**Lab 4: Functions**

Learning Outcomes:

* Understand how to write your own functions and invoke them
* Know the difference between invoking a built-in function and function of an imported module

Instructions:

* Suggest that you create a working folder **is111\lab4** in your **C** or **D** drive. Store all your solutions in this working folder.
* Challenging questions are marked with \*.

To submit:

* Please submit your working solutions via your assignment Dropbox in eLearn **within 1 week**. The Dropbox will be closed after the due date.
* Zip up all your source files into a single zip file called **<your email ID>\_lab4.zip** (e.g. **ahlian.lim.2011\_lab4.zip**). You should only submit a single zip file for each lab.

1. Copy the code below into a file named **lab4\_1.py** and complete the program as stated below:

Write code to define a function called is\_factor that takes in 2 parameters n and f and returns True if f is a factor of n and False otherwise.   
Assume that the user invokes the function with only integer values as shown in lines 08 and 09.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10 | **# Author: <your name>**  **# write code for function is\_factor here**  print('3 is a factor of 10?:', is\_factor(10,3))  print('2 is a factor of 10?:', is\_factor(10,2)) |

**Note:** Lines 08 and 09 do not belong to the function and are present to test the function written.

You can test your function by passing different values to it.

Here is a sample output when **lab4\_1** is run (with the given test code):

|  |
| --- |
| D:\is111\lab4>**python lab4\_1.py**  3 is a factor of 10: False  2 is a factor of 10: True |

To submit**: lab4\_1.py**

1. Write the output of the following two programs.

(a)

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12 | def m1(x):  str\_output = ''  for i in range(1,x+1):  str\_output += str(x)    print(str\_output)  def m2(x):  for i in range(1,x+1):  m1(i)    m2(5) |

(b)

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16 | def func1(a):  a += 1  print('in func1: calling func2')  func2(a, 1)  print('in func1: called func2')  return a  5 1  def func2(a, b):  a = a \* 10 50  c = a + b 51  print('in func2')  print(str(a) + ' + ' + str(b) + ' = ' + str(c))    a = func1(4)  print("called func1:",a) |

Output:

112123123412345

in func1: calling func2

in func2

50 + 1 + 51

in func1: called func2

called func1: 50

1

22

333

4444

55555

called func1: 5

\*note: calling function within a function and assigning value to the variable will not be updated

i.e variable a was not updated. Remains as a += 1, a = 5

To submit: -

1. In a file named **lab4\_3.py**, define a function named sum\_of\_powers that takes in 2 parameters m and n. Compute the sum of the powers from m0 to mn (i.e m0 + m1 + m2 + … + mn) and returns the sum.

To test the function written, write code in the same file (outside of the function body) to get 2 integer inputs from the user for m and n. Then call the function sum\_of\_powers passing the values of m and n.

You can assume that the user enters positive integers.

Here is a sample output when **lab4\_3** is run:

|  |
| --- |
| D:\is111\lab4>**python lab4\_3.py**  Enter value for m:**2**  Enter value for n:**4**  Sum of all powers is 31 |

To submit**: lab4\_3.py**

1. Complete the function whose definition is shown below to print all perfect squares between the 2 parameters (min and max). Copy the code given below in a file named **lab4\_4.py**

Note: You can assume that the function is invoked with positive integer values for min and max.

To get the required range of squares to be printed, you could use the sqrt method of math module.

>>> import math

>>> math.sqrt(25)

5.0

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11 | def print\_squares(min, max):  # complete the code    print\_squares(10, 100) |

The expected output with the given test code (perfect squares between 10 and 100):

|  |
| --- |
| D:\is111\lab4>**python lab4\_4.py**  16 25 36 49 64 81 100 |

To submit: **lab4**\_**4.py**

1. In a file named **lab4\_5.py** define a function called count\_characters that takes in 2 parameters, a string say s and a single character say c. The function is supposed to return the number of character c appearing in the input parameter s or None if no c character appears in s. You are NOT allowed to make use of count method of string.

Here are sample outputs when **lab4\_5** is run:

When the function is invoked as

print("Number of characters 'l' in 'hello':",count\_characters("hello", 'l'))

|  |
| --- |
| D:\is111\lab4>**python lab4\_5.py**  Number of characters 'l' in 'hello': 2 |

When the function is invoked as

print("Number of characters 'e' in 'HELLO':",count\_characters("HELLO", 'e'))

|  |
| --- |
| D:\is111\lab4>**python lab4\_5.py**  Number of characters 'e' in 'HELLO': None |

When the function is invoked as

print("e's" ,'in', "hello there:",count\_characters("hello there", 'e'))

|  |
| --- |
| D:\is111\lab4>**python lab4\_5.py**  e's in hello there: 3 |

To submit**: lab4\_5.py**

1. \*You have written a program earlier that compresses the string using “run length encoding” algorithm. As an example consider the string AAAABBCCAD, after encoding it becomes A4B2C2A1D1.   
     
   In a file named **lab4\_6.py** define two functions called encode and decode.   
   The function encode takes in a string and returns the encoded string.   
   The function decode, on the other hand takes in the compressed string and returns the original string. For example, compressed string A4B2C2D1 becomes AAAABBCCD. As another example, the string 14021303 after decoding becomes 111100111000. You can assume that a valid encoded string is passed into the function.

Here are sample outputs when **lab4\_6** is run:

When the function is invoked as print("Encoded string:" + encode("AAAABBCCAD"))

|  |
| --- |
| D:\is111\lab4>**python lab4\_6.py**  Encoded string:A4B2C2A1D1 |

When the function is invoked as print("After decoding:" + decode("a4b3C2D1"))

|  |
| --- |
| D:\is111\lab4>**python lab4\_6.py**  After decoding:aaaabbbCCD |

When the function is invoked as print("After decoding:" + decode("14021303"))

|  |
| --- |
| D:\is111\lab4>**python lab4\_6.py**  After decoding:111100111000 |

To submit**: lab4\_6.py**

1. \*A sandwich typically consists of cheese, meat or/and vegetables placed in between two slices of bread.

Consider a string “remember”, the sub-string “memb” is sandwiched between “re” and its mirror “er” on the other side. In the file named **lab4\_7.py** write a function named get\_sandwich that takes in a string and returns the sandwiched string, if any, or None.

Here are some sample outputs when **lab4\_7** is run:

When the function is invoked as

print("Sandwich substring:" + get\_sandwich("toast"))  
print("Sandwich substring:" + get\_sandwich("foolproof"))

|  |
| --- |
| D:\is111\lab4>**python lab4\_7.py**  Sandwich substring:oas Sandwich substring:lpr |

When the function is invoked as

print(">" + get\_sandwich("peep") + "<") #(it returns an empty string)

print(get\_sandwich("robot"))

|  |
| --- |
| D:\is111\lab4>**python lab4\_7.py**  ><  None |

To submit**: lab4\_7.py**