Introduction

This lesson introduces methods and explains how to declare them using example

WE'LL COVER THE FOLLOWING

What is a method?

Functions declared inside a *struct* or *class* are called *methods*.

A *method* encapsulates a *block of code* that can be called from other parts of the program.

The use of *methods* in **C#** increases **code reusability**. Literally, you can tag a particular section of code with a name (*method* name) and just call that name from anywhere of your program.

> **Note:** In **C#** a *method* may or may not have a *return* type.

Declaring a Method

Every method has a unique signature consisting of:

- **accessor** (`public`, `private`, ...)
- **optional modifier** (`abstract`)
- a **name**
- **method parameters** (if needed)

> **Note:** The **return** type is not part of the signature.

A *method* prototype looks like the following:

```
AccessModifier OptionalModifier ReturnType MethodName(InputParameters)
{
 //Method body
}
```

Method Syntax

- **AccessModifier** can be `public`, `protected`, `private` or by default *internal*.
- **OptionalModifier** can be `static` `abstract` `virtual` `override` `new` or `sealed`.
- **ReturnType** can be `void` for no **return** or can be any type from the basic ones, as `int` to complex **classes**.
- A *method* may have some or no input *parameters*.
- To set *parameters* for a method you should:
  - *declare* each one like normal variable declarations (like `int a`)
  - for *more* than *one* parameter you should use a **comma** between them (like `int a`, `int b`)

> **Note:** Parameters may have *default* values. You can set a value for the *parameter* (like `int a = 0`). If a parameter has a default value, setting the input value is optional.

Example

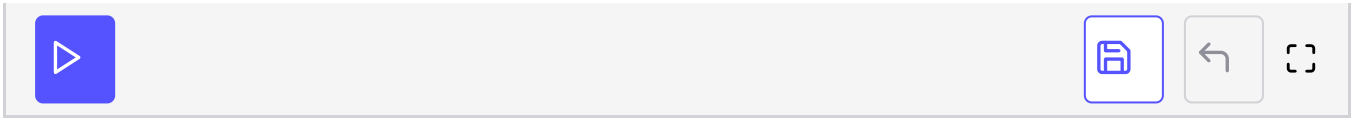
The following method example **returns** the *sum* of *two integers*:

```
using System;

class MethodExample
{
    public static int Sum(int a, int b) //method initialization
```

```
    public static int Sum(int a, int b) //method initialization
    {
        return a + b; //returning sum of a and b
    }

    static void Main()
    {
      int answer;
      answer = Sum(5,6); //calling method by passing appropriate arguments
      Console.WriteLine("answer is: {0}", answer);
    }
}
```



Access rights

We discussed **access modifiers** above and how they are used while declaring a method.

Let's learn the use of these **access modifiers** below.

```
// static: is callable on a class even when no instance of the class has been created
public static void MyMethod()

// virtual: can be called or overridden in an inherited class
public virtual void MyMethod()

// internal: access is limited within the current assembly
internal void MyMethod()

//private: access is limited only within the same class
private void MyMethod()

//public: access right from every class / assembly
public void MyMethod()

//protected: access is limited to the containing class or types derived from it
protected void MyMethod()

//protected internal: access is limited to the current assembly or types derived from the cor
protected internal void MyMethod()
```

Access Modifiers

This is all on the basic introduction of *methods* in **C#**. In the next lesson we will delve into more details of *methods*.