDecimal() | Convert string into decimal  Check if input is decimal | decVal = str.AsDecimal();  str.IsDecimal() |
| AsDateTime()  IsDateTime() | Convert string into datetime type  Check if input is date-time | dateVal = str.AsDateTime();    str.isDateTime(); |
| AsBool() IsBool() | Convert string into Boolean  Check if input is Boolean | boolVal = str.AsBool();  str.IsBool(); |
| ToString() | Convert another data type like int, array, list etc into String | myVal = 1111;  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..  arrVal.ToString() |
| () | Calculations, passing parameters etc…  (i+10)\*(i-10); multiply(i, j) |
| [] | Access values in arrays or collections. name[i] |
| ! | Reversing Boolean value  if (!isMatching) |
| && | Logical AND  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”);  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”;  String sep = {“and”};  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();  dt.ToString(); | gives output as –  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 will 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);  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 number (01, 02) |

**DateTime format specifiers**

|  |  |  |
| --- | --- | --- |
| **Format specifier** | **Name** | **Description** |
| d | Short date pattern | Represents a custom **DateTime** format string defined by the current [ShortDatePattern](https://msdn.microsoft.com/en-us/library/151kkadx(v=vs.85)) property.  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](https://msdn.microsoft.com/en-us/library/e911xxw2(v=vs.85)) property.  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](https://msdn.microsoft.com/en-us/library/s01142hb(v=vs.85)) property.  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](https://msdn.microsoft.com/en-us/library/6s54hbfa(v=vs.85)) property.  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 the formatted string can be parsed back using [Parse](https://msdn.microsoft.com/en-us/library/1k1skd40(v=vs.85)) or [ParseExact](https://msdn.microsoft.com/en-us/library/w2sa9yss(v=vs.85)) with the correct **Kind** property value.  The custom format string is "yyyy'-'MM'-'dd'T'HH':' mm': 'ss.fffffffK."  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](https://msdn.microsoft.com/en-us/library/d632bk6h(v=vs.85)) 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.  The custom format string is "DDD, dd MMM yyyy HH':' mm': 'ss 'GMT'".  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](https://msdn.microsoft.com/en-us/library/tw6x7xba(v=vs.85)) 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.  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](https://msdn.microsoft.com/en-us/library/19985y27(v=vs.85)) 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](https://msdn.microsoft.com/en-us/library/4zwd57t4(v=vs.85)) property.  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](https://msdn.microsoft.com/en-us/library/84bb9x8h(v=vs.85)) 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.  The custom format string is "yyyy'-'MM'-'dd HH':'mm':'ss'Z'".  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.  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](https://msdn.microsoft.com/en-us/library/kfxe1b67(v=vs.85)) property.  For example, the custom format string for the invariant culture is "yyyy MMMM". |
| Custom format | Custom patterns –  "MM'/'dd yyyy"  "dd.MM.yyyy"  "MM.dd.yyyy HH:mm"  "dddd, MMMM (yyyy): HH:mm:ss" | 03/17 2019  17.03.2019  03.17.2019 06:23  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**](https://www.geeksforgeeks.org/c-check-if-an-array-has-fixed-size-or-not/) | checks whether the Array has a fixed size. | string[] arrVal = new string[] {“stud1”, “stud2”, “stud3”};    arrVal.IsFixedSize; |
| [**IsReadOnly**](https://www.geeksforgeeks.org/c-check-if-an-array-is-read-only-or-not/) | Checks whether the Array is read-only. | arrVal.IsReadOnly; |
| [**IsSynchronized**](https://www.geeksforgeeks.org/c-check-if-an-array-is-synchronized-thread-safe-or-not/) | Checks whether access to the Array is synchronized (thread safe). | arrVal.IsSynchronized; |
| [**Length**](https://www.geeksforgeeks.org/how-to-find-the-length-of-an-array-in-c/) | Gets the total number of elements in all the dimensions of the Array. | arrVal.Length; |
| [**LongLength**](https://www.geeksforgeeks.org/c-longlength-property-of-an-array/) | Length in 64-bit integer | arrVal.LongLength; |
| [**Rank**](https://www.geeksforgeeks.org/how-to-find-the-rank-of-an-array-in-c/) | 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**](https://www.geeksforgeeks.org/how-to-get-synchronize-access-to-the-array-in-c-sharp/) | Gets an object used to synchronize Array access | arrVal.SyncRoot; |
| [**AsReadOnly()**](https://www.geeksforgeeks.org/c-array-asreadonlyt-method/) | Returns a read-only wrapper for the specified array. | Array.AsReadOnly(arrVal); |
| [**BinarySearch()**](https://www.geeksforgeeks.org/how-to-use-array-binarysearch-method-in-c-sharp-set-1/) | Searches a value in a one-dimensional sorted array using a [binary search](https://hackr.io/blog/binary-search-in-c) algorithm. | Array.BinarySearch(arrVal, obj); where obj is the object to be searched. |
| [**Clear()**](https://www.geeksforgeeks.org/c-array-clear-method/) | Sets a range of elements in an array to the default value of each element type. | Array.Clear(arrVal, 0, 2);  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()**](https://www.geeksforgeeks.org/c-array-constrainedcopy-method/) | 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);  where srcArr is the source array,  0 is the start index from where copy should begin,  destArr is the destination array,  3 is the place where copy should start in the destination array,  5 is the number of elements to copy |
| [**ConvertAll()**](https://www.geeksforgeeks.org/c-converting-an-array-of-one-type-to-an-array-of-another-type/) | 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);  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);  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()**](https://www.geeksforgeeks.org/c-check-if-an-array-object-is-equal-to-another-array-object/) | Determines whether the specified object is equal to the current object. | arrVal.Equals(arrVal2); |
| [**Exists()**](https://www.geeksforgeeks.org/c-check-if-an-array-contain-the-elements-that-match-the-specified-conditions/) | Determines whether the specified array contains elements that match the conditions defined by the specified predicate. | Array.Exists(srcArr, “<elementname>”); |
| [**Find()**](https://www.geeksforgeeks.org/c-array-find-method/) | 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()**](https://www.geeksforgeeks.org/c-array-findall-method/) | 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()**](https://www.geeksforgeeks.org/c-array-getenumerator-method/) | Returns an IEnumerator for the Array. | arrVal.GetEnumerator() |
| **GetHashCode()** | default hash function. | arrVal.GetHashCode() |
| [**GetLength()**](https://www.geeksforgeeks.org/c-total-number-of-elements-present-in-an-array/) | 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 integer |
| [**GetLongLength()**](https://www.geeksforgeeks.org/total-number-of-elements-in-a-specified-dimension-of-an-array-in-c/) | 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()**](https://www.geeksforgeeks.org/c-finding-the-index-of-first-element-in-the-array/) | 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()**](https://www.geeksforgeeks.org/c-finding-the-index-of-last-element-in-the-array/) | 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()**](https://www.geeksforgeeks.org/array-lastindexof-method-in-c-sharp-set-1/) | 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()**](https://www.geeksforgeeks.org/c-how-to-change-the-size-of-one-dimensional-array/) | Changes the number of elements of a one-dimensional array to the specified new size. | Array.Resize(ref arrVal, len-2);  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()**](https://www.geeksforgeeks.org/how-to-sort-an-array-in-c-sharp-array-sort-method-set-1/) | Sorts the elements in a one-dimensional array. | Array.Sort(arrVal) |
| **ToString()** | Returns a string that represents the current object. (Inherited from Object) | arrVal.ToString() |
| [**TrueForAll()**](https://www.geeksforgeeks.org/c-array-trueforall-method/) | 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) {...} else if (true) {...} else {...} |
| switch | switch (var)  { case 1: break;  case 2: break; default: break;  } |
| for | for (int i =0; i <=len; i++) {...} |
| foreach-in | foreach (int item in array) {...} |
| while | while (true) {...} |
| do... while | do {...} while (true); |
| try-catch-finally | try {...} catch (Exception e) {...} catch {...} 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(){  return val;  } |
| passing parameters | public void MyMethod(String s, int i) {  } |

**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.

76.4K

PHP Design patterns in Laminas - How Laminas uses them internally | Advanced PHP

|  |  |
| --- | --- |
| // 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)  Console.WriteLines(File.ReadAllLines(path)  // 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) |

C#cheatsheet

C# quick reference cheat sheet that provides basic syntax and methods.

[#](https://quickref.me/cs#getting-started)Getting started

**Hello.cs**

class Hello {

// main method

static void Main(string[] args)

{

// Output: Hello, world!

Console.WriteLine("Hello, world!");

}

}

Compiling and running (make sure you are in the project directory)

$ dotnet run

Hello, world!

**Variables**

int intNum = 9;

long longNum = 9999999;

float floatNum = 9.99F;

double doubleNum = 99.999;

decimal decimalNum = 99.9999M;

char letter = 'D';

bool @bool = true;

string site = "quickref.me";

var num = 999;

var str = "999";

var bo = false;

**Primitive Data Types**

| Data Type | Size | Range |
| --- | --- | --- |
| int | 4 bytes | -231 to 231-1 |
| long | 8 bytes | -263 to 263-1 |
| float | 4 bytes | 6 to 7 decimal digits |
| double | 8 bytes | 15 decimal digits |
| decimal | 16 bytes | 28 to 29 decimal digits |
| char | 2 bytes | 0 to 65535 |
| bool | 1 bit | true / false |
| string | 2 bytes per char | N/A |

**Comments**

// Single-line comment

/\* Multi-line

comment \*/

// TODO: Adds comment to a task list in Visual Studio

/// Single-line comment used for documentation

/\*\* Multi-line comment

used for documentation \*\*/

**Strings**

string first = "John";

string last = "Doe";

// string concatenation

string name = first + " " + last;

Console.WriteLine(name); // => John Doe

See: [Strings](https://quickref.me/cs#c-strings)

**User Input**

Console.WriteLine("Enter number:");

if(int.TryParse(Console.ReadLine(),out int input))

{

// Input validated

Console.WriteLine($"You entered {input}");

}

**Conditionals**

int j = 10;

if (j == 10) {

Console.WriteLine("I get printed");

} else if (j > 10) {

Console.WriteLine("I don't");

} else {

Console.WriteLine("I also don't");

}

**Arrays**

char[] chars = new char[10];

chars[0] = 'a';

chars[1] = 'b';

string[] letters = {"A", "B", "C"};

int[] mylist = {100, 200};

bool[] answers = {true, false};

**Loops**

int[] numbers = {1, 2, 3, 4, 5};

for(int i = 0; i < numbers.Length; i++) {

Console.WriteLine(numbers[i]);

}

foreach(int num in numbers) {

Console.WriteLine(num);

}

[#](https://quickref.me/cs#c-strings)C# Strings

**String concatenation**

string first = "John";

string last = "Doe";

string name = first + " " + last;

Console.WriteLine(name); // => John Doe

**String interpolation**

string first = "John";

string last = "Doe";

string name = $"{first} {last}";

Console.WriteLine(name); // => John Doe

**String Members**

| Member | Description |
| --- | --- |
| Length | A property that returns the length of the string. |
| Compare() | A static method that compares two strings. |
| Contains() | Determines if the string contains a specific substring. |
| Equals() | Determines if the two strings have the same character data. |
| Format() | Formats a string via the {0} notation and by using other primitives. |
| Trim() | Removes all instances of specific characters from trailing and leading characters. Defaults to removing leading and trailing spaces. |
| Split() | Removes the provided character and creates an array out of the remaining characters on either side. |

**Verbatim strings**

string longString = @"I can type any characters in here !#@$%^&\*()\_\_+ '' \n \t except double quotes and I will be taken literally. I even work with multiple lines.";

**Member Example**

// Using property of System.String

string lengthOfString = "How long?";

lengthOfString.Length // => 9

// Using methods of System.String

lengthOfString.Contains("How"); // => true

[#](https://quickref.me/cs#misc)Misc

**General .NET Terms**

| Term | Definition |
| --- | --- |
| Runtime | A collection of services that are required to execute a given compiled unit of code. |
| Common Language Runtime (CLR) | Primarily locates, loads, and managed .NET objects. The CLR also handles memory management, application hosting, coordination of threads, performaing security checks, and other low-level details. |
| Managed code | Code that compiles and runs on .NET runtime. C#/F#/VB are examples. |
| Unmanaged code | Code that compiles straight to machine code and cannot be directly hosted by the .NET runtime. Contains no free memory management, garbage collection, etc. DLLs created from C/C++ are examples. |

[**HEAT SHEET**](https://simplecheatsheet.com/)

Free Cheat Sheets

**Search**

**C# Cheat Sheet**

The**C# cheat sheet**includes symbol syntax and methods to help you using C#.**C#** is a general-purpose, modern and object-oriented programming language pronounced as “C sharp”. C# is a hybrid of C and C++, it is a Microsoft programming language developed to compete with Sun’s Java language. C# is an object-oriented programming language used with XML-based Web services on the .NET platform and designed for improving productivity in the development of Web applications.

[**C#: Data Types Cheat Sheet**](https://simplecheatsheet.com/c-data-types/)

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **Represents** | **Range** | **Default Value** |
| bool | Boolean value | True or False | False |
| byte | 8-bit unsigned integer | 0 to 255 | 0 |
| char | 16-bit Unicode character | U +0000 to U +ffff | ‘\0’ |
| decimal | 128-bit precise decimal values with 28-29 significant digits | (-7.9 x 1028 to 7.9 x 1028) / 100 to 28 | 0.0M |
| double | 64-bit double-precision floating point type | (+/-)5.0 x 10-324 to (+/-)1.7 x 10308 | 0.0D |
| float | 32-bit single-precision floating point type | -3.4 x 1038 to + 3.4 x 1038 | 0.0F |
| int | 32-bit signed integer type | -2,147,483,648 to 2,147,483,647 | 0 |
| long | 64-bit signed integer type | -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 | 0L |
| sbyte | 8-bit signed integer type | -128 to 127 | 0 |
| short | 16-bit signed integer type | -32,768 to 32,767 | 0 |
| uint | 32-bit unsigned integer type | 0 to 4,294,967,295 | 0 |
| ulong | 64-bit unsigned integer type | 0 to 18,446,744,073,709,551,615 | 0 |
| ushort | 16-bit unsigned integer type | 0 to 65,535 | 0 |

[**C#: Type Conversion Cheat Sheet**](https://simplecheatsheet.com/c-type-conversion/)

|  |  |
| --- | --- |
| ToBoolean | Converts a type to a Boolean value, where possible. |
| ToByte | Converts a type to a byte. |
| ToChar | Converts a type to a single Unicode character, where possible. |
| ToDateTime | Converts a type (integer or string type) to date-time structures. |
| ToDecimal | Converts a floating-point or integer type to a decimal type. |
| ToDouble | Converts a type to a double type. |
| ToInt16 | Converts a type to a 16-bit integer. |
| ToInt32 | Converts a type to a 32-bit integer. |
| ToInt64 | Converts a type to a 64-bit integer. |
| ToSbyte | Converts a type to a signed byte type. |
| ToSingle | Converts a type to a small floating-point number. |
| ToString | Converts a type to a string. |
| ToType | Converts a type to a specified type. |
| ToUInt16 | Converts a type to an unsigned int type. |
| ToUInt32 | Converts a type to an unsigned long type. |
| ToUInt64 | Converts a type to an unsigned big integer. |

[**C#: Variables Cheat Sheet**](https://simplecheatsheet.com/c-variables/)

|  |  |
| --- | --- |
| Integral types | sbyte, byte, short, ushort, int, uint, long, ulong, and char |
| Floating point types | float and double |
| Decimal types | decimal |
| Boolean types | true or false values, as assigned |
| Nullable types | Nullable data types |

**Defining**

<data\_type> <variable\_list>;

Some valid variable

int i, j, k;

char c, ch;

float f, salary;

double d;

**Initializing**

variable\_name = value;

Example

using System;

namespace VariableDefinition {

class Program {

static void Main(string[] args) {

short a;

int b ;

double c;

/\* actual initialization \*/

a = 10;

b = 20;

c = a + b;

Console.WriteLine("a = {0}, b = {1}, c = {2}", a, b, c);

Console.ReadLine();

}

}

}

=> output

a = 10, b = 20, c = 30

[**C#: Constants Cheat Sheet**](https://simplecheatsheet.com/c-constants/)

|  |  |
| --- | --- |
| \\ | \ character |
| \' | ‘ character |
| \" | ” character |
| \? | ? character |
| \a | Alert or bell |
| \b | Backspace |
| \f | Form feed |
| \n | Newline |
| \r | Carriage return |
| \t | Horizontal tab |
| \v | Vertical tab |
| \xhh . . . | Hexadecimal number of one or more digits |

**Example**

using System;

namespace EscapeChar {

class Program {

static void Main(string[] args) {

Console.WriteLine("Hello\tWorld\n\n");

Console.ReadLine();

}

}

}

=> output

Hello World

[**C#: Operators Cheat Sheet**](https://simplecheatsheet.com/c-operators/)

**# Arithmetic Operators**

|  |  |
| --- | --- |
| + | Adds two operands |
| - | Subtracts second operand from the first |
| \* | Multiplies both operands |
| / | Divides numerator by de-numerator |
| % | Modulus Operator and remainder of after an integer division |
| ++ | Increment operator increases integer value by one |
| -- | Decrement operator decreases integer value by one |

**# Relational Operators**

|  |  |
| --- | --- |
| == | Checks if the values of two operands are equal or not, if yes then condition becomes true. |
| != | Checks if the values of two operands are equal or not, if values are not equal then condition becomes true. |
| > | Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true. |
| < | Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true. |
| >= | Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true. |
| <= | Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true. |

**# Logical Operators**

|  |  |
| --- | --- |
| && | Called Logical AND operator. If both the operands are non zero then condition becomes true. |
| || | Called Logical OR Operator. If any of the two operands is non zero then condition becomes true. |
| ! | Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false. |

**# Assignment Operators**

|  |  |
| --- | --- |
| = | Simple assignment operator, Assigns values from right side operands to left side operand |
| += | Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand |
| -= | Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand |
| \*= | Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand |
| /= | Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand |
| %= | Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand |
| <<= | Left shift AND assignment operator |
| >>= | Right shift AND assignment operator |
| &= | Bitwise AND assignment operator |
| ^= | bitwise exclusive OR and assignment operator |
| |= | bitwise inclusive OR and assignment operator |

**# Miscellaneous Operators**

|  |  |
| --- | --- |
| sizeof() | Returns the size of a data type. |
| typeof() | Returns the type of a class. |
| & | Returns the address of an variable. |
| \* | Pointer to a variable. |
| ? : | Conditional Expression |
| is | Determines whether an object is of a certain type. |
| as | Cast without raising an exception if the cast fails. |

[**C#: nested switch Statements Cheat Sheet**](https://simplecheatsheet.com/c-nested-switch-statements/)

switch(ch1) {

case 'A':

Console.WriteLine("This A is part of outer switch" );

switch(ch2) {

case 'A':

Console.WriteLine("This A is part of inner switch" );

break;

case 'B': /\* inner B case code \*/

}

break;

case 'B': /\* outer B case code \*/

}

**Example**

using System;

namespace DecisionMaking {

class Program {

static void Main(string[] args) {

int a = 100;

int b = 200;

switch (a) {

case 100:

Console.WriteLine("This is part of outer switch ");

switch (b) {

case 200:

Console.WriteLine("This is part of inner switch ");

break;

}

break;

}

Console.WriteLine("Exact value of a is : {0}", a);

Console.WriteLine("Exact value of b is : {0}", b);

Console.ReadLine();

}

}

}

=> output

This is part of outer switch

This is part of inner switch

Exact value of a is : 100

Exact value of b is : 200

[**C#: Switch Statement Cheat Sheet**](https://simplecheatsheet.com/c-switch-statement/)

switch(expression) {

case constant-expression1 :

statement(s);

break;

case constant-expression2 :

case constant-expression3 :

statement(s);

break;

/\* you can have any number of case statements \*/

default : /\* Optional \*/

statement(s);

}

**Example**

using System;

namespace DecisionMaking {

class Program {

static void Main(string[] args) {

/\* local variable definition \*/

char grade = 'B';

switch (grade) {

case 'A':

Console.WriteLine("Excellent!");

break;

case 'B':

case 'C':

Console.WriteLine("Well done");

break;

case 'D':

Console.WriteLine("You passed");

break;

case 'F':

Console.WriteLine("Better try again");

break;

default:

Console.WriteLine("Invalid grade");

break;

}

Console.WriteLine("Your grade is {0}", grade);

Console.ReadLine();

}

}

}

=> output

Well done

Your grade is B

[**C#: Nested if Statements Cheat Sheet**](https://simplecheatsheet.com/c-nested-if-statements/)

if( boolean\_expression 1) {

/\* Executes when the boolean expression 1 is true \*/

if(boolean\_expression 2) {

/\* Executes when the boolean expression 2 is true \*/

}

}

**Example**

using System;

namespace DecisionMaking {

class Program {

static void Main(string[] args) {

//\* local variable definition \*/

int a = 100;

int b = 200;

/\* check the boolean condition \*/

if (a == 100) {

/\* if condition is true then check the following \*/

if (b == 200) {

/\* if condition is true then print the following \*/

Console.WriteLine("Value of a is 100 and b is 200");

}

}

Console.WriteLine("Exact value of a is : {0}", a);

Console.WriteLine("Exact value of b is : {0}", b);

Console.ReadLine();

}

}

}

=> output

Value of a is 100 and b is 200

Exact value of a is : 100

Exact value of b is : 200

[**C#: if…else if…else Statement Cheat Sheet**](https://simplecheatsheet.com/c-if-else-if-else-statement/)

if(boolean\_expression 1) {

/\* Executes when the boolean expression 1 is true \*/

}

else if( boolean\_expression 2) {

/\* Executes when the boolean expression 2 is true \*/

}

else if( boolean\_expression 3) {

/\* Executes when the boolean expression 3 is true \*/

} else {

/\* executes when the none of the above condition is true \*/

}

**Example**

using System;

namespace DecisionMaking {

class Program {

static void Main(string[] args) {

/\* local variable definition \*/

int a = 100;

/\* check the boolean condition \*/

if (a == 10) {

/\* if condition is true then print the following \*/

Console.WriteLine("Value of a is 10");

}

else if (a == 20) {

/\* if else if condition is true \*/

Console.WriteLine("Value of a is 20");

}

else if (a == 30) {

/\* if else if condition is true \*/

Console.WriteLine("Value of a is 30");

} else {

/\* if none of the conditions is true \*/

Console.WriteLine("None of the values is matching");

}

Console.WriteLine("Exact value of a is: {0}", a);

Console.ReadLine();

}

}

}

=> output

None of the values is matching

Exact value of a is: 100

[**C#: if…else Statement Cheat Sheet**](https://simplecheatsheet.com/c-if-else-statement/)

if(boolean\_expression) {

/\* statement(s) will execute if the boolean expression is true \*/

} else {

/\* statement(s) will execute if the boolean expression is false \*/

}

**Example**

[Live Demo](http://tpcg.io/DeLC15)

using System;

namespace DecisionMaking {

class Program {

static void Main(string[] args) {

/\* local variable definition \*/

int a = 100;

/\* check the boolean condition \*/

if (a < 20) {

/\* if condition is true then print the following \*/

Console.WriteLine("a is less than 20");

} else {

/\* if condition is false then print the following \*/

Console.WriteLine("a is not less than 20");

}

Console.WriteLine("value of a is : {0}", a);

Console.ReadLine();

}

}

}

=> output

a is not less than 20;

value of a is : 100

[**C#: if Statement Cheat Sheet**](https://simplecheatsheet.com/c-if-statement/)

if(boolean\_expression) {

/\* statement(s) will execute if the boolean expression is true \*/

}

**Example**

using System;

namespace DecisionMaking {

class Program {

static void Main(string[] args) {

/\* local variable definition \*/

int a = 10;

/\* check the boolean condition using if statement \*/

if (a < 20) {

/\* if condition is true then print the following \*/

Console.WriteLine("a is less than 20");

}

Console.WriteLine("value of a is : {0}", a);

Console.ReadLine();

}

}

}

=> output

a is less than 20;

value of a is : 10

[**C#: Nested Loops Cheat Sheet**](https://simplecheatsheet.com/c-nested-loops/)

do {

statement(s);

do {

statement(s);

}

while( condition );

}

while( condition );

**Example**

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

[**C#: Do…While Loop Cheat Sheet**](https://simplecheatsheet.com/c-do-while-loop/)

do {

statement(s);

} while( condition );

**Example**

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

[**C#: For Loop Cheat Sheet**](https://simplecheatsheet.com/c-for-loop/)

for ( init; condition; increment ) {

statement(s);

}

**Example**

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

[**C#: While Loop Cheat Sheet**](https://simplecheatsheet.com/c-while-loop/)

while(condition) {

statement(s);

}

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

[**C#: Methods Cheat Sheet**](https://simplecheatsheet.com/c-methods/)

class Program

{

static void MyMethod()

{

// code to be executed

}

}

**# Call a Methods**

using System;

namespace MyApplication

{

class Program

{

static void MyMethod()

{

Console.WriteLine("I have a red banana");

}

static void Main(string[] args)

{

MyMethod();

}

}

}

=> output

I have a red banana

[**C#: Arrays Cheat Sheet**](https://simplecheatsheet.com/c-arrays/)

**# Multidimensional Arrays**

string [,] names;

**Example**

using System;

namespace ArrayApplication {

class MyArray {

static void Main(string[] args) {

/\* an array with 5 rows and 2 columns\*/

int[,] a = new int[5, 2] {{0,0}, {1,2}, {2,4}, {3,6}, {4,8} };

int i, j;

/\* output each array element's value \*/

for (i = 0; i < 5; i++) {

for (j = 0; j < 2; j++) {

Console.WriteLine("a[{0},{1}] = {2}", i, j, a[i,j]);

}

}

Console.ReadKey();

}

}

}

=> output

a[0,0]: 0

a[0,1]: 0

a[1,0]: 1

a[1,1]: 2

a[2,0]: 2

a[2,1]: 4

a[3,0]: 3

a[3,1]: 6

a[4,0]: 4

a[4,1]: 8

**# Jagged Arrays**

int [][] scores;

**Example**

using System;

namespace ArrayApplication {

class MyArray {

static void Main(string[] args) {

/\* a jagged array of 5 array of integers\*/

int[][] a = new int[][]{new int[]{0,0},new int[]{1,2},

new int[]{2,4},new int[]{ 3, 6 }, new int[]{ 4, 8 } };

int i, j;

/\* output each array element's value \*/

for (i = 0; i < 5; i++) {

for (j = 0; j < 2; j++) {

Console.WriteLine("a[{0}][{1}] = {2}", i, j, a[i][j]);

}

}

Console.ReadKey();

}

}

}

=> output

a[0][0]: 0

a[0][1]: 0

a[1][0]: 1

a[1][1]: 2

a[2][0]: 2

a[2][1]: 4

a[3][0]: 3

a[3][1]: 6

a[4][0]: 4

a[4][1]: 8

**# Passing Arrays as Function Arguments**

using System;

namespace ArrayApplication {

class MyArray {

double getAverage(int[] arr, int size) {

int i;

double avg;

int sum = 0;

for (i = 0; i < size; ++i) {

sum += arr[i];

}

avg = (double)sum / size;

return avg;

}

static void Main(string[] args) {

MyArray app = new MyArray();

/\* an int array with 5 elements \*/

int [] balance = new int[]{1000, 2, 3, 17, 50};

double avg;

/\* pass pointer to the array as an argument \*/

avg = app.getAverage(balance, 5 ) ;

/\* output the returned value \*/

Console.WriteLine( "Average value is: {0} ", avg );

Console.ReadKey();

}

}

}

=> output

Average value is: 214.4

**# Param Arrays**

using System;

namespace ArrayApplication {

class ParamArray {

public int AddElements(params int[] arr) {

int sum = 0;

foreach (int i in arr) {

sum += i;

}

return sum;

}

}

class TestClass {

static void Main(string[] args) {

ParamArray app = new ParamArray();

int sum = app.AddElements(512, 720, 250, 567, 889);

Console.WriteLine("The sum is: {0}", sum);

Console.ReadKey();

}

}

}

=> output

The sum is: 2938

**# Array Class**

**Properties**

|  |  |
| --- | --- |
| IsFixedSize | Gets a value indicating whether the Array has a fixed size. |
| IsReadOnly | Gets a value indicating whether the Array is read-only. |
| Length | Gets a 32-bit integer that represents the total number of elements in all the dimensions of the Array. |
| LongLength | Gets a 64-bit integer that represents the total number of elements in all the dimensions of the Array. |
| Rank | Gets the rank (number of dimensions) of the Array. |

**Methods**

|  |  |
| --- | --- |
| Clear | Sets a range of elements in the Array to zero, to false, or to null, depending on the element type. |
| Copy(Array, Array, Int32) | Copies a range of elements from an Array starting at the first element and pastes them into another Array starting at the first element. The length is specified as a 32-bit integer. |
| CopyTo(Array, Int32) | Copies all the elements of the current one-dimensional Array to the specified one-dimensional Array starting at the specified destination Array index. The index is specified as a 32-bit integer. |
| GetLength | Gets a 32-bit integer that represents the number of elements in the specified dimension of the Array. |
| GetLongLength | Gets a 64-bit integer that represents the number of elements in the specified dimension of the Array. |
| GetLowerBound | Gets the lower bound of the specified dimension in the Array. |
| GetType | Gets the Type of the current instance. (Inherited from Object.) |
| GetUpperBound | Gets the upper bound of the specified dimension in the Array. |
| GetValue(Int32) | Gets the value at the specified position in the one-dimensional Array. The index is specified as a 32-bit integer. |
| IndexOf(Array, Object) | Searches for the specified object and returns the index of the first occurrence within the entire one-dimensional Array. |
| Reverse(Array) | Reverses the sequence of the elements in the entire one-dimensional Array. |
| SetValue(Object, Int32) | Sets a value to the element at the specified position in the one-dimensional Array. The index is specified as a 32-bit integer. |
| Sort(Array) | Sorts the elements in an entire one-dimensional Array using the IComparable implementation of each element of the Array. |
| ToString | Returns a string that represents the current object. (Inherited from Object.) |

[**C#: Strings Cheat Sheet**](https://simplecheatsheet.com/c-strings/)

**# Properties**

|  |  |
| --- | --- |
| Chars | Gets the *Char* object at a specified position in the current *String* object. |
| Length | Gets the number of characters in the current String object. |

**# Methods**

|  |  |
| --- | --- |
| public static int Compare(string strA, string strB) | Compares two specified string objects and returns an integer that indicates their relative position in the sort order. |
| public static int Compare(string strA, string strB, bool ignoreCase ) | Compares two specified string objects and returns an integer that indicates their relative position in the sort order. However, it ignores the case if the Boolean parameter is true. |
| public static string Concat(string str0, string str1) | Concatenates two string objects. |
| public static string Concat(string str0, string str1, string str2) | Concatenates three string objects. |
| public static string Concat(string str0, string str1, string str2, string str3) | Concatenates four-string objects. |
| public bool Contains(string value) | Returns a value indicating whether the specified String object occurs within this string. |
| public static string Copy(string str) | Creates a new String object with the same value as the specified string. |
| public void CopyTo(int sourceIndex, char[] destination, int destinationIndex, int count) | Copies a specified number of characters from a specified position of the String object to a specified position in an array of Unicode characters. |
| public bool EndsWith(string value) | Determines whether the end of the string object matches the specified string. |
| public bool Equals(string value) | Determines whether the current String object and the specified String object have the same value. |
| public static bool Equals(string a, string b) | Determines whether two specified String objects have the same value. |
| public static string Format(string format, Object arg0) | Replaces one or more format items in a specified string with the string representation of a specified object. |
| public int IndexOf(char value) | Returns the zero-based index of the first occurrence of the specified Unicode character in the current string. |
| public int IndexOf(string value) | Returns the zero-based index of the first occurrence of the specified string in this instance. |
| public int IndexOf(char value, int startIndex) | Returns the zero-based index of the first occurrence of the specified Unicode character in this string, starting the search at the specified character position. |
| public int IndexOf(string value, int startIndex) | Returns the zero-based index of the first occurrence of the specified string in this instance, starting the search at the specified character position. |
| public int IndexOfAny(char[] anyOf) | Returns the zero-based index of the first occurrence in this instance of any character in a specified array of Unicode characters. |
| public int IndexOfAny(char[] anyOf, int startIndex) | Returns the zero-based index of the first occurrence in this instance of any character in a specified array of Unicode characters, starting the search at the specified character position. |
| public string Insert(int startIndex, string value) | Returns a new string in which a specified string is inserted at a specified index position in the current string object. |
| public static bool IsNullOrEmpty(string value) | Indicates whether the specified string is null or an Empty string. |
| public static string Join(string separator, params string[] value) | Concatenates all the elements of a string array, using the specified separator between each element. |
| public static string Join(string separator, string[] value, int startIndex, int count) | Concatenates the specified elements of a string array, using the specified separator between each element. |
| public int LastIndexOf(char value) | Returns the zero-based index position of the last occurrence of the specified Unicode character within the current string object. |
| public int LastIndexOf(string value) | Returns the zero-based index position of the last occurrence of a specified string within the current string object. |
| public string Remove(int startIndex) | Removes all the characters in the current instance, beginning at a specified position and continuing through the last position, and returns the string. |
| public string Remove(int startIndex, int count) | Removes the specified number of characters in the current string beginning at a specified position and returns the string. |
| public string Replace(char oldChar, char newChar) | Replaces all occurrences of a specified Unicode character in the current string object with the specified Unicode character and returns the new string. |
| public string Replace(string oldValue, string newValue) | Replaces all occurrences of a specified string in the current string object with the specified string and returns the new string. |
| public string[] Split(params char[] separator) | Returns a string array that contains the substrings in the current string object, delimited by elements of a specified Unicode character array. |
| public string[] Split(char[] separator, int count) | Returns a string array that contains the substrings in the current string object, delimited by elements of a specified Unicode character array. The int parameter specifies the maximum number of substrings to return. |
| public bool StartsWith(string value) | Determines whether the beginning of this string instance matches the specified string. |
| public char[] ToCharArray() | Returns a Unicode character array with all the characters in the current string object. |
| public char[] ToCharArray(int startIndex, int length) | Returns a Unicode character array with all the characters in the current string object, starting from the specified index and up to the specified length. |
| public string ToLower() | Returns a copy of this string converted to lowercase. |
| public string ToUpper() | Returns a copy of this string converted to uppercase. |
| public string Trim() | Removes all leading and trailing white-space characters from the current String object. |

**Example**

using System;

namespace MyApplication

{

class Program

{

static void Main(string[] args)

{

string firstName = "Hanah";

string lastName = "Kim";

string name = firstName + lastName;

Console.WriteLine(name);

}

}

}

=> output

Hanah Kim

[**C#: Enums Cheat Sheet**](https://simplecheatsheet.com/c-enums/)

enum <enum\_name> {

enumeration list

};

* The enum\_name specifies the enumeration type name.
* The enumeration list is a comma-separated list of identifiers.

**Example**

using System;

namespace EnumApplication {

class EnumProgram {

enum Colors { Red, Pink, Blue, Black, White, Purple };

static void Main(string[] args) {

int FistColor = (int)Colors.Red;

int LastColor = (int)Days.Purple;

Console.WriteLine("Red: {0}", FistColor);

Console.WriteLine("Purple: {0}", LastColor);

Console.ReadKey();

}

}

}

=> output

Red: 0

Purple: 5

[**C#: Classes and Objects Cheat Sheet**](https://simplecheatsheet.com/c-classes-and-objects/)

**# Classes**

class Car

{

string color = "red";

}

**# Object**

class Car

{

string color = "red";

static void Main(string[] args)

{

Car myObj = new Car();

Console.WriteLine(myObj.color);

}

}

**#Multiple Objects**

class Car

{

string color = "red";

static void Main(string[] args)

{

Car myObj1 = new Car();

Car myObj2 = new Car();

Console.WriteLine(myObj1.color);

Console.WriteLine(myObj2.color);

}

}

[**C#: Inheritance Cheat Sheet**](https://simplecheatsheet.com/c-inheritance/)

<access\_modifier> class <base\_class\_name>

{

    // Base class Implementation

}

<access\_modifier> class <derived\_class\_name> : <base\_class\_name>

{

    // Derived class implementation

}

**Simple example of implementing inheritance in C#**

public class X

{

    public void GetDetails()

    {

        // Method implementation

    }

}

public class Y : X

{

    // your class implementation

}

class Program

{

    static void Main(string[] args)

    {

        Y y = new Y();

        y.GetDetails();

    }

}

**Example**

class Dog

{

public string type = "Poodle";

public void sound()

{

Console.WriteLine("woof, woof!");

}

}

class Poodle : Dog

{

public string typeName = "Toy";

}

class Program

{

static void Main(string[] args)

{

Poodle myPoodle = new Poodle();

myPoodle.sound();

Console.WriteLine(myPoodle.typeName + " " + myPoodle.type);

}

}

=> output

woof, woof!

Toy Poodle

[**C#: Operator Overloading Cheat Sheet**](https://simplecheatsheet.com/c-operator-overloading/)

**Example**

using System;

namespace OperatorOvlApplication {

class Box {

private double length; // Length of a box

private double breadth; // Breadth of a box

private double height; // Height of a box

public double getVolume() {

return length \* breadth \* height;

}

public void setLength( double len ) {

length = len;

}

public void setBreadth( double bre ) {

breadth = bre;

}

public void setHeight( double hei ) {

height = hei;

}

// Overload + operator to add two Box objects.

public static Box operator+ (Box b, Box c) {

Box box = new Box();

box.length = b.length + c.length;

box.breadth = b.breadth + c.breadth;

box.height = b.height + c.height;

return box;

}

}

class Tester {

static void Main(string[] args) {

Box Box1 = new Box(); // Declare Box1 of type Box

Box Box2 = new Box(); // Declare Box2 of type Box

Box Box3 = new Box(); // Declare Box3 of type Box

double volume = 0.0; // Store the volume of a box here

// box 1 specification

Box1.setLength(6.0);

Box1.setBreadth(7.0);

Box1.setHeight(5.0);

// box 2 specification

Box2.setLength(12.0);

Box2.setBreadth(13.0);

Box2.setHeight(10.0);

// volume of box 1

volume = Box1.getVolume();

Console.WriteLine("Volume of Box1 : {0}", volume);

// volume of box 2

volume = Box2.getVolume();

Console.WriteLine("Volume of Box2 : {0}", volume);

// Add two object as follows:

Box3 = Box1 + Box2;

// volume of box 3

volume = Box3.getVolume();

Console.WriteLine("Volume of Box3 : {0}", volume);

Console.ReadKey();

}

}

}

=> output

Volume of Box1 : 210

Volume of Box2 : 1560

Volume of Box3 : 5400

**Overloadable and Non-Overloadable Operators**

|  |  |
| --- | --- |
| +, -, !, ~, ++, -- | These unary operators take one operand and can be overloaded. |
| +, -, \*, /, % | These binary operators take one operand and can be overloaded. |
| ==, !=, <, >, <=, >= | The comparison operators can be overloaded. |
| &&, || | The conditional logical operators cannot be overloaded directly. |
| +=, -=, \*=, /=, %= | The assignment operators cannot be overloaded. |
| =, ., ?:, ->, new, is, sizeof, typeof | These operators cannot be overloaded. |

[**C#: Interfaces Cheat Sheet**](https://simplecheatsheet.com/c-interfaces/)

// interface interface Animal

{

void animalSound(); // interface method (does not have a body)

void run(); // interface method (does not have a body)

}

**Example**

interface IAnimal

{

void animalSound();

}

class Dog : IAnimal

{

public void animalSound()

{

Console.WriteLine("The dog says: woof woof");

}

}

class Program

{

static void Main(string[] args)

{

Dog myDog = new Dog();

myDog.animalSound();

}

}

=> output:

The dog says: woof woof

[**C#: Namespaces Cheat Sheet**](https://simplecheatsheet.com/c-namespaces/)

**# Defining a Namespace**

namespace namespace\_name {

// code declarations

}

To call the namespace-enabled version of either function or variable:

namespace\_name.item\_name;

**# The using Keyword**

Console.WriteLine ("Hello there");

Fully qualified name

System.Console.WriteLine("Hello there");

**# Nested Namespaces**

namespace namespace\_name1 {

// code declarations

namespace namespace\_name2 {

// code declarations

}

}

[**C#: Preprocessor Directives Cheat Sheet**](https://simplecheatsheet.com/c-preprocessor-directives/)

**# Preprocessor Directives in C#**

|  |  |
| --- | --- |
| #define | It defines a sequence of characters, called symbol. |
| #undef | It allows you to undefine a symbol. |
| #if | It allows testing a symbol or symbols to see if they evaluate to true. |
| #else | It allows creating a compound conditional directive, along with #if. |
| #elif | It allows for creating a compound conditional directive. |
| #endif | Specifies the end of a conditional directive. |
| #line | It lets you modify the compiler’s line number and (optionally) the file name output for errors and warnings. |
| #error | It allows generating an error from a specific location in your code. |
| #warning | It allows generating a level one warning from a specific location in your code. |
| #region | It lets you specify a block of code that you can expand or collapse when using the outlining feature of the Visual Studio Code Editor. |
| #endregion | It marks the end of a #region block. |

**# The #define Preprocessor**

#define symbol

**# Conditional Directives**

#if symbol [operator symbol]...

Where, symbol is the name of the symbol you want to test. You can also use true and false or prepend the symbol with the negation operator.

The operator symbol is the operator used for evaluating the symbol. Operators could be either of the following:

* == (equality)
* != (inequality)
* && (and)
* || (or)

[**C#: Regular Expressions Cheat Sheet**](https://simplecheatsheet.com/c-regular-expressions/)

**The Regex Class**

|  |  |
| --- | --- |
| public bool IsMatch(string input) | Indicates whether the regular expression specified in the Regex constructor finds a match in a specified input string. |
| public bool IsMatch(string input, int startat) | Indicates whether the regular expression specified in the Regex constructor finds a match in the specified input string, beginning at the specified starting position in the string. |
| public static bool IsMatch(string input, string pattern) | Indicates whether the specified regular expression finds a match in the specified input string. |
| public MatchCollection Matches(string input) | Searches the specified input string for all occurrences of a regular expression. |
| public string Replace(string input, string replacement) | In a specified input string, replaces all strings that match a regular expression pattern with a specified replacement string. |
| public string[] Split(string input) | Splits an input string into an array of substrings at the positions defined by a regular expression pattern specified in the Regex constructor. |

[**C#: Exception Handling Cheat Sheet**](https://simplecheatsheet.com/c-exception-handling/)

The syntax for using try/catch looks like:

try {

// statements causing exception

} catch( ExceptionName e1 ) {

// error handling code

} catch( ExceptionName e2 ) {

// error handling code

} catch( ExceptionName eN ) {

// error handling code

} finally {

// statements to be executed

}

**Exception Classes**

|  |  |
| --- | --- |
| System.IO.IOException | Handles I/O errors. |
| System.IndexOutOfRangeException | Handles errors generated when a method refers to an array index out of range. |
| System.ArrayTypeMismatchException | Handles errors generated when the type is mismatched with the array type. |
| System.NullReferenceException | Handles errors generated from referencing a null object. |
| System.DivideByZeroException | Handles errors generated from dividing a dividend with zero. |
| System.InvalidCastException | Handles errors generated during typecasting. |
| System.OutOfMemoryException | Handles errors generated from insufficient free memory. |
| System.StackOverflowException | Handles errors generated from stack overflow. |