MODULE 1: INTRODUCTION TO PROGRAMMING IN C#

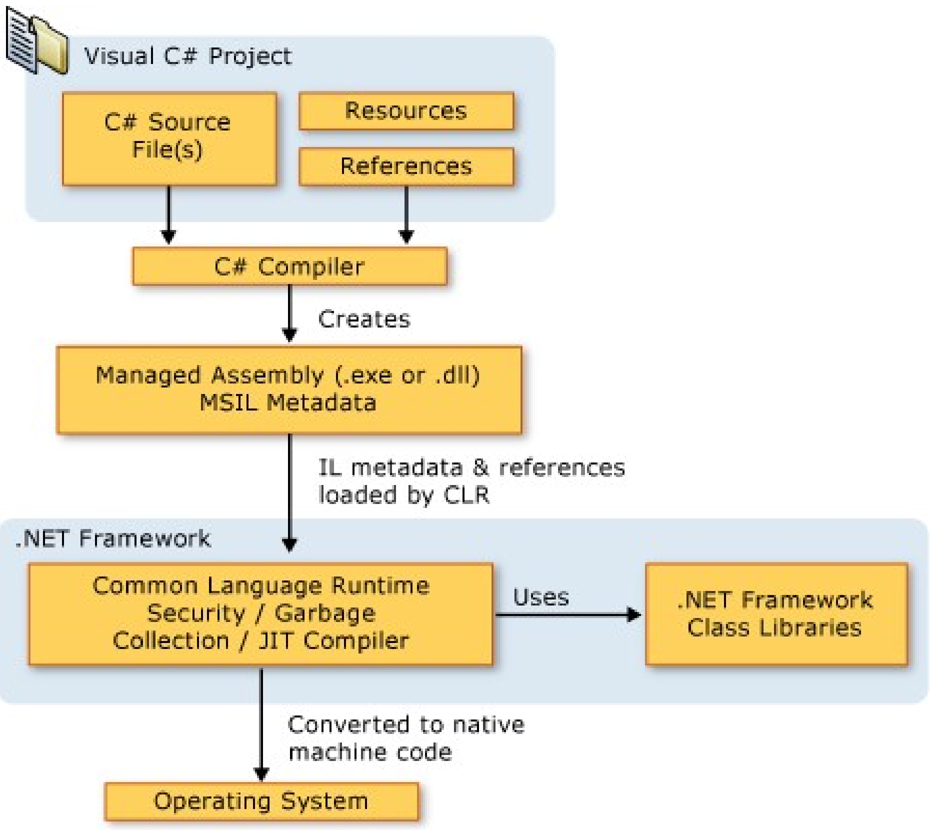
TUTORIAL 5: PROGRAM DESIGN & DEVELOPMENT

Introduction

In this last tutorial for Module 1, we shall consolidate our basic knowledge of programming by reviewing the widely-used internal functions available in C# and introducing the concept of user-defined functions. The general idea of a function is that it is a piece of code that performs a given task and may or may not return a value. The term *function* originates from the mathematical use of the term and it was widely used in older programming languages. However, in C# we refer to this concept as a ***method*** because it is slightly different in an object-oriented context. You will learn more about this in Module 2 which gives a thorough grounding in Object Oriented Programming. For now, we don't need to understand the technicalities but we should be aware that we can use some methods that are already available and we can create our own ones. The .NET environment contains a large library of methods that we can access in our programs - a typical example is the square root method introduced in an earlier tutorial, for example:

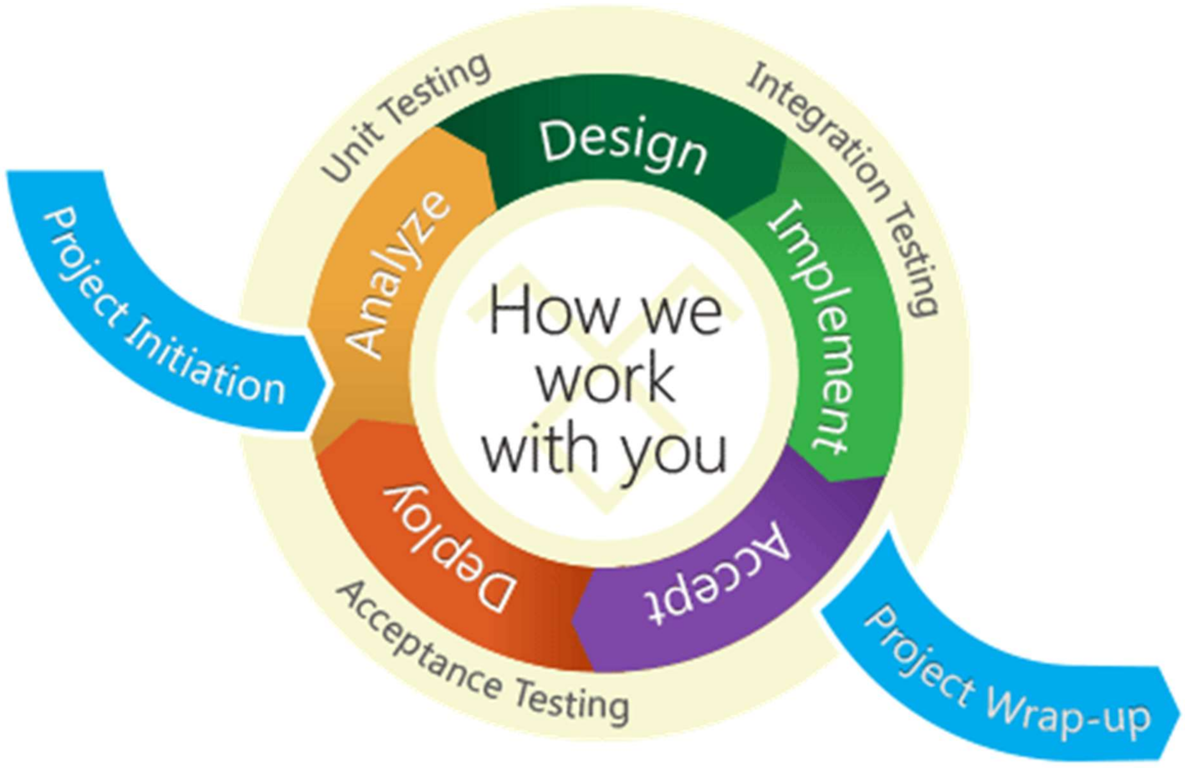
```
res = Math.Sqrt(n);
```

The method is part of the Class Libraries accessible by various programming languages within the .NET framework. The overall structure is shown below:


Visual C# Project
C# Source File(s)
Resources
References
C# Compiler
Creates
Managed Assembly (.exe or .dll) MSIL Metadata
IL metadata & references loaded by CLR
.NET Framework
Common Language Runtime Security / Garbage Collection / JIT Compiler
Uses
.NET Framework Class Libraries
Converted to native machine code
Operating System

We shall take an informal look at some basic software design principles and give an overview of testing methodology, practical implementation and documentation. We also briefly discuss how to find and avoid program errors.

This module lays the foundations for a formal study of software design and development in module 3 in which you will study the widely-used 'Agile' approach, illustrated below:



1. <u>Methods</u>

Consider the following simple problem: we have a program that performs some financial predictions on the value of an investment. For internal purposes the calculations involve floating point values, but when the displaying the result, it is required to output the result in a display in which the value has been rounded up or down to the nearest pound. The following is an illustration of a method in the Math class that will achieve this:

```
Math.Round(result);
```

Note that when you type in Math. Visual Studio displays a list of the various methods available and you can select the required one by double clicking on the name. Also, when you enter the opening parenthesis, the display indicates the required arguments). In tutorial 1, we used String methods Substring and Length. Visual Studio enables us to use an extensive range of methods for various purposes, but sometimes we need to write our own methods.

Creating your own methods

Although the .NET libraries provide you with many solutions to typical program requirements, you will find that you will want to create your own methods for your specific needs. Let's start off with a very simple example where we create a method that doubles an integer value supplied to it and returns the result.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MyMethodTest
{
    class Program
    {
        static void Main(string[] args)
        {

            Console.Write("Enter a value: ");
            Int32 value = Convert.ToInt32(Console.ReadLine());
            Int32 newvalue = Double(value);
            Console.WriteLine("Old value: {0}, New value: {1}", value, newvalue);
        }

        static Int32 Double(Int32 i)
        {
            return 2 * i;
        }
   }
}
```

You will find other examples of methods that involve 'classes' but, for the moment, stick to the format in the above example. You will learn about classes in module 2.

In the above example, the method returned an integer value and this had to be indicated in the declaration:

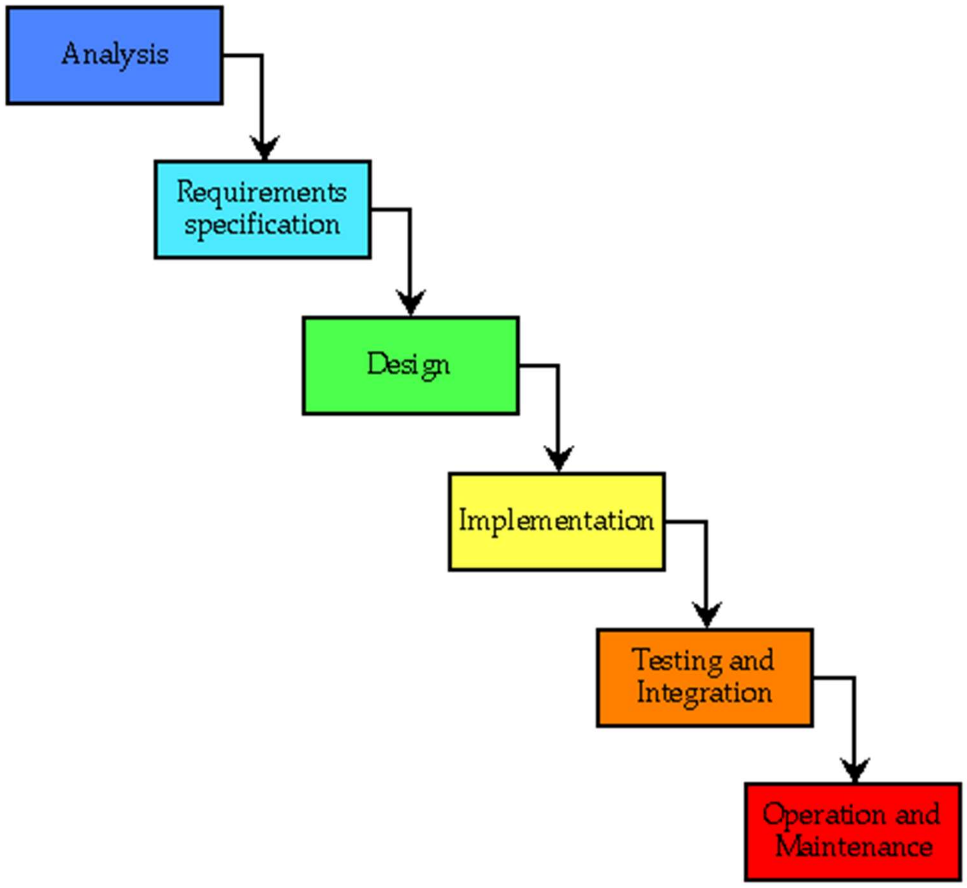
```
static Int32 Double(Int32 i)
```

If it just did something like printing a message, it would be declared as void e.g.

```
   class Program
    {
        static void Main(string[] args)
        {
            Console.Write("What is your name? ");
            string yourName = Console.ReadLine();
            SayHello(yourName);
        }
        static void SayHello(string name)
        {
            Console.WriteLine("Hello {0} ",name);
        }
    }
```

2. *Introduction to Software Development*

There are formal approaches to software development and these will be described in module 3 of this course. For now, consider the traditional 'Waterfall Model' as shown below:



This is fairly self-explanatory, but it does illustrate one crucial point – you DO NOT start by sitting down at the computer and typing code! Modern development environments such as Visual Studio make it very tempting to grab a form, add a few buttons and some boxes and try and progress towards a solution. However, a professional software developer understands the need for good design, testing and maintainability.

You will be writing a program for a customer who themselves may not have a very clear idea of exactly what they want. The analysis of the problem and subsequently establishing a formal requirements specification can be a complex and difficult process but, if done properly, can save a lot of problems later on. Large IT projects are notorious for over-running and being over-budget – this is often due to the customer changing their mind about exactly what they want, but perhaps some of this can be avoided by proper analysis in the first place!

3. *Errors and Debugging*

We saw in tutorial 2 that there are different types of programming errors: syntax, runtime, and logical errors. Syntax errors are picked up by the compiler and are generally easy to correct. Similarly, runtime errors are often easy to trace and correct, even if this involves stepping through the code using a debugging facility. Logical errors are, as we said, more difficult to trace because the code compiles and runs without any reported errors but produces strange results (not necessarily all the time). A debugger can be a help in tracing logical errors, although a long and complex program may prove to be a challenge. Logical errors are always likely to occur and may arise from some complicated situations.

One way to reduce the possibility of logical and run-time errors is to create structured programs in which individual tasks are identified and consigned to separate sections of code (in C# these would form methods). This enables individual routines with specified inputs and outputs to be written and tested separately. The program can then be assembled from a set of reliable 'building blocks'. For the moment, we can do this using simple methods as described earlier. In the next module, you will learn more about the formal structures that can provide robust code using object oriented programming.

It is sometimes possible to foresee runtime errors e.g. trying to open a file that doesn't exists. The process is known as '*Exception Handling*' and we can write program code such that, if an error occurs', we can deal with it rather than letting the program crash. The general format is as follows:

```
try
{
      program command(s)
}
catch (type of error)
{
      action to be taken when error occurs
}
finally
{
      do this regardless of whether error has occurred or not
}
```

Note that the 'finally' block is optional and often may not be required. An example situation is given below which could form part of a simple calculator program. One of the operations would be division and if we attempt to divide by zero, the program would normally crash. Our rather simplistic program requires the user to enter integer values, catches any 'divide-by-zero error' and sends the user a message to tell them that the denominator is incorrect.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ExceptionHandling
{
    class Program
    {
        static void Main(string[] args)
        {
            int x, y, result;
            Console.Write("Division. Please enter the numerator: ");
            x = Convert.ToInt32(Console.ReadLine());
            Console.Write("Division. Please enter the denominator: ");
            y = Convert.ToInt32(Console.ReadLine());

            try
            {
                result = x / y;
                Console.WriteLine("Result = {0}", Convert.ToString(result));
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }
    }
}
```

In the above program, what happens if the user enters a decimal value? What could you do to avoid this? How could you output a correct decimal result from the division of the two integer values?

Note that, in the above example, we have caught any type of exception that may have occurred. Suppose, however that we wanted to do different things for different types of exceptions. This means that we have to identify the particular exception within the 'catch' statement and repeat the 'catch' for different types of error. The general principle is outlined below:

```
            try
            {
                program commands
            }
            catch (IndexOutOfRangeException)
            {
                Console.WriteLine("array A index out of range – check program code!");
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
```

4. *Reading and Writing to Disk Files*

So far, we have entered data from the keyboard using Console.ReadLine(). If we were dealing with a large amount of data e.g. a thousand data values, this would be impractical. Instead, we can read the data from a file on a disk drive. Similarly, we can output results to a file. A simple example is given below in which we write some text to a file and then read it back in.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO; // add this to the standard list
```

```
namespace FileTest
{
    class Program
    {
        static void Main(string[] args)
        {
            // specify the file location
            string myFile = "c:\\Temp\\test.txt"; // double backslash needed in path spec.

            // write some data to the file
            using(StreamWriter sw=new StreamWriter(myFile, true)) // remove ', true' to overwrite
            {
                sw.WriteLine("This is a test");
                sw.WriteLine("Here is another line");
                sw.Close();
            }

            // now read the data from the file
            if (File.Exists(myFile))
            {
                using(StreamReader sr=new StreamReader(myFile))
                {
                    while (!sr.EndOfStream)
                    {
                        Console.WriteLine(sr.ReadLine());
                    }
                    sr.Close();
                }
            }
        }
    }
}
```

Note that you can write values to a file e.g.

```
double pi = 3.14159;
       :
       :
sw.WriteLine(pi);
```

However, the data file will store this in text format so you will need to read such values in as text and convert back to numeric format as you do so.

Additional Notes

In this tutorial, we have introduced some further programming concepts and have tried to stress the importance of software design as part of the development process as well as indicating some approaches to preventing and detecting program errors. A formal approach to this topic is presented in module 3 which will form a key part of your understanding of the role of a professional software developer.

For the moment, we wish to conclude this module by ensuring that we can analyse a given problem, produce a requirements specification, write the program and then thoroughly test it.

Notes on Good Practice

A general approach to good practice would be to follow the broad outlines of the 'Waterfall Model':

1*. Define the Problem*

Ensure that you fully understand exactly what is required. Also appreciate that client may not have thought through all the implications. A good approach is 'but what if …' to most situations. Remember that the client may be too familiar with the process to identify odd situations that might be outside the program's remit.

2. *Define the Requirements*

A requirements analysis should formally identify the full function of a program and feature the required inputs and outputs. It may be helpful for the programmer to produce sample screen layouts as part of this process.

3. *Design the Solution*

Once the client has accepted a design specification based upon the requirements analysis, the programmer can develop the software implementation. This should take the form of a well-structured program in which individual components can be independently tested and there should be appropriate documentation both within and outside the program.

4. *Test the Solution*

A robust testing strategy needs to be developed for the program that should cause various program runs to execute all branches of the program and explore all possibilities of user input. The design phase should have reduced the possibilities of errors in user input e.g. by the use of drop-down lists and calendar entries.

5. *Implement the Solution*

The final version of the program will need to be installed on the client's system and this will usually require the cooperation of their IT staff. You should ensure that it is up and running and available to all relevant staff. You will usually need to provide some staff training on how to use your new system.

When developing software for a client, the programmer must try to prevent illegal copying. One possibility is to require the user to register their copy and, in return, receive an authentication code. The program will generate a serial number that is required for registration and the codes can be checked against carefully hidden details.

6. *Maintain the System*

There is no point in providing an IT system without some form of support system. There should be a support system available (built-in help files, on-line support and possible on-site visits). Most contracts will involve details of on-going support and relevant costs. There should be a facility for ongoing updates to the software together with an agreement regarding any possible costs.

On-Line Resources for Further Study

The following website gives examples of the String method:

http://www.completecsharptutorial.com/csharp-articles/csharp-string-function/

The following links provide videos on the topics covered:

https://www.youtube.com/watch?v=qoSq_GYD0Q8	(Requirements Analysis)
https://www.youtube.com/watch?v=QwygwfqOHsI	(Testing)
https://www.youtube.com/watch?v=QwygwfqOHsI	(Methods)

An explanation and sample code for exception handling can be found at:

https://www.tutorialspoint.com/csharp/csharp_exception_handling.htm

Some further details on functions and parameters can be found at:

http://csharp.net-tutorials.com/basics/function-parameters/

Core Practical Exercises

1. **Console Programs**

1.1 Write a program that will input a value, N, and then calculate the sum of first N natural numbers

(1 + 2 + 3 + ……….. + N)

1.2 The factorial symbol uses the exclamation mark to indicate that a whole number is to be multiplied by successive lower numbers down to 1 e.g. 5! = 5 x 4 x 3 x 2 x 1 = 120. Write a program to input a value of N and calculate the value of N! You should check that N is greater than 0, but you can allow N = 1.

1.3 A local social club keeps a list of its members who are required to renew their membership annually. Their names and renewal dates are stored in a 2-D array (in this case defined within the program). Design and write a program that allows the club secretary to view a list of names whose membership has expired. Also produce a list of those people whose membership will expire within the current month.

2. **Windows Form Applications**

2.1 Design and write a program that will Read a list of names from a text file and display them in a Listbox when the Form loads. The Form should include a search facility whereby the user can enter a name and click a button to start a search of the names in the Listbox for the required name.
If the name is found it should be highlighted in the Listbox, otherwise a suitable message should be displayed

2.2 Create a similar program to 2.1 but this time each line of the text file will contain a name and telephone number separated by a comma. The program should display only the list of names in a drop down selection box. When the user selects a name the corresponding telephone number should be displayed in the Textbox.

Hint: If each line is read into a string variable name line, the 2 components can be extracted to a string array "items" by: items = line.Split(,);

2.3 Design and write a simple calculator program that enables the user to enter two values, x and y, and to provide the facilities of addition (x+y), subtraction (x-y), multiplication (x * y), and division (x/y). Use separate methods for each arithmetic operation.

Further Exercises

1. The exercise in 1.2 featured factorials. You might have noticed that N! is equal to N x (N-1)! and this suggests an alternative programming technique – recursion (where a function can call itself). In this case, if we have a function (method) *factorial(n)*, it would return the value 1 if n was equal to 1, but otherwise return n x factorial(n-1). Design and write a program that enables the user to input N and to output N! using recursion.

   What should the programmer consider when writing programs that involve recursion?

2. Take the following scenario: you live in a small community and you are known to have some knowledge about computers. A local company approaches you with a view to developing a website for them so that they can provide an on-line shopping facility. How should you respond to this opportunity if you regard yourself as a professional software developer?