

**LAB # 1:****INTRODUCTION TO PYTHON****Objectives:**

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

**Hardware/Software required:**

Hardware: Desktop/ Notebook Computer

Software Tool: Python 3.6.2

**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 computing 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 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:

```
if (x)
{
    if (y)
    {
        f1();
    }
    f2();
}
```

in Python would be:

```
if x:
    if y:
        f1()
    f2()
```

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. You will find a script to start up IDLE in the Tools/scripts directory of your Python distribution. IDLE requires Tkinter.

### Lab Tasks:

#### 1. Installing Python:

Download Python 3.6.2 for Windows from the following link:

[www.python.org](http://www.python.org)

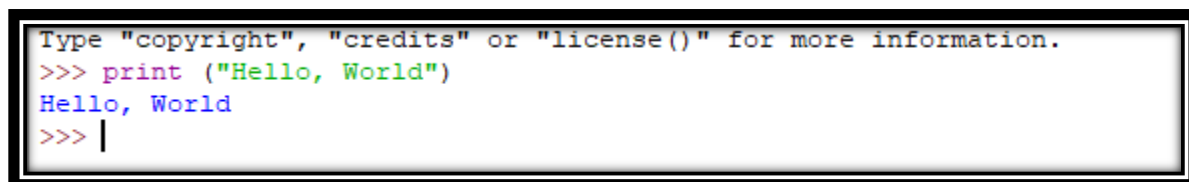
Install the downloaded executable file.

#### 2. Opening IDLE

After installing Python, go to the start menu and search Python. Run the program labeled as Integrated Development Environment (IDLE) **IDLE (Python 3.6 32-bit)**

#### 3. Print "Hello, World!" on CLI

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

A screenshot of a Python interactive shell window. The window has a title bar and a text area. The text area contains the following text: "Type 'copyright', 'credits' or 'license()' for more information." followed by a prompt ">>> print ('Hello, World')", the output "Hello, World", and a new prompt ">>> |".

```
Type "copyright", "credits" or "license()" for more information.
>>> print ("Hello, World")
Hello, World
>>> |
```

#### 4. Mathematical Operations in Python:

- ADDITION

```
>>> 1 + 1
```

```
2
```

- SUBTRACTION

```
>>> 6-5
```

```
1
```

- MULTIPLICATION

```
>>> 2*5
```

```
10
```

- EXPONENT

```
>>> 5**2
```

```
25
```

- DIVISION

By default the data type is float however you can change the data type as well.

```
>>> 21/3
```

```
7.0
```

```
>>> 23/3
```

```
7.6666
```

```
>>> 21/3
7.0
>>> 23/3
7.666666666666667
>>> int (23/3)
7
>>> |
```

- REMAINDER

```
>>> 23%3
```

```
2
```

## 5. Operators in Python

Course Instructor: Dr. Arslan Shaukat

Lab Instructor: Tahira Iqbal

NUST College of E&ME

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:

- parentheses ()
- exponents \*\*
- multiplication \*, division \, and remainder %
- addition + and subtraction -

Note: When an expression has operators of same precedence, output is calculated on the basis of associativity. Mostly operators have left-right associativity, however Exponent has right-left associativity.

```
>>> 10-4*2
2
>>> #Now when we put paranthesis output will be changed
>>> (10-4)*2
12
>>> (10/5*3)
6.0
>>> #as multiplication, division and remainder have same level of precedence
>>> #so output will be calculated from Left to right (left-right associativity)
>>> #Now if we put a paranthesis, output will be changed
>>> (10/(5*3))
0.6666666666666666
>>> #exponent right-left associativity
>>> 2**3**2 # 3**2=9----> 2**9=512
512
>>> (2**3)**2 #paranthesis will change the output
64
>>> |
```

## 7. Comments in Python:

```
>>> #Commented Code
```

## 8. Variables:

Rules for variable names:

- names can not start with a number
- names can not contain spaces, use \_ instead
- names can not contain any of these symbols:

```
:'",<>/?|\!@#%^&*~--+
```

- it's considered best practice ([PEP8](#)) that names are lowercase with underscores
- avoid using Python built-in keywords like list and str

```
>>> a=5
>>> a
5
>>> print ("The value of a is", a)
The value of a is 5
>>> print (a*5)
25
>>> #original value of a will still be 5
>>> a
5
>>> a= "Elif"
>>> age = 30
>>> print (a, "is", age, "years old")
Elif is 30 years old
>>> |
```

You can check the type of the variable using command ***type(variable name)***

```
>>> type(a)
<class 'str'>
>>> type (age)
<class 'int'>
>>> |
```

## 9. Strings:

Strings are actually sequence of letters/elements , we can grab a letter out of string using indexing

```
>>> 'hello everyone'
'hello everyone'
>>> "hello everyone"
'hello everyone'
>>> print ("hello")
hello
>>> 'Hello 'Elif''
SyntaxError: invalid syntax
>>> #ABOVE STATEMENT GAVE ERROR BECUASE
>>> #YOU CAN'T USE THE SAME TYPE OF QUOTATIONS
>>> #TWICE IN A LINE
>>> # YOU CAN USE THE DOUBLE AND SINGLE QUOTES
>>> "Hello 'Elif'"
"Hello 'Elif'"
>>> |
```

```
>>> #TO PRINT ON A NEW LINE USE \n
>>> print ("Hello \n Everyone")
Hello
Everyone
>>> |
```

The length of the string can be checked using the *len* keyword. It counts all the characters including spaces

```
>>> my_string= "Hello World!!"
>>> len(my_string)
13
>>> |
```

### Indexing in string

```

>>> #to grab first letter
>>> a[0]
'H'
>>> a[-1]
'd'
>>> #starting from -1,-2... will grab the letters from the end of the string
>>> a[2:] #this will return the whole string execept the 0th and 1st letter
'llo World'
>>> a[:3] #grab the letter UPTO 3rd index
'Hel'
>>> a[::2] #grab everything while going in steps of 2(skip 2-1=1 letter)
'HloWrld'
>>> #to reverse a string
>>> a[::-1]
'dlroW olleH'
>>> |

```

Once the string is created you can't change its any of the letter however you can concatenate other stings with it

```

>>> w1="good"
>>> w2="morning"
>>> w3="people"
>>> w1+w2+w3
'goodmorningpeople'
>>> |

```

### 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

**10. Boolean Operators in Python:**

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

OPERATOR	DESCRIPTION	SYNTAX
and	Logical AND: True if both the operands are true	x and y
or	Logical OR: True if either of the operands is true	x or y
not	Logical NOT: True if operand is false	not x

**11. Conditional Statements:****'if' - Statement**

```

y=2
if y>1:
    print ("value of y is", y)

```

This will print value of y is 2

**'if - else' - Statement**

Course Instructor: Dr. Arslan Shaukat

Lab Instructor: Tahira Iqbal

NUST College of E&ME



```
x=5
if (x==5):
    print("true")
else:
    print("false")
|
```

This will print "True" as output

### 'elif' - Statement

```
name= "John"
if (name=="Elif"):
    print ("Hello Elif")
elif (name=="John"):
    print ("Hello John")
else:
    print ("Hello, What is your name")
|
```

This will print Hello John

### Another example

z = 4

if z > 70:

    print ("Something is very wrong")

elif z < 7:

    print ("This is normal")

## 12. Input from user:

To get input from the user , use keyword input

```
>>> a= input ("What is your name")
What is your name Elif
>>> print (a)
Elif
>>> |
```

### 13. 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"

### 14. Loops in Python:

#### 'while' loop

```
x=0

while (x<2):
    print ("value of x is", x)
    x=x+1;
else:
    print("loop completed")
|
```

#### 'for' loop

```
for i in range(1,5): #i is variable
    print (i)
else:
    print ("loop complete")
|
```

## 15. Functions:

### How to call a function?

function\_name(parameters)

Code Example - Using a function

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

The computer would actually see this:

```
a=140
```

### Define a Function?

Function allow us to call the same statements over and over again by just one command.

Some functions are built-in

e.g range()

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]
```

**Defining a new function:**

```
def function_name(parameter_1, parameter_2):
```

    this statement is written within the function body

```
return;
```

```
def greet(name):  
    print("Hello", name)  
|
```

```
RESTART: E7/A  
>>> greet('elif')  
Hello elif  
>>> |
```

Using RETURN allow to store the result in other variable so it can be used in future

```
def add_num(n1,n2):  
    return n1+n2
```

```
>>> add_num(4,5)  
9  
>>> s= add_num(4,5)  
>>> print(s)  
9  
>>> |
```

**16. 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. 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.

Course Instructor: Dr. Arslan Shaukat

Lab Instructor: Tahira Iqbal

NUST College of E&ME

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

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

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

```
>>> a= [1,2,3,]
>>> a
[1, 2, 3]
>>> a.append('hello')
>>> a
[1, 2, 3, 'hello']
```

Using + sign wont make the changes permanent

```
>>> a+ ['5', '6']
[1, 2, 3, 'hello', '5', '6']
>>> a
[1, 2, 3, 'hello']
>>> |
```

## 17. Compound datatype:

In python indexing start from 0

```
>>> a = ['hello', '1','2', 'hi']
>>> a[0]
'hello'
>>> a[:3]    #grab UPTO 3rd element
['hello', '1', '2']
>>> 2* a[:3]  #repeat twice
['hello', '1', '2', 'hello', '1', '2']
>>> a[1]= 'world'
>>> a
['hello', 'world', '2', 'hi']
>>> |
```

## 18. Replace some items:

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

Course Instructor: Dr. Arslan Shaukat

Lab Instructor: Tahira Iqbal

NUST College of E&ME

**19. Remove some items:**

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

**20. Clear the list: replace all items with an empty list:**

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

**21. Length of list:**

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

**22. Nest lists:**

```
>>> q = [2, 3]
>>> p = ['1', q, '4']
>>> len(p)
3 #length will be 3 however on printing p , ['1', [2, 3], '4']
>>> p[1]
[2, 3]
```

**23. Functions of lists:**

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

If we pass another list in `append(list)` then the new list will be shown as 1 element

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

**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)`.

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

**list.pop([i]):** Remove the item at the given position in the list, and return it e.g `a.pop(0)` removes the 1<sup>st</sup> item. If no index is specified, `a.pop()` removes and returns the last item in the list.

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

Course Instructor: Dr. Arslan Shaukat

Lab Instructor: Tahira Iqbal

NUST College of E&ME

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

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

## 24. Tuples:

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

**NOTE: Tuples are used when we need to make sure that the elements must not change. Otherwise we mostly use lists**

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
```

*SO 'ORANGE' is mentioned once in the variable fruit.*

```
>>> fruit
```

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

Course Instructor: Dr. Arslan Shaukat

Lab Instructor: Tahira Iqbal

NUST College of E&ME

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

```
True
```

```
>>> 'crabgrass' in fruit
```

```
False
```

## 24. Dictionaries:

Dictionaries are similar to hash tables. In lists, the values are stored and accessed by their index however dictionaries are more manageable as the values/elements are now stored by a **key**

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

```
>>> dct= {'name': 'elif', 'age': '30'}
>>> dct['age']
'30'
```

Dictionary values can be of any data type.

```
>>> my_dct = {'key1':123,'key2':[12,23,33],'key3':['item0','item1','item2']}
```

A new value can be added to the dictionary

```
>>> dct['res']= 'uk'
>>> dct
{'name': 'elif', 'age': '30', 'res': 'uk'}
>>> |
```

Any value can be deleted by using its 'key'

```
>>> del dct['age']
>>> dct
{'name': 'elif', 'res': 'uk'}
>>> |
```



**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 Fibonacci series up to ‘n’ points. After calculating the series, the method should return it to main.****Q3: Write a method to calculate factorial of a number entered by the user.****Q4: 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”]

**Conclusion:**

In this lab, you learned the basics of Python and its important features.

**NOTE:** A lab journal is expected to be submitted for each lab.