LAB # 1:

INTRODUCTION TO PYTHON

Objectives:

- Familiarize students with Python
- To teach students the conventional coding practices in Python

Hardware/Software required:

Hardware: Desktop/ Notebook Computer
Software Tool: Python (latest stable version)

Introduction:

Python is a high-level general-purpose programming language. Because code is automatically compiled to byte code and executed, Python is suitable for use as a scripting language, Web application implementation language, etc. Because Python can be extended in C and C++, Python can provide the speed needed for even compute intensive tasks.

Features of Python:

- Contains in-built sophisticated data structures like strings, lists, dictionaries, etc.
- The usual control structures: if, if-else, if-elif-else, while, plus a powerful collection iterator (for).
- Multiple levels of organizational structure: functions, classes, modules, and packages. They assist in organizing code. An excellent and large example is the Python standard library.
- Compile on the fly to byte code -- Source code is compiled to byte code without a separate compile step. Source code modules can also be "pre-compiled" to byte code files.
- Python is portable language, can run on different platforms such as Windows, Linux and Unix etc.
- It is Dynamically typed language i.e. Data Type of value is decided in run-time, not in advance.
- Python uses indentation to show block structure. Indent one level to show the beginning of a block. Out-dent one level to show the end of a block. As an example, the following C-style code:

```
int x = 10;
if x > 5
    {
     x = x+1;
     if x < 15
       {
           x = x-1;
       }
     }
printf('%d',x);
```

in Python would be:

```
x = 10;
if x > 5:
    x = x+1;
    if x < 15:
        x = x-1;
print(x)
```

And the convention is to use four spaces (and no tabs) for each level of indentation.

Interactive Python:

If you execute Python from the command line with no script, Python gives you an interactive prompt. This is an excellent facility for learning Python and for trying small snippets of code. Many of the examples that follow were developed using the Python interactive prompt.

In addition, there are tools that will give you a more powerful and fancy Python interactive mode. One example is IPython, which is available at http://ipython.scipy.org/. You may also want to consider using IDLE. IDLE is a graphical integrated development environment for Python. It contains a Python shell which helps you code in python by Auto-indent, Debugging and Color-coding your program

Lab Tasks:

1. Installing Python:

Download Python 3.6.2 for Windows from the following link:

https://www.python.org/downloads/

Install the downloaded executable file.

2. Opening IDLE

After installing Python, go to the start menu and search Python. Run the program labeled as Integrated Development Environment (IDLE).

3. Print "Hello, World!" on CLI

>>> print ("Hello, World!")

4. Mathematical Operations in Python:

>>> 1 + 1

2

>>> 6-5

1

```
>>> 2*5
    10
>>> 5**2
    25
>>> 21/3
    7
>>> 23/3
    7
>>> 23.0/3.0
    7.6666...
>>> 23%3
    2
```

5. Operators in Python

Command	Name	Example	Output
+	Addition	4+5	9
-	Subtraction	8-5	3
*	Multiplication	4*5	20
/	Division	19/3	6
%	Remainder	19%3	5
**	Exponent	2**4	16

6. Operator Precedence in Python:

Operators	Descriptions

()	Parentheses
**	Exponent (Raise to the power)
* / % //	Multiplication, Divide, Reminder, Floor Division
+ -	Addition, Subtraction
>> <<	Right and Left Bitwise Shift

7. Comments in Python:

>>> #Commented Code

8. Variables:

Variables are containers for storing data values. In Python, it is not needed to declare variables by specific data type. The Data type of a variable is decided at run-time.

```
>>> x = 5;
>>> print(type(x));
<class 'int'>
>>> y = 3;
>>> print("x plus y is",x+y)
x plus y is 8
>>> x = "WELCOME"
>>> print(type(x))
<class 'str'>
```

type () keyword tells the type of argument.

Variable Names

- Must start with underscore or with a letter.
- Cannot start with a number.
- Can contain underscores and alpha –numeric letters.
- Variables names are case-sensitive

9. Strings:

Strings are represented either by using double or single quotes.

Python Example:

```
word1 = "Good"
word2 = "Morning"
word3 = "to you too!"
print (word1, word2)
sentence = word1 + " " + word2 + " " +word3
print (sentence)
print(word1[2])
```

10. Boolean Operators in Python:

Boolean operators return either TRUE or FALSE

Expression	Function
<	less than
<=	less that or equal to
>	greater than
>=	greater than or equal to
!=	not equal to
<>	not equal to (alternate)
==	equal to

11. Conditional Statements:

'if - else' - Statement

```
a = 1
if a > 5:
   print ("This shouldn't happen.")    #condition true
else:
```

```
    print ("This should happen.")      #condition false
```

'elif' - Statement

```
z = 4

if z > 70:
    print ("Something is very wrong")
elif z < 7:
    print ("This is normal")
```

12. Input from user:

input(" "):

```
a = input ("Enter Value for variable a: ")
print (a)
```

input: Reads a string of text from user input.

Example:

```
name = input ("What's your name Lad? ")
print (name, "... what a nice name!")
```

Output:

What is your name Lad? Ali

Ali... what a nice name!

13. Indexes of String:

In Python strings are considered as arrays and characters in a string are numbered with indexes starting at 0:

Example:

```
name = "P. aishh"
```

Index	0	1	2	3	4	5	6	7
Character	P	.		a	i	s	h	h

Accessing an individual character of a string:

variableName [***index***]

Example:

print (name, "starts with", **name[0]**)

Output:

P. aishh starts with P

14. String Properties:

len(*string*) - number of characters in a string (including spaces)

str.lower(*string*) - lowercase version of a string

str.upper(*string*) - uppercase version of a string

Example:

```
name = "Linkin Park"
length = len(name)
big_name = str.upper(name)
print (big_name, "has", length, "characters")
```

Output:

LINKIN PARK has 11 characters

15. Strings and numbers:

ord(*text*) - converts a string into a number.

Example: ord('a') is 97, ord("b") is 98, ...

Characters map to numbers using standardized mappings such as *ASCII* and *Unicode*.

chr(*number*) - converts a number into a string.

Example: chr(99) is "c"

16. Loops in Python:

'while' loop

```
a = 0
while a < 10:
```

```
    a = a + 1
    print (a)
```

'for' loop

```
for i in range(1, 5):
    print (i)
for i in range(1, 5):
    print (i)
else:
    print ('The for loop is over')
```

range ():

If you do need to iterate over a sequence of numbers, the built-in function range() comes in handy. It generates lists containing arithmetic progressions:

>>> range(10) [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

It is possible to let the range start at another number, or to specify a different increment (even negative; sometimes this is called the 'step'):

```
>>> range(5, 10)
[5, 6, 7, 8, 9]
>>> range(0, 10, 3)
[0, 3, 6, 9]
>>> range(-10, -100, -30)
[-10, -40, -70]
```

Example:

```
for i in range(5):
    print(i)
```

17. Functions:

Define a Function?

```
def function_name(parameter_1, parameter_2):
    this statement is written within the function body
   return;
```

How to call a function?

function_name(parameters)

Code Example - Using a function

```
a = multiplybytwo(70)
```

The computer would see this:

```
a=140
```

Function Scope:

```
def changeme( mylist ):
   "This changes a passed list into this function"
   mylist = [1,2,3,4]; # This would assign new reference in mylist
   print ("Values inside the function: ", mylist)
   return
# Now you can call changeme function
mylist = [10,20,30];
changeme( mylist );
print ("Values outside the function: ", mylist)
```

18. Lists:

Lists are what they seem - a list of values. Each one of them is numbered, starting from zero. You can remove values from the list and add new values to the end. Example: Your many cats' names. *Compound* data types used to group together other values. The most versatile is the *list*, which can be written as a list of comma-separated values (items) between square brackets. List items need not all have the same type.

```
cats = ['Tom', 'Snappy', 'Kitty', 'Jessie', 'Chester']

print cats[2]
cats.append('Catherine')
```

```
#Remove your 2nd cat, Snappy. Woe is you.
del cats[1]
```

19. Compound datatype:

```
>>> a = ['spam', 'eggs', 100, 1234]

>>> a[1:-1]
        ['eggs', 100]
>>> a[:2] + ['bacon', 2*2]
        ['spam', 'eggs', 'bacon', 4]
>>> 3*a[:3] + ['Boo!']
        ['spam', 'eggs', 100, 'spam', 'eggs', 100, 'spam', 'eggs', 100, 'Boo!']
>>> a= ['spam', 'eggs', 100, 1234]
>>> a[2] = a[2] + 23
>>> a
        ['spam', 'eggs', 123, 1234]
```

20. Replace some items:

```
>>> a[0:2] = [1, 12]
>>> a
        [1, 12, 123, 1234]
```

21. Remove some items:

```
>>> a[0:2] = []
>>> a
        [123, 1234]
```

22. Clear the list: replace all items with an empty list:

```
>>> a[:] = []
>>> a
        []
```

23. Length of list:

```
>>> a = ['a', 'b', 'c', 'd']
>>> len(a)
        4
```

24. Nest lists:

```
>>> q = [2, 3]
>>> p = [1, q, 4]
>>> len(p)
        3
>>> p[1]
        [2, 3]
```

25. Functions of lists:

list.append(x): Add an item to the end of the list; equivalent to a[len(a):] = [x].

Example:

```
my_list = [1, 2, 3]

my_list.append(4)

print(my_list)  # Output: [1, 2, 3, 4]
```

list.extend(L): Extend the list by appending all the items in the given list, equivalent to a[len(a):] = L.

Example:

```
my_list = [1, 2, 3]

my_list.extend([4, 5, 6])

print(my_list)  # Output: [1, 2, 3, 4, 5, 6]
```

list.insert(i, x): Insert an item at a given position. The first argument is the index of the element before which to insert, so a.insert(0, x) inserts at the front of the list, and a.insert(len(a), x) is equivalent to a.append(x).

Example:

```
my_list = [1, 2, 3]

my_list.insert(1, 10)  # Insert 10 at index 1

print(my_list)  # Output: [1, 10, 2, 3]
```

list.remove(x): Remove the first item from the list whose value is x. It is an error if there is no such item.

Example:

```
my_list = [1, 2, 3, 2]

my_list.remove(2)  # Removes the first occurrence of 2

print(my_list)  # Output: [1, 3, 2]
```

list.pop([i]): Remove the item at the given position in the list, and return it. If no index is specified, a.pop() removes and returns the last item in the list.

Example:

```
my_list = [1, 2, 3, 4]

popped_item = my_list.pop(1)  # Remove and return item at index 1

print(popped_item)  # Output: 2

print(my_list)  # Output: [1, 3, 4]


# Without an index

popped_item = my_list.pop()  # Removes and returns the last item

print(popped_item)  # Output: 4

print(my_list)  # Output: [1, 3]
```

list.count(x): Return the number of times x appears in the list.

```
my_list = [1, 2, 2, 3, 2]

count = my_list.count(2)

print(count)  # Output: 3
```

list.sort(): Sort the items of the list, in place.

```
my_list = [1, 2, 3, 4]

my_list.reverse()

print(my_list)  # Output: [4, 3, 2, 1]
```

list.reverse(): Reverse the elements of the list, in place.

```
my_list = [1, 2, 3, 4]

my_list.reverse()

print(my_list)  # Output: [4, 3, 2, 1]
```

26. Tuples:

Tuples are just like lists, but you **can't** change their values. Again, each value is numbered starting from zero, for easy reference.

Example:

```
my_tuple = (1, 2, 3) #where in list we add values in [] brackets

# my_tuple[0] = 10  # This will raise an error

print(my_tuple)  # Output: (1, 2, 3)
```

Example: the names of the months of the year.

months = ('January' , 'February', 'March', 'April', 'May', 'June', 'July', 'August', 'September', 'October', 'November', ' December')

index	Value
0	Jan
1	Feb
2	Mar
3	April
4	May
5	Jun
6	Jul
7	Aug
8	Sep
9	Oct
10	Nov
11	Dec

```
>>> basket = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']

>>> fruit = set(basket)  # create a set without duplicates

>>> fruit

        set(['orange', 'pear', 'apple', 'banana'])

>>> 'orange' in fruit   # fast membership testing

      True

>>> 'crabgrass' in fruit
```

False

27. Dictionaries:

Dictionaries are like what their name suggests - a dictionary. In a dictionary, you have an 'index' of words, and for each of them a definition.

In python, the word is called a 'key', and the definition a 'value'. The values in a dictionary are not numbered - they are not in any specific order, either - the key does the same thing. You can add, remove, and modify the values in dictionaries.

Example: telephone book.

```
phonebook = { 'ali':8806336, 'omer':6784346,'shoaib':7658344, 'saad':1122345 }

  #Add the person '' to the phonebook:
  phonebook['waqas'] = 1234567

  # Remove the person '' to the phonebook:
  del phonebook['shoaib']

phonebook = {'Andrew Parson':8806336, \
  'Emily Everett':6784346, 'Peter Power':7658344, \
  'Lewis Lame':1122345}

  #Add the person 'Gingerbread Man' to the phonebook:
  phonebook['Gingerbread Man'] = 1234567

  #Delete the person 'Gingerbread Man' to the phonebook:
  del phonebook['Andrew Parson']
```

LAB EVALUATION:

Q1: Write a simple calculator program. Follow the steps below:

Declare and define a function name “Menu” which displays a list of choices for user such as addition, subtraction, multiplication etc. It takes the choice from user as an input and return.

Define and declare a separate function for each choice.

In the main body of the program call respective function depending on user’s choice.

Program should not terminate till user chooses last option that is “Quit”.

Q2: Write a method to calculate factorial of a number entered by the user.

Q3: Write a program that lets the user enter in some English text, then converts the text to Pig-Latin.

To review, Pig-Latin takes the first letter of a word, puts it at the end, and appends "ay". The only exception is if the first letter is a vowel, in which case we keep it as it is and append "hay" to the end.

E.g. "hello" -> "ellohay", and "image" -> "imagehay"

Hint: *Split the entered string through split() method and then iterate over the resultant list, e.g. "My name is John Smith".split(" ") -> ["My", "name", "is", "John", "Smith"]*

NOTE: A lab report is expected to be submitted for each lab. (A simple PDF will be sufficient).