## **About Basics**

C# is a statically-typed language, which means that everything has a type at compile-time. Choosing a name for a **[variable](https://www.guru99.com/c-sharp-variables-operator.html" \t "https://exercism.org/tracks/csharp/concepts/_blank)** is referred to as defining a variable. Once a variable is defined, setting or updating a value is called variable assignment. A variable can be defined either by explicitly specifying its type, or by using the **[var keyword](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/var" \t "https://exercism.org/tracks/csharp/concepts/_blank)** to have the C# compiler infer its type based on the assigned value. The use of var is known as *type inference*.

int explicitVar = 10; *// Explicitly typed*

var implicitVar = 10; *// Implicitly typed*

The value of a variable can be assigned and updated using the **[= operator](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/assignment-operator" \t "https://exercism.org/tracks/csharp/concepts/_blank)**. Once defined, a variable's type can never change.

var count = 1; *// Assign initial value*

count = 2; *// Update to new value*

*// Compiler error when assigning different type// count = false;*

C# is an object-oriented language and requires all **[functions](https://csharp.net-tutorials.com/classes/methods/" \t "https://exercism.org/tracks/csharp/concepts/_blank)** to be defined in a *class*, which are defined using the **[class keyword](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/classes" \l "declaring-classes" \t "https://exercism.org/tracks/csharp/concepts/_blank)**. Objects (or *instances*) are created by using the new keyword.

class Calculator

{

*// ...*

}

var calculator = new Calculator();

A function within a class is referred to as a *method*. Each **[method](https://docs.microsoft.com/en-us/dotnet/csharp/methods" \t "https://exercism.org/tracks/csharp/concepts/_blank)** can have zero or more parameters. All parameters must be explicitly typed, there is no type inference for parameters. Similarly, the return type must also be made explicit. Values are returned from methods using the **[return keyword](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/return" \t "https://exercism.org/tracks/csharp/concepts/_blank)**. To allow a method to be called by code in other files, the public access modifier must be added.

class Calculator

{

public int Add(int x, int y)

{

return x + y;

}

}

Methods are invoked using dot (.) syntax on an instance, specifying the method name to call and passing arguments for each of the method's parameters. Arguments can optionally specify the corresponding parameter's name.

var calculator = new Calculator();var sum\_v1 = calculator.Add(1, 2);var sum\_v2 = calculator.Add(x: 1, y: 2);

If the method to be called is defined in the same class as the method that calls it, the class name can be omitted.

If a method does not use any object *state*, the method can be made *static* using the static modifier. Static methods are also invoked using dot (.) syntax, but are invoked on the class itself instead of an instance of the class.

If a class only has static methods, it too can be made static using the static modifier. This expresses the intention of the class.

static class Calculator

{

public static int Multiply(int x, int y)

{

return x \* y;

}

}

Calculator.Multiply(2, 4); *// => 8*

Scope in C# is defined between the { and } characters.

C# supports two types of **[comments](https://www.w3schools.com/cs/cs_comments.asp" \t "https://exercism.org/tracks/csharp/concepts/_blank)**. Single line comments are preceded by // and multiline comments are inserted between /\* and \*/.

Integer values are defined as one or more (consecutive) digits and support the **[default mathematical operators](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/arithmetic-operators" \l "addition-operator-" \t "https://exercism.org/tracks/csharp/concepts/_blank)**.

A variable name **[must follow some rules](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/inside-a-program/identifier-names" \t "https://exercism.org/tracks/csharp/concepts/_blank)** like starting either by a letter or an underscore, and **[should follow C# naming convention](https://docs.microsoft.com/en-us/dotnet/standard/design-guidelines/naming-guidelines" \t "https://exercism.org/tracks/csharp/concepts/_blank)**. If a variable name collides with a reserved **[C# keyword](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/" \t "https://exercism.org/tracks/csharp/concepts/_blank)**, it must be escaped using @. Use of that notation is not recommended, but it can be encountered in exceptional cases, for example: var @this, var @class or var @var.

There are a couple of concepts that are so fundamental to a C-family language like C# that they occur naturally in almost any piece of non-trivial code. These are **[mutation](https://benmccormick.org/2016/06/04/what-are-mutable-and-immutable-data-structures-2" \t "https://exercism.org/tracks/csharp/concepts/_blank)** and **[scope](https://www.geeksforgeeks.org/scope-of-variables-in-c-sharp/" \t "https://exercism.org/tracks/csharp/concepts/_blank)**. Mutation is the idea that a variable can have its value changed in the course of a program's lifetime. Scope is the idea that the value associated with a name (of a program element) is only accessible within the code "area" where it is defined. The principal code areas in C# are *class/struct* and *function* (typically methods) but can also include the bodies of loops and other constructs.