C# Functions

Functions are key to procedural programming. In other langauges the same concept might be referred to as methods or procedures. They enable pieces of code to be re-used. If the function body is updated the code that uses that function will instantly pick up on the new implementation of the function. For example we have used routines to grab the 1st character of a String and assign it to a char. Rather than reewriting several lines of code each time this process is required the same function can be reused.

`static void Main(string[] args){ }` is a function, it is the first line of cocde that C# calls when it runs. A Console based C# program must include a Main() function.

In the following line of code the system function Parse() is called to convert a String to an int. C# is an Object Orientated language. What it is saying here is that Parse() is a function (method) of the static class Int16.

```
int num = Int16.Parse("2");
```

There are many built in C# functions but you also get to write your own. There are 2 parts; the body of the function which defines what processing will go on and the calling of the function at some point in the code. A function that calls itself is a recursive function.

Here is simple program that shows a function being declared and used (called)

```
class Program
{
      static void doNothing() //declaration of function
       {   //function body
           Console.WriteLine("got here");
       }
       static void Main(string[] args)
       {
           doNothing(); //call function in Main
           Console.ReadLine();
       }
}
```

The function doNothing is static because we do not need to declare an instance of the class Program to use it.

Here a, slightly, more useful program takes a number and doubles it.

```
static void doDouble(int num) //accept an int
{   //function body
            num = num * 2;
            Console.WriteLine("num is "  + num);
}
static void Main(string[] args)
{
            doDouble(4); //call function in Main
            Console.ReadLine();
}
```

This function must be passed an int for it to work. The line `doDouble();` would cause an error as no value has been passed to the function. This line, `doDouble(4.40f);` is also asking for trouble as a float not an int is being passed to the function.

Here is a better implementation of the function. This accepts and returns an int. The output is in the main rather than within the function. It not usually a good idea to have functions output to the user as they are designed to be reused and might spit out irelevant details.

```
static int doBetterDouble(int num) //accept an int
{
    num = num * 2;
    return num; //we now need a return value
}
static void Main(string[] args)
{
    int ans = doBetterDouble(4); //there must be a value to assign
    //the return value of the function to
    Console.WriteLine("ans is " + ans);
    Console.ReadLine();
}
```

Any valid data can be sent to the function.

```
int goodNum = 3;
int ans = doBetterDouble(goodNum); //good
float badNum = 3.14157f;
int ans = doBetterDouble(badNum); //bad
```

It is the programmer's responsibility to ensure that data of an appropriate type is passed to the function and that the return value is handled correctly.

The previous functions demonstrate the principles of how functions work but do not achieve anything that could not have been done better within a few lines of code in main().

There are many cases where a cunning algorithm needs to be used more than once where functions come into their own. Here the routine to grab the 1st letter of a String and assign it to a char has been placed in a nice tidy function.

```
static char stringToChar(String word) //accept a String
{   //function body
    String temp = word.Substring(0, 1);
    char letter =Char.Parse(temp);
    return letter;
}
static void Main(string[] args)
{
    Console.WriteLine("Enter some text " );
    String text = Console.ReadLine();
    char letter1 = stringToChar(text);
    Console.WriteLine("letter1 is " + letter1 + " text is " + text);
    Console.ReadLine();
}
```

If several letters were to be input the stringToChar function can be called several times without needed to type the Substring and Parse lines over and over.

Using functions

The following functions are all trivial but follow the correct syntax rules. Create a program that uses all of them, passing the correct data types and assigning the return values to appropriate variables.

```
static int number0()
{
     int value = 0;
     return value;
}

static double circum (float radius)
{
      double circle = (double)radius;
      circle = System.Math.PI * radius * 2;
      return circle;
}

static bool divBy5(int num)
{
     int div = num % 5;
     if (div==0)
        return true;
     else
        return false;
}
```

```
static bool goesInTo(int numTop, int numBot)
{
     int div = numTop % numBot;
     if (div==0)
                return true;
     else
                return false;
}
```

Exercises

1. Create a function to convert degrees to radians. This should accept and return a double. 1 radian is 180/pi degrees.
2. Create a function to return an int random number between 1 and 100. This function will return but not accept data. This is a modification of the code used in the selection worksheet exercise 2.
3. Modify the function to return an int random number up to an int value passed to that function rather than capping the upper limit at 100.
4. Modify the stringToChar function to only return a letter. If the String passed to the function is not a letter it should return the rogue value -1. `char.IsLetter()` can be used to test the value of the char within the function.