**PYTHON WEEK 2 DAY 2 FUNCTIONS**

**Using Functions**

We create functions by providing three pieces of information. The name of the function, a list of zero or more parameters, and, optionally, a block of code which provides the return value (a function can return nothing).

We normally define functions in script files for the simple reason that we do not want to type them more than once, we just want to edit a function (if necessary).

We must make a firm distinction between an argument value and a parameter:

·         **Argument**

The argument is the object used in an application of a function; it may be referenced by other variables or objects.

·         **Parameter**

The parameter is a variable name that is part of the function and is a local variable within the function body.

We define a function by using the following syntax:

|  |
| --- |
| def function\_name(parameter <,...>):  #suite |

The function\_name part is the name used to call a function. The parameter part shows that zero or more parameters can be given to a function. The suite part is where the functionality is coded.

The first line of a function’s suite should always be a comment**.** This comment must briefly explain what the function does.

When the return statement is included in the function’s suite, it means that the function returns a result, for example a calculated value. The return statement can also be used to break out of a function.

There are three types of functions in Python:

·         **Ordinary** functions

·         **Procedure** functions

·         **Factory** functions

**Ordinary** functions are functions that follow mathematical procedures. They will receive an argument, perform a specific calculation with the argument, and return a result.

**Procedure** functions normally do not return a result; they are called to execute a procedure. For example a function can be created to set up a connection to a database.

**Factory** functions do not take parameters. The function generates values. Some factory functions work by accessing an object encapsulated in a module. For example, you will access the random number generator encapsulated in the random module.

The following is an example of how programs use functions to be more efficient:

**Example 1 – Function:**

|  |
| --- |
| def calculateTax(salary):    """Calculates and prints a given salary’s tax"""    if salary > 30000 :      rate = 0.2    elif salary > 10000 :      rate = 0.15    else:      rate = 0.1    tax = salary \* rate    print (tax)    print ("Enter the amount on which you want to calculate tax:")  calculateTax(int(input())); |

Notice that the function must be defined before it can be called. The program is divided into two separate programs: the main program and the calculateTax function are separate pieces of code. Each time calculateTax() is called, it will run through the function’s piece of code.

The rate and tax variables are only accessible in the calculateTax function itself; you cannot use or access them outside of calculateTax. The variables rate, tax, and salary are known as local variables, because they are local to the function in which they are declared and cannot be accessed by other functions.

If we want the value of a local variable to be available in another part of the application, one way of doing this is to return the value to the program or function which called it. For example, if we want the value of the tax variable to be available in the main program, we can use a return statement to accomplish this. Remember that it is only the value of the variable that is being returned and not the variable itself.

**Example 2 – Default parameters:**

|  |
| --- |
| 1 x = 15  2  3 def num(y = x):  4   """This function assigns a default value to y"""  5   return y  6  7 x = 5             #Reassign 'x' to 5.  8 print (num())     #Returns 15 (default value not changed) |

Using default parameters in functions has one advantage. You would automatically assign a default value to a parameter within a function. The default value would only be used if a function call does not include a specific parameter’s value; the parameter would then take the default value that was specified in the function’s declaration.

The above example assigns 15 to the x variable. A function is then declared called num. The num function has only one parameter (y), which has a default value of 15. Line 7 changes the value of x to 5. This change would not affect the default value of y, because num has already been declared. Line 8 would then thus print 15.

**The random module**

The basic random() function generates a random floating point number as output. Python uses the Mersenne Twister as the core generator.

The Mersenne Twister is an algorithm for generating random numbers. It has a very long period before its sequence of numbers will repeat (219937 – 1). It is also a very fast.

The functions supplied by the random module are actually bound methods of a hidden instance of the random.Random class.

The following example is an example of a function with the use of the random module:

**Example 3 – Random module:**

|  |
| --- |
| import random    def lotto\_number():    """The result from a lotto number draw is returned."""    num = random.randrange(1,47)    return str(num)    for n in range(1,6):    print (lotto\_number()) |

Read through the program and try to find a flaw. This program will work, but not as it should. The program contains a logical error, because there are no checks to determine if a number is drawn more than once. There are various ways double values can be checked and avoided. This will be covered later on.

Some of the other methods provided by the random module are listed below:

**Table 1 – Random functions**

|  |  |
| --- | --- |
| **function** | **Description** |
| randint(self, x, y) | Return random integer in range [x, y], including both end points. |
| sample(self, popu, i) | Chooses i unique random elements from a population sequence or set.    Returns a new list containing elements while leaving the original population unchanged. (Lists are explained in Unit 2.)    The resulting list is in selection order. For example, 12, 45, 32, 16, 1 |
| choice(self, seq) | Choose a random element from a non-empty sequence. |
| randrange(x,y) | Return random integer in range [x, y), including x but excluding y. |

|  |  |
| --- | --- |
| **NOTE** | The sample, randint and choice functions will be used later in this course. |
|  |

**Example:**

Write a program that requests the user to input three numbers. These numbers must be passed to two functions:

* A function that calculates and prints the product of the numbers.

* A second function that calculates and returns the average of the numbers.

The average returned by the function is then printed.

**Example 4 – Two functions working together:**

|  |
| --- |
| 1 def determineProduct(val1, val2, val3):  2   product = val1 \* val2 \* val3  3   print ("The product of the numbers is: " + str(product))  4  5 def determineAve(val1, val2, val3):  6   average = (val1 + val2 + val3) / 3  7   return average  8  9 print ("Enter 3 numbers:")  10 num1 = int(input("Number %d: " % x))  11 num2 = int(input("Number %d: " % x))  12 num3 = int(input("Number %d: " % x))  13  14 determineProduct(num1, num2, num3)  15 result = determineAve(num1, num2, num3)  16 print ("The average of the numbers is:", str(result)) |

**Output*:***

|  |
| --- |
| >>>  Enter 3 numbers:  Number 1: 25  Number 2: 31  Number 3: 15  The product of the numbers is: 11625  The average of the numbers is: 23.6666666667  >>> |

The following points can be noted regarding the previous example:

·         The main program calls both functions, one after another. The first function (lines 1 to 3) receives copies of all three numbers input by the user into the parameter variables, val1, val2, and val3. (lines 9 to 12).

·         After determining and printing the product of the numbers (line 3), the program breaks out of the function and continues to the next line of code, where the determineAve function is called (line 14).

·         This function receives the three numbers as input parameters, calculates the average of the numbers, and returns the average (lines 5 to 7).

·         The program receives the average which is contained within in the result variable; the result variable is then printed (line 16).

**Recursive functions**

Recursion in programming is a method where the solution to a problem depends on the same solution’s result.

A recursive function calls itself when it is executed.

The following example uses a recursive function called Fibonacci.

In the Fibonacci sequence of numbers, each number is the sum of the previous two numbers, starting with 0 and 1. This sequence begins 0, 1, 1, 2, 3, 5, 8, 13, 21, 34…

**Example 5 – Fibonacci recursive function:**

|  |
| --- |
| def Fibonacci(num):      if num <= 1:  # base case     return 1    else:     return num \* Fibonacci(num - 1) # recursive call    for i in range(11):    print ("%2d = %d" % (i, Fibonacci(i))) |

**Output*:***

|  |
| --- |
| >>>   0 = 1   1 = 1   2 = 2   3 = 6   4 = 24   5 = 120   6 = 720   7 = 5040   8 = 40320   9 = 362880  10 = 3628800  >>> |

**Activity 1:** This activity will be an **individual** activity and will be graded.

**Time**: 30 minutes

#### **Instructions**

Write a program that prompts the user to input the maximum temperatures, in degrees Fahrenheit (° F), recorded for a city over the last two days. A function converts the temperatures to degrees Celsius (° C), and returns these values to a variable which prints the temperatures in ° C. Another function is then called, which determines and prints a message stating which of the two days was the coldest, or if the temperature was the same.

The formula to convert °F to °C is:

C = 5 / 9 \* (F –32)

**Activity 2:** This activity will be a **dev team** activity and will be graded.

**Time**: 60 minutes

#### **Instructions**

Write a program that generates a random number (make use of a function to generate these random numbers). It then prompts the user to guess the number in a range of 1 to 20, with clues of ‘too high’ or ‘too low’ whenever the user inputs a number higher or lower than the randomly generated number respectively, until the user guesses it.

Your output should be like the following:

|  |
| --- |
| >>>  I am busy thinking of a number, can you guess it…?  Guess the number between 1 and 20: 4  “Too Low”    I am busy thinking of a number, can you guess it…?  Guess the number between 1 and 20: 10  “Too Low”    I am busy thinking of a number, can you guess it…?  Guess the number between 1 and 20: 14  “Congratulations, you’ve guessed it”  >>> |