**Methods In C#**

**What Are Methods In C# Programming**

* A method is a group of statements that together perform a task.
* Methods are functions declared in a class and may be used to perform operations on class variables.
* They are blocks of code that can take parameters and may or may not return a value.
* It is used to perform specific task.
* Methods are reusable.
* A method implements the behavior of an object, which can be accessed by instantiating the object of the class in which it is defined and then invoking the method.
* Methods specify the manner in which a particular operation is to be carried out on the required data members of the class.
* Every C# program has at least one class with a method named Main.

**What is Main Method?**

* The Main() method indicates to the CLR that this is the first method of the program which is declared within a class and specifies where the program execution begins.
* Every C# program that is to be executed must have a Main() method as it is the entry point to the program.
* The return type of the Main()in C# can be int or void.

**The following syntax is used to create a method:**

<access\_modifier><return\_type><MethodName> ([list of parameters]){

// body of the method

}

﻿

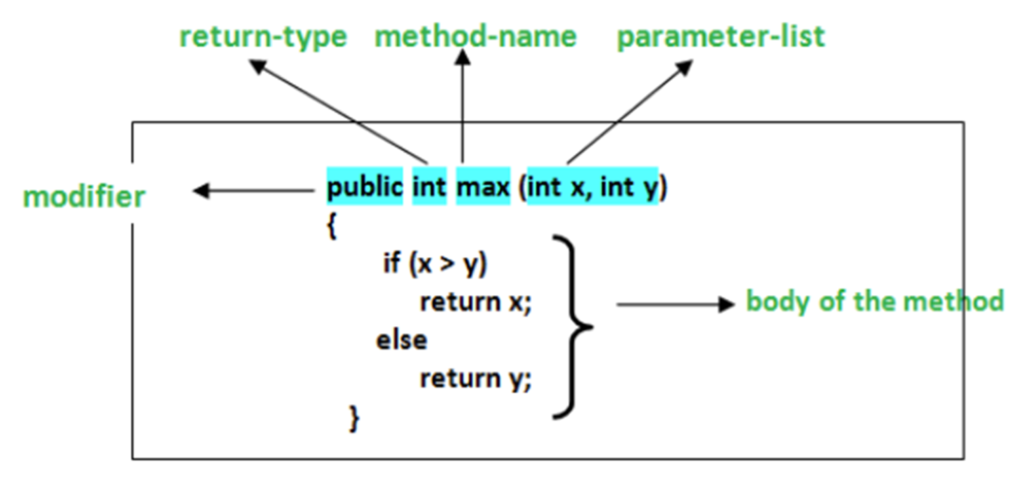
**where,**

* **access\_modifier**: Specifies the scope of access for the method.
* **return\_type**: Specifies the data type of the value that is returned by the method and it is optional.
* **MethodName**: Specifies the name of the method.
* **list of parameters:** Specifies the arguments to be passed to the method.

**Conventions to be followed for naming methods state that a method name:**

* Cannot be a C# keyword, cannot contain spaces, and cannot begin with a digit
* Can begin with a letter, underscore, or the “@” character
* Some examples of valid method names are: Add(), Sum\_Add(), and @Add().
* Invalid method names include 5Add, AddSum(), and int().

**The syntax for defining a method in C# is as follows −**



**The following code shows the definition of a method named Add() that adds two integer numbers:**

using System;

class Sum

{

int Add(int numOne, int numTwo)

{

int addResult = numOne + numTwo;

Console.WriteLine(“Addition = “ + addResult);

. . .

}

}

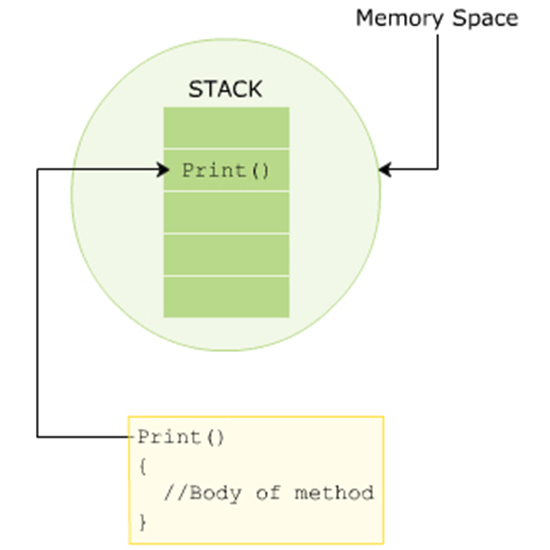
**In Above code,**

* The Add() method takes two parameters of type int and performs addition of those two values.
* Finally, it displays the result of the addition operation.

**Invoking Or Calling Methods**

* A method can be invoked in a class by creating an object of the class where the object name is followed by a period (.) and the name of the method followed by parentheses.
* In C#, a method is always invoked from another method. This is referred to as the calling method and the invoked method is referred to as the called method.

**The following figure displays how a method invocation or call is stored in the stack in memory and how a method body is defined:**



**Code Snippet**

class Book{

string \_bookName;

public string Print() {

return \_bookName;

}

public void Input(string bkName) {

\_bookName = bkName;

}

static void Main(string[] args)

{

Book objBook = new Book();

objBook.Input(“C#-The Complete Reference”);

Console.WriteLine(objBook.Print());

}

}

**In Above Code:**

* The Main() method is the calling method and the Print() and Input()methods are the called methods.
* The Input() method takes in the book name as a parameter and assigns the name of the book to the \_bookName variable.
* Finally, the Print()method is called from the Main()method and it displays the name of the book as the output.

**Output**

C#-The Complete Reference

**Method Parameters and Arguments**

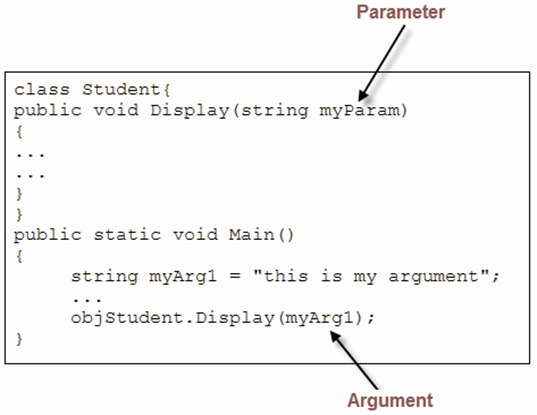
**Parameters**

* The variables included in a method definition are called parameters. Which may have zero or more parameters, enclosed in parentheses and separated by commas. If the method takes no parameters, it is indicated by empty parentheses.

**Arguments**

* When the method is called, the data that you send into the method's parameters are called arguments.

**The following figure shows an example of parameters and arguments:**



**Named and Optional Arguments**

* A method in a C# program can accept multiple arguments that are passed based on the position of the parameters in the method signature.
* A method caller can explicitly name one or more arguments being passed to the method instead of passing the arguments based on their position.
* An argument passed by its name instead of its position is called a named argument.
* While passing named arguments, the order of the arguments declared in the method does not matter.
* Named arguments are beneficial because you do not have to remember the exact order of parameters in the parameter list of methods.

**The following code demonstrates how to use named arguments:**

using System;  
   
class Student  
{  
void printName(String firstName, String lastName)   
{  
Console.WriteLine("First Name = {0}, Last Name = {1}",   
firstName, lastName);  
 }  
static void Main(string[] args)  
{  
 Student student = new Student();  
 /\*Passing argument by position\*/  
student.printName("Henry","Parker");  
 /\*Passing named argument\*/  
student.printName(firstName: "Henry", lastName: "Parker");  
student.printName(lastName: "Parker", firstName:   
"Henry");  
 /\*Passing named argument after positional argument\*/  
student.printName("Henry", lastName: "Parker");  
   
 }  
 }

**In Above Code,**

* The first call to the printNamed() method passes positional arguments.
* The second and third call passes named arguments in different orders.
* The fourth call passes a positional argument followed by a named argument.

**Output**

First Name = Henry, Last Name = Parker

First Name = Henry, Last Name = Parker

First Name = Henry, Last Name = Parker

First Name = Henry, Last Name = Parker

**The following code shows another example of using named arguments:**

using System;

class TestProgram

{

void Count(int boys, int girls)

{

Console.WriteLine(boys + girls);

}

static void Main(string[] args)

{

TestProgramobjTest = new TestProgram();

objTest.Count(boys: 16, girls: 24);

}

}

* C# also supports optional arguments in methods and can be emitted by the method caller.
* Each optional argument has a default value.

**The following code shows how to use optional arguments:**

using System;

class OptionalParameterExample

{

void printMessage(String message="Hello user!") {

Console.WriteLine("{0}", message);

}

static void Main(string[] args)

{

OptionalParameterExampleopExample = new OptionalParameterExample();

opExample.printMessage("Welcome User!");

opExample.printMessage();

}

}

**In Above code,**

* The printMessage() method declares an optional argument message with a default value Hello user!.
* The first call to the printMessage() method passes an argument value that is printed on the console.
* The second call does not pass any value and therefore, the default value gets printed on the console.

**Output**

Welcome User!  
Hello user!