C# Methods / Functions with Examples

In c#, the **Method** is a separate code block, and that contains a series of statements to perform particular operations. Methods must be declared either in class (/tutorial/csharp/csharp-classes-and-objects-with-examples) or struct (/tutorial/csharp/csharp-structures-structs) by specifying the required parameters.

Generally, methods are useful to improve code reusability by reducing code duplication. If we have the same functionality to perform in multiple places, then we can create one method with the required functionality and use it wherever it is required in the application.

Syntax of C# Methods

As discussed, c# **Methods** must be declared either in a class (/tutorial/csharp/csharp-classes-and-objects-with-examples) or struct (/tutorial/csharp/csharp-structures-structs) by specifying the required access level, return type, name of the method, and any method parameters as shown below.

```
class class_name
{
    ...
    ...
    <Access_Specifier> <Return_Type> Method_Name()
    {
        // Statements to Execute
    }
    ...
    ...
}
```

If you observe the above syntax, we defined the method in a **class** with various parameters, which are

- **Access_Specifier** - It is used to define an access level, either **public** or **private**, etc., to allow other classes to access the method. If we didn't mention any access modifier, then by default, it is **private**.
- **Return_Type** - It is used to specify the type of value the method can return. If the method is not returning any value, then we need to mention **void** as the return type.
- **Method_Name** - It must be a unique name to identify the method in a class.
- **Parameters** - The method parameters are useful to send or receive data from a method, and these method parameters are enclosed within parentheses and are separated by commas. If no parameters are required for a method, we need to define a method with empty parentheses.

In c#, both **methods** and **functions** are the same, there is no difference, and these are just different terms to do the same thing in c#.

Following are examples of defining the different types of methods in the c# programming language.

```
class Users
{
    public void GetUsers() {
       // Statements to Execute
    }
    private void InsertUserDetails(string name, int age) {
       // Statements to Execute
    }
    protected string GetUserDetails(int userid)
    {
       // Statements to Execute
    }
}
```

If you observe the above example, we defined different methods with different access modifiers, return types, and different parameters based on our requirements.

Now we will see the complete example using the methods in the c# programming language.

C# Methods Example

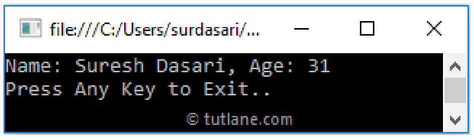
Following is an example of using methods in the c# programming language.

```
using System;

namespace Tutlane
{
     class Program
     {
         static void Main(string[] args)
         {
             Program p = new Program();
             string result = p.GetUserDetails("Suresh Dasari", 31);
             Console.WriteLine(result);
             Console.WriteLine("Press Enter Key to Exit..");
             Console.ReadLine();
         }
         public string GetUserDetails(string name, int age)
         {
             string info = string.Format("Name: {0}, Age: {1}", name, age);
             return info;
         }
     }
}
```

If you observe the above example, we created a **GetUserDetails** method and passed parameters to perform the required operations. We are accessing the **GetUserDetails** method by creating an instance of the **Program** class in the **Main** method to show the result.

When we execute the above c# program, we will get the result below.



```
Name: Suresh Dasari, Age: 31
Press Any Key to Exit..
```

This is how we can use methods in c# applications based on our requirements.

C# Static Methods

In c#, if we create methods with static (/tutorial/csharp/csharp-static-keyword), then we can directly invoke those methods from the class level without creating an object.

If you observe the above example, the **Main** method is static (/tutorial/csharp/csharp-static-keyword), and we are invoking the **GetUserDetails** method by creating an instance of the **Program** class. Instead, if we create a static method, we can directly access those methods in the **Main()** method without creating an instance of an object.

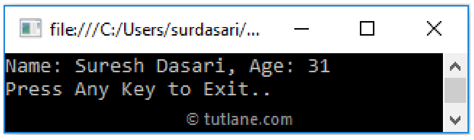
Following is the example of creating static (/tutorial/csharp/csharp-static-keyword) methods to access those methods directly in the **Main()** method without creating an instance of an object in c#.

```
using System;

namespace Tutlane
{
    class Program
    {
        static void Main(string[] args)
        {
            string result = GetUserDetails("Suresh Dasari", 31);
            Console.WriteLine(result);
            Console.WriteLine("Press Enter Key to Exit..");
            Console.ReadLine();
        }
        public static string GetUserDetails(string name, int age)
        {
            string info = string.Format("Name: {0}, Age: {1}", name, age);
            return info;
        }
    }
}
```

If you observe the above example, we created a method **GetUserDetails()** with static (/tutorial/csharp/csharp-static-keyword). Hence, we can access that method directly in the **Main()** method without creating an instance of a **class** object.

When we execute the above c# program, we will get the result below.



```
Name: Suresh Dasari, Age: 31
Press Any Key to Exit..
```

This is how we can create static methods in c# applications to access without creating an instance of an object based on our requirements.

C# Methods with No Return Type and Parameters

As per our requirements, we can create methods in c# applications with or without return types and parameters. If we use **void** as a return type for the method, then that method won't return any value.

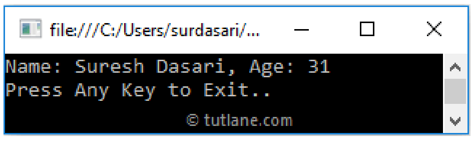
Following is the example of creating a method without having any return value and parameters in the c# programming language.

```
using System;

namespace Tutlane
{
     class Program
     {
         static void Main(string[] args)
         {
             GetUserDetails();
             Console.WriteLine("Press Enter Key to Exit..");
             Console.ReadLine();
         }
         public static void GetUserDetails()
         {
             string name = "Suresh Dasari";
             int age = 31;
             Console.WriteLine("Name: {0}, Age: {1}", name, age);
         }
     }
}
```

If you observe the above example, we created a method called "**GetUserDetails**" without having any return type (**void**) and parameters.

When we execute the above c# program, we will get the result below.



```
Name: Suresh Dasari, Age: 31
Press Any Key to Exit..
```

This is how we can create methods in c# without having any return type and parameters based on our requirements.

C# Passing Parameters to Methods

If we create a method with parameters in c#, then we need to pass parameters to that method while calling it in our application.

In c#, we have different ways to pass parameters to the method; those are

Parameters	**Description**
Value Parameters (/tutorial/csharp/csharp-pass-by-value-with-examples)	These are called "**input parameters**" and these parameters will pass a copy of the original value instead of the original parameters. So the changes made to the parameters in the called method will not affect the original values when control returns to the caller.
Reference Parameters (/tutorial/csharp/csharp-pass-by-reference-ref-with-examples)	These are called "**input/output parameters**" and these will pass a memory reference of original parameters. So the changes made to the parameters in the called method will affect the original values when control returns to the caller.
Output Parameters (/tutorial/csharp/csharp-out-parameter-with-examples)	These are called "**output parameters**" and these are more like reference type parameters, but the only difference is we don't need to initialize them before passing.

In the next chapters, we will learn more about parameter passing to methods or functions in a detailed manner with examples in the c# programming language.