**Tutorial C# Basics 6**

**Methods**

V1.0

24/08/2022

In the previous tutorials, we have covered all three logic structures in C#. From this tutorial, you will study features of programming language and coding patterns. In this tutorial, you will study a feature called “method”. Let’s get started.

**Topics**

[**1.** **“Method” (or function) in programming** 1](#_Toc110535313)

[**2.** **Inputs / outputs of a method** 2](#_Toc110535314)

[**3.** **C# method** 2](#_Toc110535315)

[**4.** **Method parameters and return values** 3](#_Toc110535316)

[**Class activity 1** 3](#_Toc110535317)

[**Class activity 2** 4](#_Toc110535318)

[**Class activity 3** 5](#_Toc110535319)

[**Step 1.** **Create a new project** 5](#_Toc110535320)

[**Step 2.** **Implement the “RockPaperScissors” method** 5](#_Toc110535321)

[**Step 3.** **Implement the main program** 5](#_Toc110535322)

[**Step 4.** **Create a CLI based rock-paper-scissors game** 5](#_Toc110535323)

[**5.** **Create reusable method** 6](#_Toc110535324)

[**Class activity 4** 6](#_Toc110535325)

[**6.** **Using the “ref” modifier in a method** 7](#_Toc110535326)

[**Model answers** 8](#_Toc110535327)

[**Class activity 1** 9](#_Toc110535328)

[**Class activity 2** 9](#_Toc110535329)

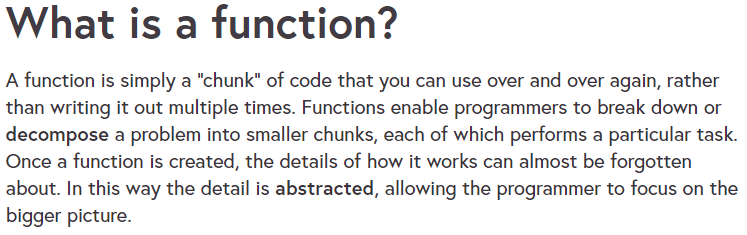
[**Class activity 3** 10](#_Toc110535330)

1. **“Method” (or function) in programming**

Spend a few minutes and watch the following video. Note that the term “method” and “function” are interchangeable in programming.

<https://view.vzaar.com/18710123/video>

So, what is method (or function) in programming?



1. **Inputs / outputs of a method**

A method can be represented with the following chart

* Inputs – data to be processed, a method could have 0~n inputs
* Method body – the logic to process the input data
* Outputs – data included in the result; a method could have 0~n outputs

Method Body

Inputs (0~n)

Outputs (0~n)

For example, there is a method to calculate the area of a rectangle. Then, for this method

* **Length** and **width** are the inputs
* **Area = length \* width** is the body
* **Area** is the output

1. **C# method**

Use the [C# Methods](https://www.w3schools.com/cs/cs_methods.php) tutorial on w3schools website for study. In this tutorial, you will study

* How to define a method (with 0 input and 0 outputs)
* How to call a method in the Main program

For now, please keep in mind, when declaring a method



* The key word “**static**” is mandatory in the declaration. We will talk about the “static” key word in the “Object Oriented Programming” topics
* If there is 0 input, then the round brackets “()” after the method name is empty.
* If there is 0 output, then the key word “**void**” must be used.

Now you understand that the “Main” entry of the program is a also method

A picture containing logo

Description automatically generated

* Main is a special method because it is the entry point of the program
* Your program must have one and only one Main method
* If you have more than one Main methods in a project, Visual Studio will report an error and will not be able to launch the program

1. **Method parameters and return values**

Most methods require input & output. Study the [C# Method Parameters](https://www.w3schools.com/cs/cs_method_parameters.php) tutorial on w3schools for how to

* Declare parameters (the inputs) for a method
* Declare the type of return value (the output) for a method
* How to return data in the method implementation
* How to call a method and pass arguments

Please note that a method in C# could only have one return value, meaning one output. If more than one outputs are required, you need to use one of the following techniques

* Pass arguments as reference and modify the arguments (will be discussed later in this tutorial)
* Define a “complex” data type for return value (will be discussed in the future in the OOP topics)

Also note that

* Variables that are declared inside a method have method level scope. These are not accessible outside the method.
* These variables don’t exist after method’s execution is over.

**Class activity 1**

Create the following 4 methods

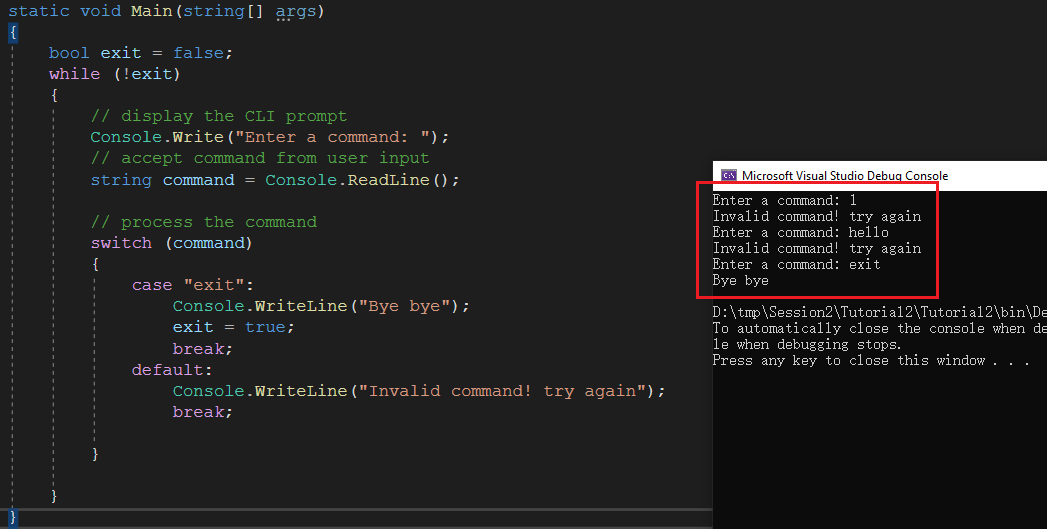
* Addition – calculates a + b and return the result
* Subtraction – calculates a - b and return the result
* Multiplication – calculates a \* b and return the result
* Division– calculates a / b and return the result

Then, test each method in the Main method

|  |
| --- |
| Your codes |
| Test results |

**Class activity 2**

In last tutorial, you have created a Command Line Interface application. below is the code of the mini CLI,



Based on the 4 methods created in the activity 1, create a CLI based calculator that support the following commands

* Addition – calculates a + b and return the result
* Subtraction – calculates a - b and return the result
* Multiplication – calculates a \* b and return the result
* Division– calculates a / b and return the result

|  |
| --- |
| Your codes |
| Test results |

**Class activity 3**

Follow the steps below and finish the rock-paper-scissors game

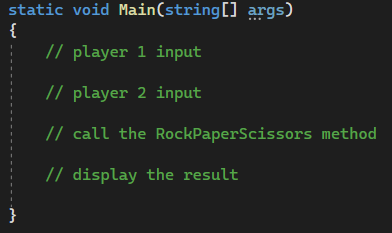
* 1. **Create a new project**
  2. **Implement the “RockPaperScissors” method**

The main structure has been created for you. You are required to complete the method.

|  |
| --- |
| // Method  // RockPaperScissors  // Parameters  // player1: int. 1-Rock, 2-Paper, 3-Scissors, anything else is invalid input  // player1: int. 1-Rock, 2-Paper, 3-Scissors, anything else is invalid input  // Return value  // int: 0 - a tie game  // 1 - if player 1 wins  // 2 - if player 2 wins  // -1 - invalid input  static int RockPaperScissors(int player1, int player2)  {  if(player1 < 1 || player1 > 3 || player2 < 1 || player2 > 3)  {  // invalid input  return -1;  }  int result;  **// your codes**  return result;  } |

* 1. **Implement the main program**

Complete the main program using the following logic



* 1. **Create a CLI based rock-paper-scissors game**

The CLI based game support the following commands

* Play – start a new game. The game is scored
* Show – display the current score for both players
* Reset – reset the scores to 0
* Help – display the supported commands
* Exit – to quit the game

You are required to create some additional methods for the “show” and “help” commands. And think about if it is possible to create a method for “reset” (if not, what is the problem)?

1. **Create a reusable method**

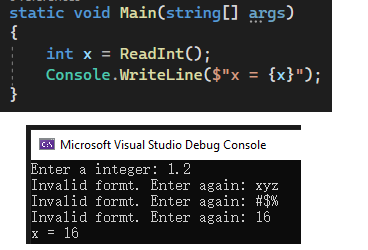
**Class activity 4**

Method is a reusable procedure. Programmer can reuse a method without knowing how it is implemented. To reuse an existing method in a program, all you need to know is what it can do and how call it.

The following method will take user input from console interface and convert the input to a “int” value. If the input is not an integer format, user must input again until a valid integer format is entered.

|  |
| --- |
| // Method: ReadInt  // Parameters: no parameter  // Return value: an integer converted from user input  static int ReadInt()  {  while (true)  {  try  {  int input = Convert.ToInt32(Console.ReadLine());  return input;  }  catch  {  Console.Write("Invalid format. Enter an integer: ");  }  }  } |

You have not studied the “try … catch” feature in C# yet, and do not understand how it works, but it is ok. You can still study the input/output structure of the method, then you will be able to use it. Here is an example.



Now, incorporate the “ReadInt” method into the calculator program developed above.

|  |
| --- |
| Your code: |
| Test result: |

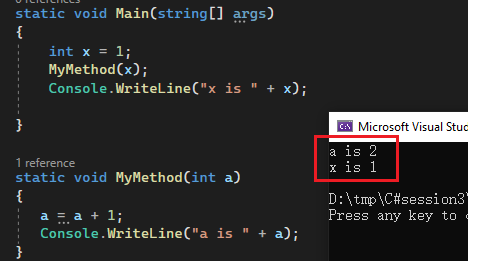
The ReadDouble method below might be useful as well. Save the code and reuse it in another project.

|  |
| --- |
| // Parameters: no parameter  // Return value: a double float number converted from user input  static double ReadDouble()  {  while (true)  {  try  {  double input = Convert.ToDouble(Console.ReadLine());  return input;  }  catch  {  Console.Write("Invalid format. Enter a number: ");  }  }  } |

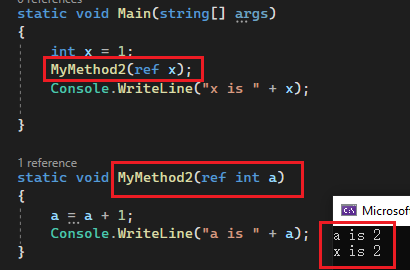
1. **Using the “ref” modifier in a method**

In all the methods created before, we can only return one result at the end of a method as the output. Sometimes we would like a method to have multiple outputs. And one of the techniques is to use parameter to return something.

By default, when calling a method and pass argument to the parameters, only values are passed in instead of the variable itself. So, if we try to modify the value of parameters, the variables on the called side will not be changed. In the example below, the value of variable x remains 1. Because when MyMethod is called, a new variable **a** is created and value of **x** is assigned to **a**. Because **x** and **a** refer to different memory, modifying **a** will not impact the value of **x**.



Be default, arguments are passed by “value” as above. However, with the “**ref**” modifier, we could pass arguments by “reference”, so that a parameter is now a reference (of the same memory block) to the argument from the called. Hence modifying “ref” parameter will modify the value of the argument on the called side. In the example below, **x** is 2 after called the MyMethod2 method.



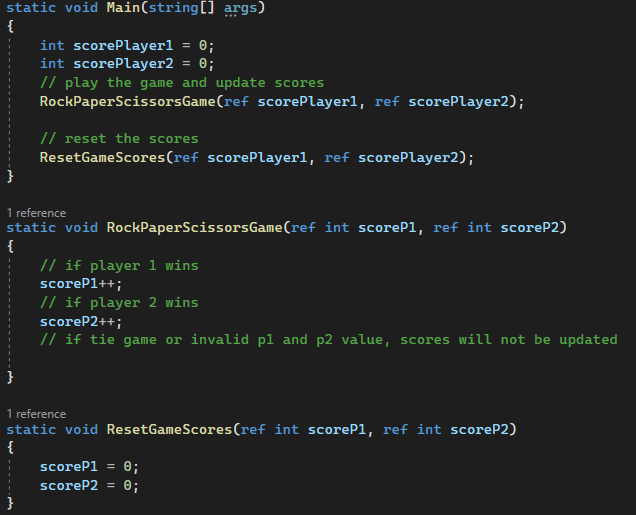
Please note that the key word “ref” must be added when defining a reference parameter in the method as well as in the caller when passing argument.

Let’s have a look at another example:

The methods below could

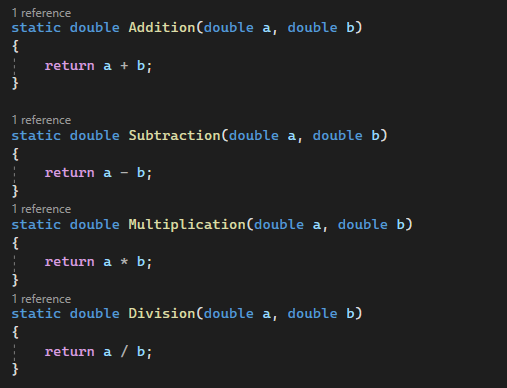
* RockPaperScissorsGame - modify the score for players in a rock-paper-scissors game
* ResetGameScores - reset the scores to 0

Note game logic is not implemented, and you could try to apply the following code to the game method you developed above.



**Model answers**

**Class activity 1**



Text

Description automatically generated

**Class activity 2**

|  |
| --- |
| static void Main(string[] args)  {  bool exit = false;  while (!exit)  {  // display the menu  Console.WriteLine("Select a function:");  Console.WriteLine(" 1. Addition: a + b");  Console.WriteLine(" 2. Subtraction: a - b");  Console.WriteLine(" 3. Multiplication: a \* b");  Console.WriteLine(" 4. Division: a / b");  Console.WriteLine(" 5. Exit the program");  Console.Write("> ");  string command = Console.ReadLine();  double a, b, result;  switch (command)  {  case "1": // Addition  Console.WriteLine("Enter a: ");  a = Convert.ToDouble(Console.ReadLine());  Console.WriteLine("Enter b: ");  b = Convert.ToDouble(Console.ReadLine());  result = Addition(a, b);  Console.WriteLine("Result is: " + result);  break;  case "2": // Subtraction  Console.WriteLine("Enter a: ");  a = Convert.ToDouble(Console.ReadLine());  Console.WriteLine("Enter b: ");  b = Convert.ToDouble(Console.ReadLine());  result = Subtraction(a, b);  Console.WriteLine("Result is: " + result);  break;  case "3": // Multiplication  Console.WriteLine("Enter a: ");  a = Convert.ToDouble(Console.ReadLine());  Console.WriteLine("Enter b: ");  b = Convert.ToDouble(Console.ReadLine());  result = Multiplication(a, b);  Console.WriteLine("Result is: " + result);  break;  case "4": // Division  Console.WriteLine("Enter a: ");  a = Convert.ToDouble(Console.ReadLine());  Console.WriteLine("Enter b: ");  b = Convert.ToDouble(Console.ReadLine());  result = Division(a, b);  Console.WriteLine("Result is: " + result);  break;  case "5": // Exit the program  exit = true;  break;  default:  Console.WriteLine("Invalid command, try again!");  break;  } // switch (command)  } // while (!exit)  Console.WriteLine("Byebye");  } // static void Main(string[] args)  static double Addition(double a, double b)  {  return a + b;  }  static double Subtraction(double a, double b)  {  return a - b;  }  static double Multiplication(double a, double b)  {  return a \* b;  }  static double Division(double a, double b)  {  return a / b;  } |

**Class activity 3**

|  |
| --- |
| static void Main(string[] args)  {  bool exit = false;  int scoreP1 = 0;  int scoreP2 = 0;  while (!exit)  {  // display the menu  Console.WriteLine("Enter a command:");  Console.Write("> ");  string command = Console.ReadLine();  switch (command)  {  case "play":  // user inputs  Console.WriteLine("1-Rock, 2-Paper, 3-Scissors");  Console.Write("Player1: ");  int p1 = Convert.ToInt32(Console.ReadLine());  Console.Write("Player2: ");  int p2 = Convert.ToInt32(Console.ReadLine());  // call the game function  int result = RockPaperScissors(p1, p2);  // update the scores  if (result == 1)  {  Console.WriteLine("Player1 wins");  scoreP1++;  }else if (result == 2)  {  Console.WriteLine("Player2 wins");  scoreP2++;  } else if (result == 0)  {  Console.WriteLine("Tie game");  }  else  {  Console.WriteLine("invalid inputs");  }  break;  case "show":  DisplayScores(scoreP1, scoreP2);  break;  case "reset":  scoreP1 = 0;  scoreP2 = 0;  break;  case "help":  Help();  break;  case "exit": // Exit the program  exit = true;  break;  default:  Console.WriteLine("Invalid command, try again!");  break;  } // switch (command)  } // while (!exit)  Console.WriteLine("Byebye");  } // static void Main(string[] args)  static void DisplayScores(int scoreP1, int scoreP2)  {  Console.WriteLine($"Player1 {scoreP1} : {scoreP2} Player2");  }  // Method  // Help  static void Help()  {  Console.WriteLine("Type a command");  Console.WriteLine(" play: to play a game");  Console.WriteLine(" show: to display the result");  Console.WriteLine(" reset: to reset the scores");  Console.WriteLine(" help: to view all commands");  Console.WriteLine(" exit: to quit the program");  }  // Method  // RockPaperScissors  // Parameters  // player1: int. 1-Rock, 2-Paper, 3-Scissors, anything else is invalid input  // player1: int. 1-Rock, 2-Paper, 3-Scissors, anything else is invalid input  // Return value  // int: 0 - a tie game  // 1 - if player 1 wins  // 2 - if player 2 wins  // -1 - invalid input  static int RockPaperScissors(int player1, int player2)  {  if (player1 < 1 || player1 > 3 || player2 < 1 || player2 > 3)  {  // invalid input  return -1;  }  int result;  if (player1 == 1 && player2 == 1)  {  result = 0;  }  else if (player1 == 1 && player2 == 2)  {  result = 2;  }  else if (player1 == 1 && player2 == 3)  {  result = 1;  }  else if (player1 == 2 && player2 == 1)  {  result = 1;  }  else if (player1 == 2 && player2 == 2)  {  result = 0;  }  else if (player1 == 2 && player2 == 3)  {  result = 2;  }  else if (player1 == 3 && player2 == 1)  {  result = 2;  }  else if (player1 == 3 && player2 == 2)  {  result = 1;  }  else // (player1 == 3 && player2 == 3)  {  result = 0;  }  return result;  } |

**Class activity 4**

|  |
| --- |
| static void Main(string[] args)  {  Console.WriteLine("Enter x:");  int x = ReadInt();  Console.WriteLine($"x is {x}");  } // static void Main(string[] args)  // Method: ReadInt  // Parameters: no parameter  // Return value: an integer converted from user input  static int ReadInt()  {  while (true)  {  try  {  int input = Convert.ToInt32(Console.ReadLine());  return input;  }  catch  {  Console.Write("Invalid format. Enter an integer: ");  }  }  } |

End of this tutorial