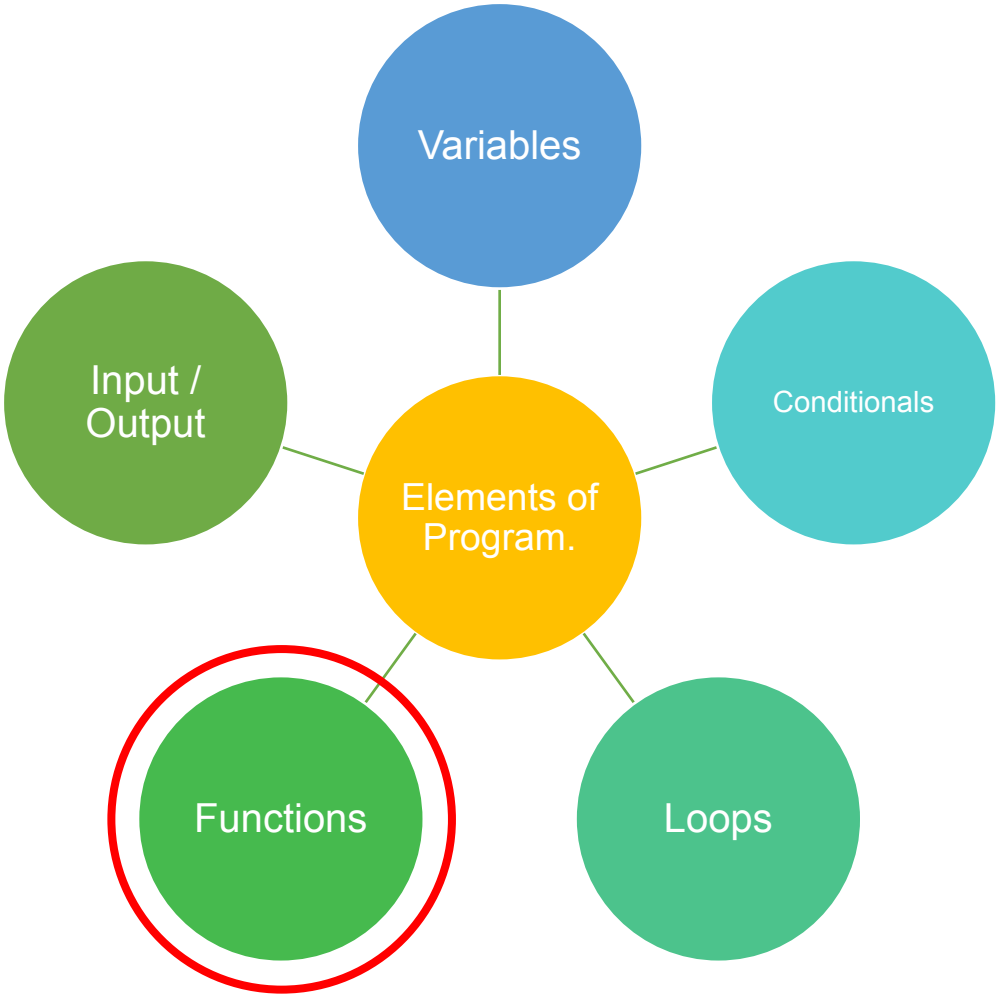
Functions

A4 – Constructs and Techniques and Their Implementation in Programming

Ikromkhujaev Ilyoskhuja

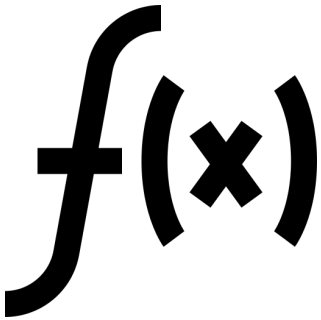
Elements

- To develop any instruction there are some elements needed or we can essentially present in all language.
- So any programming language is made up of 5 basic elements of the instructions.



Functions - Methods

- A **method** is a block of code which only runs when it is called.
- You can pass data, known as parameters, into a method.
- Methods are used to perform certain actions, and they are also known as **functions**.
- Why use methods? To reuse code: define the code once, and use it many times.



Create a Method

A method is defined with the name of the method, followed by parentheses **()**.

C# provides some pre-defined methods, which you already are familiar with, such as `Main()`.

Example

Create a method inside the Program class:

```
class Program
{
  static void MyMethod()
  {
    // code to be executed
  }
}
```

Create a Method

- `MyMethod()` is the name of the method
- `static` means that the method belongs to the Program class and not an object of the Program class.

  You will learn more about objects and how to access methods through objects later in this lesson.
- `void` means that this method does not have a return value.

  You will learn more about return values later in this lesson.

Example

Create a method inside the Program class:

```
class Program
{
  static void MyMethod()
  {
    // code to be executed
  }
}
```

Call a Method

- To call (execute) a method, write the method's name followed by two parentheses **()** and a semicolon;
- In the following example, `MyMethod()` is used to print a text (the action), when it is called:

Example

Inside `Main()`, call the `myMethod()` method:

```
static void MyMethod()
{
  Console.WriteLine("I just got executed!");
}

static void Main(string[] args)
{
  MyMethod();
}

// Outputs "I just got executed!"
```

Call a Method

A method can be called multiple times:

Example

```
static void MyMethod()
{
  Console.WriteLine("I just got executed!");
}

static void Main(string[] args)
{
  MyMethod();
  MyMethod();
  MyMethod();
}

// I just got executed!
// I just got executed!
// I just got executed!
```

Method Parameters

- Information can be passed to methods as parameter. Parameters act as variables inside the method.
- They are specified after the method name, inside the parentheses. You can add as many parameters as you want, just separate them with a comma.
- When a **parameter** is passed to the method, it is called an **argument**.

Method Parameters

- The following example has a method that takes a string called **fname** as parameter. When the method is called, we pass along a first name, which is used inside the method to print the full name:

Example

```
static void MyMethod(string fname)
{
  Console.WriteLine(fname + " Refsnes");
}

static void Main(string[] args)
{
  MyMethod("Liam");
  MyMethod("Jenny");
  MyMethod("Anja");
}

// Liam Refsnes
// Jenny Refsnes
// Anja Refsnes
```

W3 schools

Multiple Parameters

You can have as many parameters as you like, just separate them with commas:

Example

```
static void MyMethod(string fname, int age)
{
  Console.WriteLine(fname + " is " + age);
}

static void Main(string[] args)
{
  MyMethod("Liam", 5);
  MyMethod("Jenny", 8);
  MyMethod("Anja", 31);
}

// Liam is 5
// Jenny is 8
// Anja is 31
```

Default Parameter Value

- You can also use a default parameter value, by using the equals sign (=).
- If we call the method without an argument, it uses the default value ("Norway").

A parameter with a default value, is often known as an "optional parameter". From the example, country is an optional parameter and "Norway" is the default value.

Example

```
static void MyMethod(string country = "Norway")
{
  Console.WriteLine(country);
}

static void Main(string[] args)
{
  MyMethod("Sweden");
  MyMethod("India");
  MyMethod();
  MyMethod("USA");
}

// Sweden
// India
// Norway
// USA
```

Return Values

- In the previous slides, we used the `void` keyword in all examples, which indicates that the method should not return a value.

- If you want the method to return a value, you can use a primitive data type (such as `int` or `double`) instead of `void`, and use the `return` keyword inside the method:

Example

```
static int MyMethod(int x)
{
  return 5 + x;
}

static void Main(string[] args)
{
  Console.WriteLine(MyMethod(3));
}

// Outputs 8 (5 + 3)
```

Return Values

This example returns the sum of a method's **two parameters**:

Example

```
static int MyMethod(int x, int y)
{
  return x + y;
}

static void Main(string[] args)
{
  Console.WriteLine(MyMethod(5, 3));
}

// Outputs 8 (5 + 3)
```

Return Values

You can also store the result in a variable (recommended, as it is easier to read and maintain):

Example

```
static int MyMethod(int x, int y)
{
  return x + y;
}

static void Main(string[] args)
{
  int z = MyMethod(5, 3);
  Console.WriteLine(z);
}

// Outputs 8 (5 + 3)
```

Named Arguments

It is also possible to send arguments with the `key: value` syntax.

That way, the order of the arguments does not matter:

Example

```
static void MyMethod(string child1, string child2, string child3)
{
  Console.WriteLine("The youngest child is: " + child3);
}

static void Main(string[] args)
{
  MyMethod(child3: "John", child1: "Liam", child2: "Liam");
}

// The youngest child is: John
```

Next lecture

- In the next lecture we continue focusing on «Constructs and techniques and their implementation»
- “Microsoft Visual C# Step by Step” Microsoft Press