USMAN INSTITUTE OF TECHNOLOGY

Department of Computer Science
CS121 Object Oriented Programming

Lab # 05
Functions in Python

Objective:

This experiment introduces the students to the concept of Functions in Python programming language

Name of Student: _____

Roll No: _____ **Sec.** _____

Date of Experiment: _____

Marks Obtained/Remarks: _____

Signature: _____

Lab 01: Introduction to Python Programming

Introduction to Python

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

Python is Interpreted: Python is processed at runtime by the interpreter. You do not need to compile your program before executing it.

Python is Interactive: You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

Python is Object-Oriented: Python supports Object-Oriented style or technique of programming that encapsulates code within objects.

Python is a Beginner's Language: Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

Python Program

If you are running new version of Python, then you would need to use print statement with parenthesis as in **print ("Hello, Python!")**. However, in Python version 2.4.x, you do not need the parenthesis. The above line produces the following result:

```
Hello, Python!
```

Quotation in Python

Python accepts single ('), double (") and triple (''' or """) quotes to denote string literals, as long as the same type of quote starts and ends the string. The triple quotes are used to span the string across multiple lines. For example, all the following are legal

```
word = 'word'
sentence = "This is a sentence."
paragraph = """This is a paragraph. It is
made up of multiple lines and sentences."""
```

Comments in Python

A hash sign (#), that is not inside a string literal, begins a comment. All characters after the # and up to the end of the physical line are part of the comment and the Python interpreter ignores them.

Assigning Values to Variables

Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable. The equal sign (=) is used to assign values to variables. The operand to the left of the = operator is the name of the variable and the operand to the right of the = operator is the value stored in the variable.

```
counter = 100          # An integer assignment
miles   = 1000.0       # A floating point
name    = "Usman"      # A string
print (counter)
print (miles); print (name)
```

Data Types

The data stored in memory can be of many types. For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters. Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.
Python has five standard data types

- Numbers
- String
- List
- Tuple
- Dictionary

Python Numbers

Number data types store numeric values. Number objects are created when you assign a value to them.

```
var1 = 1
var2 = 10
```

Python Strings

Strings in Python are identified as a contiguous set of characters represented in the quotation marks. Python allows for either pairs of single or double quotes. Subsets of strings can be taken using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.

```
str = 'Hello World!'
print (str)          # Prints complete string
print (str[0])       # Prints first character of the string
print (str[2:5])     # Prints characters starting from 3rd to 5th
print (str[2:])      # Prints string starting from 3rd character
print (str * 2)      # Prints string two times
print (str + "TEST") # Prints concatenated string
```

This will produce the following result

```
Hello World!
H
Llo
llo World!
Hello World!Hello World!
Hello World!TEST
```

Python Lists

The list is the most versatile data type available in Python, which can be written as a list of comma-separated values (items) between square brackets. Important thing about a list is that the items in a list need not be of the same type. Creating a list is as simple as putting different comma-separated values between square brackets.

```
list1 = ['physics', 'chemistry', 1997, 2000]
list2 = [1, 2, 3, 4, 5, 6, 7 ]
print ("list1[0]: ", list1[0])
print ("list2[1:5]: ", list2[1:5])
```

When the above code is executed, it produces the following result

```
list1[0]:  physics
list2[1:5]:  [2, 3, 4, 5]
```

Updating Lists

You can update single or multiple elements of lists by giving the slice on the left-hand side of the assignment operator, and you can add to elements in a list with the append() method.

```
list = ['physics', 'chemistry', 1997, 2000]
print ("Value available at index 2 : ", list[2])

list[2] = 2001
print ("New value available at index 2 : ", list[2])
```

Delete List Elements

To remove a list element, you can use either the del statement if you know exactly which element(s) you are deleting. You can use the remove() method if you do not know exactly what is the index of the item to delete.

```
list = ['physics', 'chemistry', 1997, 2000]

print (list)
del list[2]
print ("After deleting value at index 2 : ", list)
```

Python Tuples

A tuple is another sequence data type that is similar to the list. A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parenthesis. The main difference between lists and tuples are − Lists are enclosed in brackets ([ ]) and their elements and size can be changed, while tuples are enclosed in parentheses (( )) and cannot be updated. Tuples can be thought of as read-only lists.

```
tuple = ( 'abcd', 786 , 2.23, 'john', 70.2  )
tinytuple = (123, 'john')
print (tuple)             # Prints complete tuple
print (tuple[0])          # Prints first element of the tuple
print (tuple[1:3])        # Prints elements starting from 2nd till 3rd
print (tuple[2:])         # Prints elements starting from 3rd element
print (tinytuple * 2)     # Prints tuple two times
print (tuple + tinytuple) # Prints concatenated tuple
```

This produces the following result

```
('abcd', 786, 2.23, 'john', 70.200000000000003)
abcd
(786, 2.23)
(2.23, 'john', 70.200000000000003)
(123, 'john', 123, 'john')
```

```
('abcd', 786, 2.23, 'john', 70.200000000000003, 123, 'john')
```

The following code is invalid with tuple, because we attempted to update a tuple, which is not allowed. Similar case is possible with lists

```
tuple = ( 'abcd', 786 , 2.23, 'john', 70.2  )
list = [ 'abcd', 786 , 2.23, 'john', 70.2  ]
tuple[2] = 1000    # Invalid syntax with tuple
list[2] = 1000     # Valid syntax with list
```

Python Dictionary

Python's dictionaries are kind of hash-table type. They consist of key-value pairs. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object. Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]).

```
dict = {}
dict['one'] = "This is one"
dict[2]     = "This is two"
tinydict = {'name': 'john','code':6734, 'dept': 'sales'}

print (dict['one'])       # Prints value for 'one' key
print (dict[2])           # Prints value for 2 key
print (tinydict)          # Prints complete dictionary
print (tinydict.keys())   # Prints all the keys
print (tinydict.values()) # Prints all the values
```

This produces the following result

```
This is one
This is two
{'name': 'john', 'dept': 'sales', 'code': 6734}
dict_keys(['name', 'dept', 'code'])
dict_values(['john', 'sales', 6734])
```

Python 3 - Decision Making

Decision-making is the anticipation of conditions occurring during the execution of a program and specified actions taken according to the conditions. Decision structures evaluate multiple expressions, which produce TRUE or FALSE as the outc ome. You need to determine which action to take and which statements to execute if the outcome is TRUE or FALSE otherwise. Following is the general form of a typical decision making structure found in most of the programming languages.

IF Statement

The IF statement is similar to that of other languages. The if statement contains a logical expression using which the data is compared and a decision is made based on the result of the comparison.

```
if expression:
   statement(s)
```

```
var1 = 100
if var1:
   print ("1 - Got a true expression value")
```

```
   print (var1)
var2 = 0
if var2:
   print ("2 - Got a true expression value")
   print (var2)
print ("Good bye!")
```

IF...ELIF...ELSE Statements

An else statement can be combined with an if statement. An else statement contains a block of code that executes if the conditional expression in the if statement resolves to 0 or a FALSE value.
The else statement is an optional statement and there could be at the most only one else statement following if.

```
if expression:
   statement(s)

else:
   statement(s)
```

```
amount = int(input("Enter amount: "))

if amount<1000:
   discount = amount*0.05
   print ("Discount",discount)
else:
   discount = amount*0.10
   print ("Discount",discount)

print ("Net payable:",amount-discount)
```

The elif statement allows you to check multiple expressions for TRUE and execute a block of code as soon as one of the conditions evaluates to TRUE.
Similar to the else, the elif statement is optional. However, unlike else, for which there can be at the most one statement, there can be an arbitrary number of elif statements following an if.

```
if expression1:
   statement(s)
elif expression2:
   statement(s)
elif expression3:
   statement(s)
else:
   statement(s)
amount = int(input("Enter amount: "))

if amount<1000:
   discount = amount*0.05
   print ("Discount",discount)
elif amount<5000:
   discount = amount*0.10
   print ("Discount",discount)
else:
   discount = amount*0.15
```

```
   print ("Discount",discount)

print ("Net payable:",amount-discount)
```

Loops

In general, statements are executed sequentially − The first statement in a function is executed first, followed by the second, and so on. There may be a situation when you need to execute a block of code several number of times.

Programming languages provide various control structures that allow more complicated execution paths. A loop statement allows us to execute a statement or group of statements multiple times. The following diagram illustrates a loop statement.

While Loop Statements

A while loop statement in Python programming language repeatedly executes a target statement as long as a given condition is true.

```
count = 0
while (count < 9):
   print ('The count is:', count)
   count = count + 1

print ("Good bye!")
```

For Loop Statements

The for statement in Python has the ability to iterate over the items of any sequence, such as a list or a string.

```
>>> for var in list(range(5)):
   print (var)
```

```
for letter in 'Python':     # traversal of a string sequence
   print ('Current Letter :', letter)
print()
fruits = ['banana', 'apple',  'mango']
for fruit in fruits:        # traversal of List sequence
   print ('Current fruit :', fruit)

print ("Good bye!")
```

```
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : h
Current Letter : o
Current Letter : n

Current fruit : banana
Current fruit : apple
Current fruit : mango
Good bye!
```

Student Exercise

1. Write a function that computes the sum of the digits in an integer. Use the following function header

   def sumDigits(n):

   Write a test program that prompts the user to enter an integer and displays the sum of all its digits.

2. Write the following function to display three numbers in increasing order

   def displaySortedNumbers(num1, num2, num3):

   Write a test program that prompts the user to enter three numbers and invokes the function to display them in increasing order

3. Write the following function to display an integer in reverse order.

   def reverse(number):

   Write a test program that prompts the user to enter an integer and displays its reversal

4. Write the following function which returns True if the number is a palindrome.

   def isPalindrome(number):

   A number is a palindrome if its reversal is the same as itself. Write a test program that prompts the user to enter an integer and reports whether the integer is a palindrome.
   Hint: Use the **reverse** function to implement **isPalindrome**

5. Write a function that returns the number of days in a year using the following header

   def numberOfDaysInAYear(year):

   Write a test program that displays the number of days in the year from 2010 to 2020

6. Write a function that converts milliseconds to hours, minutes, and seconds using the following header

   def convertMilli(millis):

   The function returns three strings each for hours, minutes, and seconds. For example, convertMillis(5500) returns the strings ‘0’, ‘0’, and ‘5’, convertMillis(100000) returns the strings ‘0’, ‘1, ’40’, and convertMillis(555550000) returns the strings ‘154, ’19’, and ’10’

   Write a test program that prompts the user to enter a value for milliseconds and displays a string in the format of hours:minute:seconds