

Methods in C#

- A **method** is a group of statements that together perform a task.
- You may define your own methods in C#
- Using method can have many advantages, including the following:
 - You can reuse the code within a method.
 - You can easily test individual methods.
 - If it's necessary to make any code modifications, you can make modifications within a single method, without altering the program structure.
 - You can use the same method for different inputs.
- Every C# program has at least one class with a method named **Main**.
- To use a method, you need to:
 - Define the method
 - Call the method

Defining Methods in C#

- When you define a method, you basically declare the elements of its structure.
- The syntax for defining a method in C# is as follows:

```
<Access Modifier> <Return Type> <Method Name> (Parameter List)
```

```
{  
    Method Body  
}
```

- Following are the various elements of a method:
 - **Access Modifier:** This determines the visibility of a variable or a method from another class.

- **Return Type:** A method may return a value. The return type is the data type of the value the method returns. If the method is not returning any values, then the return type is void.
- **Method Name:** Method name is a unique identifier and it is case sensitive. It cannot be same as any other identifier declared in the class.
- **Parameter List:** Enclosed between parentheses, the parameters are used to pass and receive data from a method. Parameters are optional.
- **Method Body:** This contains the set of instructions needed to complete the required activity.

Examples of Methods

- Method without parameters and without return type:

```
public void Add()  
{  
    int a = 10;  
    int b = 20;  
    int sum = a + b;  
  
    Console.WriteLine("Sum is " + sum);  
}
```

- Method with parameters and without return type:

```
public void Add(int x, int y)  
{  
    int sum = x + y;  
  
    Console.WriteLine("Sum is " + sum);  
}
```

```
}
```

- Method with parameters and with return type:

```
public int Add(int x, int y)
```

```
{
```

```
    int sum = x + y;
```

```
    return sum;
```

```
}
```

Method Parameters

- When method with parameters is called, you need to pass the parameters to the method.
- There are different ways that parameters can be passed to a method:
- **Value Parameters:**
 - This method copies the actual value of an argument into the formal parameter of the method.
 - In this case, changes made to the parameter inside the method have no effect on the argument.
- **Reference Parameters:**
 - This method copies the reference to the memory location of an argument into the formal parameter.
 - This means that changes made to the parameter affect the argument.

- **Output Parameters:**
 - This method helps in returning more than one value.
- **Parameter Array:**
 - You can pass n number of parameters to a method.

Example of Passing By Value and Reference

```
public static void Main()
{
    int i = 0;
    string s = "hello";
    int[] a = { 0, 1, 2, 4, 8 };
    Console.WriteLine("Before Changes:");
    Console.WriteLine("i = " + i);
    Console.WriteLine("s = " + s);
    Console.WriteLine("a[0] = " + a[0]);
    ChangeValues(i, s, a);
    Console.WriteLine("\nAfter Changes:");
    Console.WriteLine("i = " + i);
    Console.WriteLine("s = " + s);
    Console.WriteLine("a[0] = " + a[0]);
}

static void ChangeValues(int num, string str,
int[] array)
```

```
{  
    num = 100;    str = "bye";    a[0] = 100;  
}
```

Output:

Before Changes:

i = 0

s = hello

a[0] = 0

After Changes:

i = 0

s = hello

a[0] = 100

Passing Parameters by Value

- In this mechanism, when a method is called, a new storage location is created for each value parameter.
- The values of the actual parameters are copied into them.
- Hence, the changes made to the parameter inside the method have no effect on the argument.
- Example on the next slide demonstrates the concept.

```
static void Main(string[] args)  
{  
    int a = 10, b = 20;  
    Console.WriteLine("Before swapping:");  
    Console.WriteLine("a: " + a);  
    Console.WriteLine("b: " + b);  
}
```

```
Swap(a, b);

Console.WriteLine("\nAfter swapping:");
Console.WriteLine("a: " + a);
Console.WriteLine("b: " + b);
}

static void Swap(int x, int y)
{
    int temp = x; // save the value of x
    x = y;        // put y into x
    y = temp;     // put temp into y
}
```

Output:

Before swapping:

a: 10

b: 20

After Swapping:

a: 10

b: 20

Passing Parameters by Reference

- A reference parameter is a reference to a memory location of a variable.

- When you pass parameters by reference, unlike value parameters, a new storage location is not created for these parameters.
- The reference parameters represent the same memory location as the actual parameters that are supplied to the method.
- You can pass value type parameters as reference parameters using the ref keyword.
- Example on the next slide demonstrates the concept.

Passing Parameters by Reference

```
static void Main(string[] args)
{
    int a = 10, b = 20;
    Console.WriteLine("Before swapping:");
    Console.WriteLine("a: " + a);
    Console.WriteLine("b: " + b);

    Swap(ref a, ref b);

    Console.WriteLine("\nAfter swapping:");
    Console.WriteLine("a: " + a);
    Console.WriteLine("b: " + b);
}

static void Swap(ref int x, ref int y)
{
```

```
int temp = x; // save the value of x
x = y;       // put y into x
y = temp;    // put temp into y
}
```

Output:

Before swapping:

a: 10

b: 20

After Swapping:

a: 20

b: 10

Passing Parameters by Reference

- When the code on the previous slide is compiled and executed, it produces the following result:
- This shows that the values have changed inside the Swap method and this change reflects in the Main method.
- You also need to add the ref keyword when you invoke the method Swap(ref a, ref b);
- Always remember, any variable must be initialized before it is passed into a method, whether it is passed in by value or by reference.

Output Parameters

- A return statement is used for returning only one value from a method.

- Sometimes, there is a need for methods to be able to output more than one value.
- Again, this can be accomplished by using the output parameters.

```
static void Main(string[] args)
{
    int a = 10, b = 20, sum, mul; // sum and mul
    aren't initialized

    Calculate(a, b, out sum, out mul);
    Console.WriteLine("Sum = " + sum);
    Console.WriteLine("Mul = " + mul);
}

static void Calculate(int a, int b, out int
sum, out int mul)
{
    sum = a + b;
    mul = a * b;
}
```

Output:

Sum = 30

Mul = 200

Exercise

- Write a method **TestString()** that finds out if a string has period and/or comma.

- The `TestString()` method accepts a string and two bool output parameters: **hasPeriod** and **hasComma**.
- The output parameters are set true if a period and/or comma is found in the string, false otherwise.
- You can check for a character in a string by using the `Contains()` method on a string, like this:

```
if (str.Contains(',')) { ... }
```

- Incorporate the **TestString()** method into an app that receives a string from the user and provide the appropriate outputs.

```
num = 100;          str = "bye";          a[0] = 100;
```

Exercise - Solution

```
static void TestString(string str, out bool  
hasPeriod, out bool hasComma)
```

```
{  
    hasPeriod = hasComma = false;  
    if (str.Contains('.'))  
        hasPeriod = true;  
    if (str.Contains(','))  
        hasComma = true;  
}
```

```
static void Main(string[] args)
```

```
{  
    Console.Write("Enter a string that might  
contain a period and/or comma: ");
```

```
string s = Console.ReadLine();

TestString(s, out bool hasPeriod, out bool
hasComma);

if (hasPeriod || hasComma)

    Console.WriteLine("The string contains a
period and/or comma");
else

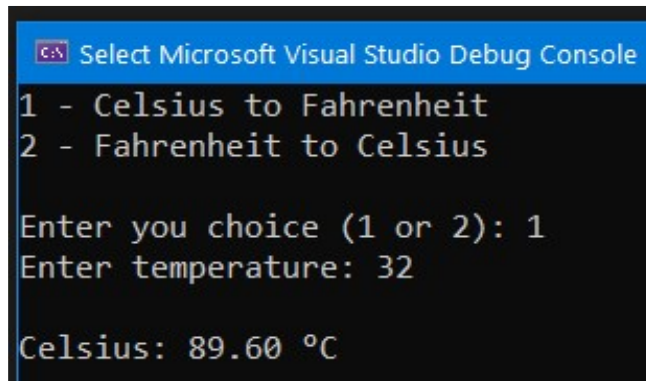
    Console.WriteLine("The string doesn't
contain a period and/or comma");
}
```

Do It Yourself!

- Exercise: Temperature Conversions:
- Implement the following methods:
 - a) Method Fahrenheit returns the Fahrenheit equivalent of a Celsius temperature, using the calculation:
$$f = 9.0 / 5.0 * c + 32;$$
 - b) Method Celsius returns the Celsius equivalent of a Fahrenheit temperature, using the calculation”
$$c = 5.0 / 9.0 * (f - 32);$$
- Use the methods from parts (a) and (b) to write an app that enables the user either to enter a Celsius temperature and display the

Fahrenheit equivalent or enter a Fahrenheit temperature and display the Celsius equivalent.

- Display the unit (°F or °C)



```
C# Select Microsoft Visual Studio Debug Console
1 - Celsius to Fahrenheit
2 - Fahrenheit to Celsius

Enter you choice (1 or 2): 1
Enter temperature: 32

Celsius: 89.60 °C
```

HW on methods:

Write C# programs for each of the following:

- 1- Write a C# method called `Power()` that receives two integer numbers X and Y and returns X^Y . **Example:** `Power(2,3)`, the result will be 8.
- 2- Write a C# method called `Factorial()` that calculates the factorial of any integer number.
Factorial of any integer number $X = 1*2*3*...X$: **Example:** `Factorial(4)`, the result will be 24.
- 3- Write a C# method to calculate the **area** and the **perimeter** of any rectangle. The method receives the Length L and the Width W of the rectangle and returns the area and the perimeter.
- 4- Write a C# method to calculate the **volume** of any cylinder. The method receives the radius r and the height h of the cylinder and

calculates the volume according to this formula: $\text{Volume} = r^2\pi h$ and returns the result.

- 5- Write a C# method that receives an integer array of size 10 and returns the **maximum** value among the array elements.
- 6- Write a C# method that receives an integer array of size 10 and returns the **average** of the array elements.
- 7- Write a C# method that searches an integer array of size 10 for a given value. If the value is found, the method returns its index, otherwise, it returns -1.
- 8- Write a C# method called `sum_column()` that receives an integer matrix of size 10x10 and calculates the summation of a given column. Example: `sum_column(marks, 3)`, that means calculating the summation of the 3rd column of the matrix marks.
- 9- Write a C# method called `sum_main_diagonal()` that receives an integer matrix of size 10x10 and calculates the summation of the main diagonal elements.
- 10- Write a C# method called `sum_secondary_diagonal()` that receives an integer matrix of size 10x10 and calculates the summation of the secondary diagonal elements.