NOIDA INSTITUTE OF ENGINEERING AND TECHNOLOGY

GREATER NOIDA

SUMMER TRAINING REPORT

ON

"PYTHON"

Submitted in fulfillment of the requirement

For the award of degree

BACHELOR OF TECHNOLOGY

SESSION (2018-2022)

SUBMITTED TO:

Mr. Vikash Mishra

Ms. Chitvan Gupta

Ms. Smiley Ganghi

SUBMITTED BY:

YOGESH KUMAR YADAV

1813310246

INDEX

PREFACE

The objective of a practical training is to learn something about industries practically and to be familiar with a working style of a technical worker to adjust simply according to industrial Environment. As a part of academic syllabus of four year degree course in ITE, every student is required to undergo practical training of 2 weeks. I am a student of 2nd year ITE and this report is written on the basis of practical knowledge acquired during the period of practical training taken at *"Noida Institute of Engineering And Technology, Gr. Noida"*. This report deals with the equipment their relation and their general operating principle. Sincere efforts have been made to present this report on Python – A Programming and Scripting Language.

The objective of a practical training is to learn something about industries practically and to be familiar with a working style of a technical worker to adjust simply according to industrial environment. This report deals with the equipment's their relation and their general operating principle. Python, an interpreted language which was developed by ***Guido van Rossum*** came into implementation in 1989. The language supports both object oriented and procedure oriented approach. Python is designed to be a highly extensible language. Python works on the principle of "there is only one obvious way to do a task" rather than "there is more than one way to solve a particular problem". Python is very easy to learn and implement. The simpler syntax, uncomplicated semantics and approach with which Python has been developed makes it very easier to learn. A large number of python implementations and extensions have been developed since its inception.

ACKNOWLEDGEMENT

It is my pleasure to be indebted to various people, who directly or indirectly contributed in the development of this work and who influenced my thinking, behavior, and acts during the course of study.

I express my sincere gratitude to **Mr. Raman Batra** worthy Executive Vice President for providing me an opportunity to undergo summer training at NIET.

I am thankful to all the members for their support, cooperation, and motivation provided me during the training for constant inspiration presence and blessings.

Lastly, I would like to thank the almighty and my parents for their moral support and my friends with whom I shared my day-to-day experience and received lots of suggestions that improved my quality of work.

History of Python

Python was developed in1980 by Guido Van Rossum at the National Research Institute for Mathematics and Computer Science in the Netherlands as a successor of ABC language capable of exception handling and interfacing. Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

Van Rossum picked the name Python for the new language from a T.V. show, Monty Python's Flying Circus.

In December 1989 the creator developed the 1st python interpreter as a hobby and then on 16th October 2000, Python 2.0 was released with many new features.

In December 1989, I was looking for a "hobby" programming project that would keep me occupied during the week around Christmas. My office … would be closed, but I had a home computer, and not much else on my hands. I decided to write an interpreter for the new scripting language I had been thinking about lately: a descendant of ABC that would appeal to Unix/C hackers. I choose Python as a working title for the project, being a slightly irreverent mood (and a big fan of Monty Python's Flying Circus).

Python Versions

- Python 1.0 – January 1994
- Python 1.5 – December 31 ,1997
- Python 1.6 – September 5, 2000
- Python 2.0 – October 16, 2000
- Python 2.1 – April 17, 2001
- Python 2.2 – December 21,2001
- Python 2.3 – July 29,2003
- Python 2.4 – November 30, 2004
- Python 2.5 – September 19, 2006
- Python 2.6 – October 1,2008
- Python 2.7 – July 3,2010
- Python 3.0 – December 3, 2008
- Python 3.1 – June 27, 2010
- Python 3.2 – February 20, 2011
- Python 3.3 – September 29, 2012
- Python 3.4 –March 16, 2014
- Python 3.5 – September 13, 2015
- Python 3.6 – December 23, 2016
- Python 3.7 – June 27, 2018

Why Python?

The language's core philosophy is summarized in the document The Zen Of Python (PEP 20), which includes aphorisms such as…..

- Beautiful is better than ugly
- Simple is better than complex
- Complex is better than complicated
- Readability counts
- Explicit is better than implicit

Characteristics of Python

Interpreted Language: Python is processed at runtime by Python Interpreter.

Easy to read: Python source-code is clearly defined and visible to the eyes.

Portable: Python codes can be run on a wide variety of hardware platforms having the same interface.

Object-Oriented Language: It supports object-oriented features and the techniques of programming.

Interactive Programming Language: Users can interact with the python interpreter directly for writing programs.

Easy Language: Python is easy to learn especially for beginners.

Straight forward Syntax: The formation of Python syntax is simple and straightforward which also makes it popular.

Scalable: Python provides an improved structure for supporting large programs than shell-scripts.

Extendable: Users can add low level modules to Python interpreter.

BASIC OPERATORS IN PYTHON

1. **Arithmetic operators:** Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication and division

Operator	Meaning	Example
+	Add two operands or unary plus	x + y +2
-	Subtract right operand from the left or unary minus	x - y -2
*	Multiply two operands	x * y
/	Divide left operand by the right one (always results into float)	x / y
%	Modulus - remainder of the division of left operand by the right	x % y (remainder of x/y)
//	Floor division - division that results into whole number adjusted to the left in the number line	x // y
**	Exponent - left operand raised to the power of right	x**y (x to the power y)

2. **Relational Operators:** Relational operators compares the values. It either returns true or false according to the condition.

Operator	Use	Description
>	op1 > op2	Returns true if op1 is greater than op2
>=	op1 >= op2	Returns true if op1 is greater than or equal to op2
<	op1 < op2	Returns true if op1 is less than op2
<=	op1 <= op2	Returns true if op1 is less than or equal to op2
==	op1 == op2	Returns true if op1 and op2 are equal
!=	op1 != op2	Returns true if op1 and op2 are not equal

3. **Logical operators:** Logical operators perform **Logical AND**, **Logical OR** and **Logical NOT** operations.

Operator	Meaning	Example
and	True if both the operands are true	x and y
or	True if either of the operands is true	x or y
not	True if operand is false (complements the operand)	not x

4. **Bitwise operators:** Bitwise operators acts on bits and performs bit by bit operation.

Bitwise operators in Python

Operator	Meaning	Example
&	Bitwise AND	x& y = 0 (0000 0000)
|	Bitwise OR	x | y = 14 (0000 1110)
~	Bitwise NOT	~x = -11 (1111 0101)
^	Bitwise XOR	x ^ y = 14 (0000 1110)
>>	Bitwise right shift	x>> 2 = 2 (0000 0010)
<<	Bitwise left shift	x<< 2 = 40 (0010 1000)

5. **Assignment operators:** Assignment operators are used to assign values to the variables.

6. **Special operators:** There are some special type of operators like-

   a. **Identity operators-**

   is and **is not** are the identity operators both are used to check if two values are located on the same part of the memory. Two variables that
   Are equal does not imply that they are identical.

   b. **Membership operators-**

   in and **not in** are the membership operators; used to test whether a Value or variable is in a sequence.

PYTHON DATA STRUCTURE

1. List

The list is a 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 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. *For example-*
list1= {'physics', 'chemistry', 1997, 2000};
list2= {1, 2, 3, 4, 5};
list3= {"a", "b", "c", "d"};

Similar to string indices, list indices start at 0, and lists can be sliced, concatenated and so on.

Accessing Values in Lists
To access values in lists, use the square brackets for slicing along with the index or indices to obtain value available at that index. *For example-*
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]

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

Update
```
list={'physics','chemistry',1997,2000};
print"Value available at index 2:"
print list[2]
list[2]=2001;
print"New value available at index 2 :"
print list[2]
```

Output
Value available at index 2 : 1997
New value available at index 2 : 2001

Delete
```
list1={'physics','chemistry',1997,2000};
print list1
del list[2];
print"After deleting value at index 2";
[physics','chemistry',2000]
```

2. Basic List of Operations

Python Expression	Results	Description
len([1, 2, 3])	3	Length
[1, 2, 3] + [4, 5, 6]	[1, 2, 3, 4, 5, 6]	Concatenation
['Hi!'] * 4	['Hi!', 'Hi!', 'Hi!', 'Hi!']	Repetition
3 in [1, 2, 3]	True	Membership
for x in [1, 2, 3]: print x,	1 2 3	Iteration

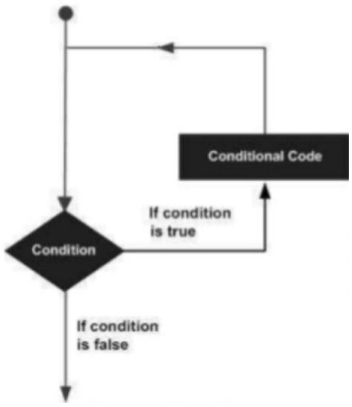
3. List Functions with Description

SN	Methods with Description
1	list.append(obj) Appends object obj to list
2	list.count(obj) Returns count of how many times obj occurs in list
3	list.extend(seq) Appends the contents of seq to list
4	list.index(obj) Returns the lowest index in list that obj appears
5	list.insert(index, obj) Inserts object obj into list at offset index
6	list.pop(obj=list[-1]) Removes and returns last object or obj from list
7	list.remove(obj) Removes object obj from list
8	list.reverse() Reverses objects of list in place
9	list.sort([func]) Sorts objects of list, use compare func if given

Loops & Conditional Statements Loops

Programming language provide various control structure that allow for more complicated execution paths.
A loop statement allows us to execute a statement or group of student's multiple times. The following diagram illustrates a loop statement.

=



1. Python programming language provides following types of loops to handle looping requirements:

Loop Type	Description
while loop	Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body.
for loop	Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
nested loops	You can use one or more loop inside any another while, for or do..while loop.

Examples: -

- For Loop

```
>>>for mynum in [1,2,3,4,5]:
        print “Hello” mynum

Hello 1
Hello 2
Hello 3
Hello 4
Hello 5
```

Print each fruit in a fruit list:

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
print(x)
```

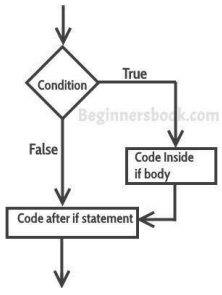
While Loop:

```
>>>count=0
>>>while(count<4):
print "The count is:",count
count=count+1
The count is: 0
The count is: 1
The count is: 2
The count is: 3
```

2. Conditional Statements:

Decision making is anticipation of conditions occurring while execution of the program and specifying actions taken according to the conditions.

Decision structures evaluate multiple expressions which produce *TRUE* or *FALSE* As outcome. You need to determine which action to take and which statements to execute if outcome is true or false otherwise.



Python programming language provides following types of decision-making statements.

Statement	Description
if statements	An **if statement** consists of a boolean expression followed by one or more statements.
if...else statements	An **if statement** can be followed by an optional **else statement**, which executes when the boolean expression is FALSE.
nested if statements	You can use one **if** or **else if** statement inside another **if** or **else if** statement(s).

Examples

If statement:

```
>>>state = 'Texas'
>>>if state == 'Texas':
print "TX"
TX
```

If-else statement:

```
>>>if state == 'Texas':
print "TX"
else:
print "(Inferior state)"
```

if...else...if statement:

```
>>>if name == "John":
        print ("Hi John")
    elif name== "Walker":
        print ("Hi Walker")
    else:
        print("Imposter")
```

The Break Statement

With the break statement we can stop the loop before it has looped through all the items:

Example

```
Exit the loop when x is "banana":
fruits = ["apple", "banana", "cherry"]
for x in fruits:
print(x)
if x == "banana":
break
```

The Continue Statement

With the continue statement we can stop the current iteration of the loop, and continue with the next:

Example

```
Do not print banana:
fruits = ["apple", "banana", "cherry"]
for x in fruits:
if x == "banana":
continue
print(x)
```

Arrays

Arrays are used to store multiple values in one single variable:

Example

Create an array containing car names:
cars = ["Ford", "Volvo", "BMW"]
An array is a special variable, which can hold more than one value at a time.
If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:
car1 = "Ford"
car2 = "Volvo"
car3 = "BMW"
However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?
The solution is an array!
An array can hold many values under a single name, and you can access the values by referring to an index number.

Access the Elements of an Array

You refer to an array element by referring to the index number.
Example
Get the value of the first array item:
x = cars [0]

The Length of an Array

Use the len () method to return the length of an array (the number of elements in an array).
Example:
Return the number of elements in the cars array:
x = len(cars)

Looping Array Elements

You can use the for in loop to loop through all the elements of an array.
Example
Print each item in the cars array:
for x in cars:
print(x)

Adding Array Elements

You can use the append () method to add an element to an array.
Example
Add one more element to the cars array:
cars.append("Honda")

Removing Array Elements

You can also use the remove () method to remove an element from the array.
Example
Delete the element that has the value "Volvo":
cars.remove("Volvo")

1. Array Methods

Python has a set of built-in methods that you can use on lists/arrays.

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the first item with the specified value

2. Processing of Array

```
arr=array('i',[8,9,6,5,3,2,4,3])
print("\n Let's say array is:",arr)
arr.append(21)
print("\n arr.append(21):",arr)
arr.insert(1,111)
print("\n arr.insert(1,111):",arr)
print("\n are.count(3):",arr.count(3))
print("\n arr.index(5):",arr.index(5))
arr.pop()
print("\n arr.arr.pop():",arr)
arr.remove(6)
print("\n arr.remove(6):",arr)
arr2=arr
arr2.reverse()
print("\n arr.reverse():",arr2)
lst=arr.tolist()
print("\n list from array:",lst)
# strings=str(arr.tostring())
# print("\n String from Array:",strings)
```

Dictionary

A dictionary is a collection which is unordered, changeable and indexed. In Python dictionaries are written with curly brackets, and they have keys and values.

Example

Create and print a dictionary:

```
thisdict = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}
print(thisdict)
```

1. Basic operations on dictionary

- **Accessing Items**

You can access the items of a dictionary by referring to its key name, inside square brackets:
Example
Get the value of the "model" key:
x = thisdict["model"]

- **Change Values**

You can change the value of a specific item by referring to its key name:
Example
Change the "year" to 2018:
thisdict = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}
thisdict["year"] = 2018

- **Loop Through a Dictionary**

You can loop through a dictionary by using a for loop.
When looping through a dictionary, the return value are the keys of the dictionary, but there are methods to return the values as well.
Example
Print all key names in the dictionary, one by one:
for x in thisdict:
print(x)

- **Check if Key Exists**

To determine if a specified key is present in a dictionary use the in keyword:
Example
Check if "model" is present in the dictionary:
thisdict = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}
if "model" in thisdict:
print("Yes, 'model' is one of the keys in the thisdict dictionary")

- **Dictionary Length**

To determine how many items (key-value pairs) a dictionary has, use the len() method.
Example
Print the number of items in the dictionary:
print(len(thisdict))

- **Adding Items**

Adding an item to the dictionary is done by using a new index key and assigning a value to it:
Example
thisdict = {
"brand": "Ford",
"model": "Mustang",

```
"year": 1964
}
thisdict["color"] = "red"
print(thisdict)
```

- **Removing Items**

There are several methods to remove items from a dictionary:

Example

The pop() method removes the item with the specified key name:

```
thisdict = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}
thisdict.pop("model")
print(thisdict)
```

- **Copy a Dictionary**

You cannot copy a dictionary simply by typing dict2 = dict1, because: dict2 will only be a reference to dict1, and changes made in dict1 will automatically also be made in dict2.

There are ways to make a copy, one way is to use the built-in Dictionary method copy().

Example

Make a copy of a dictionary with the copy() method:

```
thisdict = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}
mydict = thisdict.copy()
print(mydict)
```

The dict() Constructor

It is also possible to use the dict() constructor to make a new dictionary:

Example

```
thisdict = dict(brand="Ford", model="Mustang", year=1964)
# note that keywords are not string literals
# note the use of equals rather than colon for the assignment
print(thisdict)
```

2. Dictionary Methods

Python has a set of built-in methods that you can use on dictionaries.

Method	Description
clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and values
get()	Returns the value of the specified key
items()	Returns a list containing the a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary

File Handling in Python

Python too supports file handling and allows users to handle files i.e., to read and write files, along with other many file handling options, to operate on files. The concept of file handling has stretched over various other languages, but the implementation is either complicated or lengthy. Each line of code includes a sequence of characters and they form text file. Each line of a file is terminated with a special character, called EOL or End of Line characters like comma{,} or newline character. It ends the current line and tells the interpreter a new one has begun.

We use open () f unction in Python to open a file in read or write mode. As explained above, open () will return a file object. To return a file object we use **open ()** function along with two arguments, that accepts file name and the mode, whether to read or write. So the syntax being: **open (filename, mode)**.

There are three kinds of mode, that Python provides and how files can be opened-

1. **"r"**, for reading.

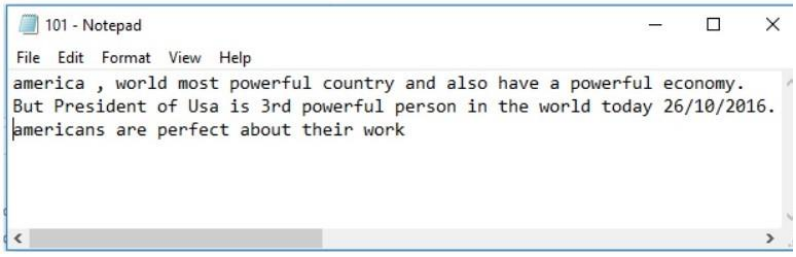
2**. "w"**, for writing.

3**. "a"**, for appending.

4. **"r+"**, for both reading and writing.

Example:

Ex-It is a notepad file (101.txt)




```
dic={}
words=[]

with open("101.txt")as f1:
    for line in f1:
        words=words+line.split()

for i in range(len(words)):
    count=0
    for j in range(len(words)):
        if words[i]==words[j]:
            count=count+1
            dic[words[i]]=count

for count in dic:
    print(count+" "+str(dic[count]))
    f4=open("105.txt","a+")
    f4.writelines(count+" "+str(dic[count]))
```

It reads the words from 101.txt file and print the words which are

Present in the file and also tell that word occurring how many times.

Numpy Function

NumPy is a Pyhton package. It stands for 'Numemrical Python'. It is a library consisting of multidimensional array objects and a collection of routlines for processing of array.

Numeric, the ancestor of NumPy, was developed by Jim Hugunin. Another package Numarray was also developed, having some additional functionalities. In 2005, Travis Oliphant created NumPy package by incorporating the features of Numarray into numeric package. There are many contributors to this open source project.

1. Operations using NumPy

Using NumPy, a developer can perform the following operations –

1. Mathematical and logical operations on arrays.
2. Fourier transforms and routines for shape manipulation.
3. Operations related to linear algebra.

   NumPy has Built-in functions for linear algebra and random number generation.

2. Simple program to create a Matrix

First of all, we import "NumPy" package then using this we take input in NumPy function as a list then we create a matrix

```
: import numpy as np
  b=np.array([[1,6,5,2,3,45]])
  b.shape=(3,2)
  print(b)

[[ 1  6]
 [ 5  2]
 [ 3 45]]
```

There is many more function that can be performed by using this like that take sin value of given value, print a zerotrix etc, we also take any image in the form of array.

```
]: Z = np.zeros((8,8),dtype=int)
   print(Z)

[[0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]]

]: import numpy as np
   a = np.array([0,30,45,60,90])

   print ('Sine of different angles:' )
   # Convert to radians by multiplying with pi/180
   print (np.sin(a*np.pi/180) )
   print ('\n')

   print ('Cosine values for angles in array:')
   print (np.cos(a*np.pi/180) )
   print ('\n')

Sine of different angles:
[0.         0.5        0.70710678 0.8660254  1.        ]


Cosine values for angles in array:
[1.00000000e+00 8.66025404e-01 7.07106781e-01 5.00000000e-01
 6.12323400e-17]
```

Applications of Python

1. Audio/Video Applications
2. Console Applications
3. Image Applications
4. Mobile Applications
5. Science and Education Applications

Scope of Python

- Science

  -Bioinformatics

- System Administration

  -Unix

  -Web logic

  -Web sphere

- Web Application Development

  -CGI

Testing Scripts

What Can We Do With Python

1. System Programming
2. Graphical User Interface Programming
3. Internet Scripting
4. Component Integration
5. Database Programming
6. Gaming, Images, Robot,etc.

Who Uses Python Today?

Data Scientists - probably the largest portion of python community

2. Game Developers - (pygame, …)

3. Educators - since it is readable and easy to learn

4. Network Admins/Systems Administrators/Security Testers - makes a good alternative to bash scripting.

5. Web Developers - Django, flask, etc…

6. IOT Developers - Raspberry PI, serial communication

7. Python is being applied in real revenue generating products by real companies.

8. Google makes extensive use of Python in its web search system, and employs Python's creator.

9. Intel, Cisco, Hewlett-Packard, Seagate, Qualcomm and IBM use Python for hardware testing.

10. The YouTube video sharing is largely written in Python.

Conclusion

I believe the trial has shown conclusively that it is both possible and desirable to use Python as the principal teaching language.

- It is Free (as in both cost and source code).
- It is trivial to install on a Windows PC allowing students to take their interest further.
- It is a flexible tool that allows both the teaching of traditional procedural programming and modern OOP.
- It is a real world programming language that can be used in academia and the commercial world.
- It appears to be quicker to learn and, in combination with its many libraries, this offers the possibility of more rapid student development allowing the course to be made more challenging and varied.

Most Importantly, Its clean syntax offers increased understanding and enjoyment for students.