

LAB Manual

PART A

(PART A: TO BE REFERRED BY STUDENTS)

Experiment No. 01

A.1 Aim:

Exploring basics of python like data types (strings, list, array, dictionaries, set, tuples) and control statements.

A.2 Prerequisite:

1. C, JAVA Language

A.3 Outcome:

After successful completion of this experiment students will be able to understand basic concepts in python

A.4 Theory:

Python is a high-level, interpreted and general-purpose dynamic programming language that focuses on code readability. The syntax in Python helps the programmers to do coding in fewer steps as compared to Java or C++. The language founded in the year 1991 by the developer Guido

Van Rossum has the programming easy and fun to do. The Python is widely used in bigger organizations because of its multiple programming paradigms. They usually involve imperative and object-oriented functional programming. It has a comprehensive and large standard library that has automatic memory management and dynamic features.

Interactive

- Interpreted
- Modular
- Dynamic
- Object-oriented
- Portable
- High level· Extensible in C++ & C

Advantages or Benefits of Python:

The Python language has diversified application in the software development companies such as in gaming, web frameworks and applications, language development, prototyping, graphic design applications, etc. This provides the language a higher plethora over other programming languages used in the industry. Some of its advantages are

- **Extensive Support Libraries** It provides large standard libraries that include the areas like string operations, Internet, web service tools, operating system interfaces and protocols. Most of the highly used programming tasks are already scripted into it that limits the length of the codes to be written in Python.

- **Integration Feature** Python integrates the Enterprise Application Integration that makes it easy to develop Web services by invoking COM or COBRA components. It has powerful control capabilities as it calls directly through C, C++ or Java via Jython. Python also processes XML and other markup languages as it can run on all modern operating systems through same byte code.

- **Improved Programmer's Productivity** The language has extensive support libraries and clean object-oriented designs that increase two to ten fold of programmer's productivity while using the languages like Java, VB, Perl, C, C++ and C#.

- **Productivity** With its strong process integration features, unit testing framework and enhanced control capabilities contribute towards the increased speed for most applications and productivity of applications. It is a great option for building scalable multi-protocol network applications.

Limitations or Disadvantages of Python Python has varied advantageous features, and programmers prefer this language to other programming languages because it is easy to learn and code too.

However, this language has still not made its place in some computing arenas that includes Enterprise Development Shops. Therefore, this language may not solve some of the enterprise solutions, and limitations include:

- **Difficulty in Using Other Languages:**

The Python lovers become so accustomed to its features and its extensive libraries, so they face problem in learning or working on other programming languages. Python experts may see the declaring of cast "values" or variable "types", syntactic requirements of adding curly braces or semi colons as an onerous task.

- **Weak in Mobile Computing** Python has made its presence on many desktop and server platforms, but it is seen as a weak language for mobile computing. This is the reason very few mobile applications are built in it like Carbonnelle.

- **Gets Slow in Speed:**

Python executes with the help of an interpreter instead of the compiler, which causes it to slow down because compilation and execution help it to work normally. On the other hand, it can be seen that it is fast for many web applications too.

- **Run-time Errors:**

The Python language is dynamically typed so it has many design restrictions that are reported by some Python developers. It is even seen that it requires more testing time, and the errors show up when the applications are finally run.

- **Underdeveloped Database Access Layers:**

As compared to the popular technologies like JDBC and ODBC, the Python's database access layer is found to be bit underdeveloped and primitive. However, it cannot be applied in the

enterprises that need smooth interaction of complex legacy data.

A.5 Procedure:

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. For example –

```
#!/usr/bin/python
counter = 100 # An integer assignment
miles = 1000.0 # A floating point
name = "John" # A string
print counter
print miles
print name
```

Here, 100, 1000.0 and "John" are the values assigned to counter, miles, and name variables, respectively. This produces the following result –

```
100
1000.0
John
```

Multiple Assignment

Python allows you to assign a single value to several variables simultaneously. For example –

```
a = b = c = 1
```

Here, an integer object is created with the value 1, and all three variables are assigned to the same memory location. You can also assign multiple objects to multiple variables. For example–

```
a,b,c = 1,2,"john"
```

Here, two integer objects with values 1 and 2 are assigned to variables a and b respectively, and one string object with the value "john" is assigned to the variable c.

Standard 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. For example –

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

You can also delete the reference to a number object by using the del statement. The syntax of the del statement is –

```
del var1[,var2[,var3[....,varN]]]]
```

You can delete a single object or multiple objects by using the del statement. For example –

```
del var  
del var_a, var_b
```

Python supports four different numerical types –

- int (signed integers)
- long (long integers, they can also be represented in octal and hexadecimal)
- float (floating point real values)
- complex (complex numbers)

Python allows you to use a lowercase l with long, but it is recommended that you use only an uppercase L to avoid confusion with the number 1. Python displays long integers with an uppercase L.

A complex number consists of an ordered pair of real floating-point numbers denoted by $x + yj$, where x and y are the real numbers and j is the imaginary unit.

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.

The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator. For example –

```
#!/usr/bin/python
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

Lists are the most versatile of Python's compound data types. A list contains items separated by commas and enclosed within square brackets ([]). To some extent, lists are similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data types.

The values stored in a list can be accessed using the slice operator ([] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1. The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator. For example–

```
#!/usr/bin/python
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
tinylist = [123, 'john']
print list # Prints complete list
print list[0] # Prints first element of the list
print list[1:3] # Prints elements starting from 2nd till 3rd
print list[2:] # Prints elements starting from 3rd element
print tinylist * 2 # Prints list two times
print list + tinylist # Prints concatenated lists
```

This produce 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']
```

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 parentheses.

The main differences 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. For example –

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

This produce 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 –

```
#!/usr/bin/python
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 work like associative arrays or hashes found in Perl and 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 ([]). For example –

```
#!/usr/bin/python
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 produce the following result –

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

Dictionaries have no concept of order among elements. It is incorrect to say that the elements are "out of order"; they are simply unordered.

PART B

(PART B : TO BE COMPLETED BY STUDENTS)

Roll no.: A01	Name: Mayur Shankar Giri
Class: AI & DS	Batch: A1
Date of Experiment: 10/01/2024	Date of Submission: 17/01/2024
Grade:	

Document created by the student:

Data types program:

```
import array
```

```
strng = "Hello"
```

```
tuple = (1,2,3,4,5)
```

```
lst = [3,4,5,6]
```

```
intgr = 9
```

```
flt = 3.5
```

```
rng = range(9)
```

```
dictnry = {"name": "John", "age": 30}
```

```
bte = bytes(2)
```

```
btearray = bytearray("hello", "utf-8" )
```

```
boolean = True
```

```
st = {2,7,8,9}
```

```
cmplx = 2+3j
```

```
arr = array.array('i',[1,2,3,5,6])
```

```
print("string = ", strng)
```

```
print("Tuple = ", tuple)
```

```
print("List = ", lst)
```

```
print("Integer = ", intgr)
```

```
print("Float = ", flt)
```

```
print("Range = ", rng)
```

```
print("Dictionary = ", dictnry)
```

```
print("Bytes = ", bte)
```

```
print("Bytearray = ", btearray)
```

```
print("Boolean = ", boolean)
```

```
print("Set = ", st)
```



```
print("Complex Number = ", cmplx)
```

```
for i in range(len(arr)):
    print(f"Array[{i}] = {arr[i]}")
```

```
lst[1] = "2"
print("List = ", lst)
```

```
arr[1] = 9
#print("Array = ", arr)
for i in range(len(arr)):
    print(f"Array[{i}] = {arr[i]}")
```

```
dictnry["roll no"] = "20"
print("Dictionary = ", dictnry)
```

```
#tple[1] = 2
#strng[1] = "2"
#st[1] = "2"
```

Output:

```
D:\A2_41>python datatypes.py
string = Hello
Tuple = (1, 2, 3, 4, 5)
List = [3, 4, 5, 6]
Integer = 9
Float = 3.5
Range = range(0, 9)
Dictionary = {'name': 'John', 'age': 30}
Bytes = b'\x00\x00'
Bytearray = bytearray(b'hello')
Boolean = True
Set = {8, 9, 2, 7}
Complex Number = (2+3j)
Array[0] = 1
Array[1] = 2
Array[2] = 3
Array[3] = 5
Array[4] = 6
List = [3, '2', 5, 6]
Array[0] = 1
Array[1] = 9
Array[2] = 3
Array[3] = 5
Array[4] = 6
Dictionary = {'name': 'John', 'age': 30, 'roll no': '20'}
```

Control Statements program:

```
print("Welcome!!")
print("1) Addition")
print("2) Subtraction")
print("3) Division")
print("4) Multiplication")
print("5) Modulus")
print("6) Exponent")
print("7) Floor Division")
```

```
ch = input("Enter the choice: ")
```

```
if ch == '1':
```

```
    num1 = int(input("Enter the first number: "))
    num2 = int(input("Enter the second number: "))
    sum = num1 + num2
    print(f"The addition of {num1} and {num2} is : {sum}")
```

```
elif ch == '2':
```

```
    num1 = float(input("Enter the first number: "))
    num2 = float(input("Enter the second number: "))
    sub = num1 - num2
    print(f"The subtraction of {num2} from {num1} is: {sub}")
```

```
elif ch == '3':
```

```
    num1 = float(input("Enter the first number: "))
    num2 = float(input("Enter the second number: "))
    mul = num1 * num2
    print(f"The multiplication of {num1} and {num2} is: {mul}")
```

```
elif ch == '4':
```

```
    num1 = float(input("Enter the first number: "))
    num2 = float(input("Enter the second number: "))
    div = num1 / num2
    print(f"The division of {num1} by {num2} is: {div}")
```

```
elif ch == '5':
```

```
    num1 = float(input("Enter the first number: "))
    num2 = float(input("Enter the second number: "))
    mod = num1 % num2
    print(f"The subtraction of {num1} by {num2} is: {mod}")
```

```
elif ch == '6':
```

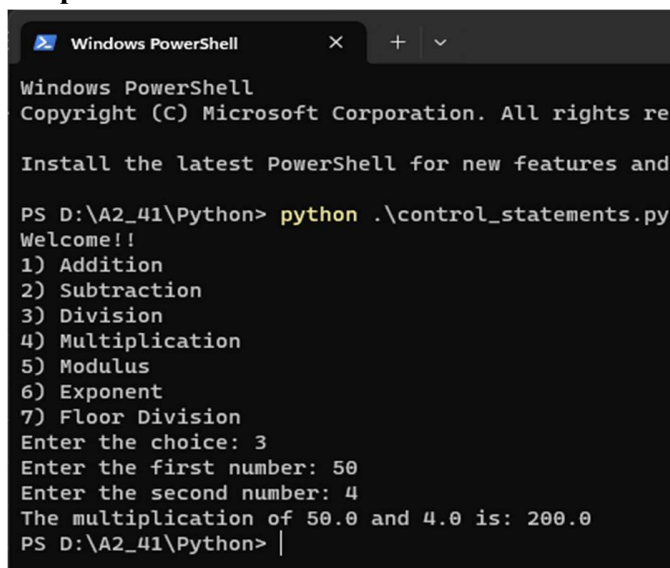
```
    num1 = float(input("Enter the first number: "))
    num2 = float(input("Enter the second number: "))
```

```

    mod = num1 % num2
    print(f"The subtraction of {num1} by {num2} is: {mod}")
elif ch == '7':
    num1 = float(input("Enter the first number: "))
    num2 = float(input("Enter the second number: "))
    mod = num1 % num2
    print(f"The subtraction of {num1} by {num2} is: {mod}")
else:
    print("Invalid choice!")

```

Output:



```

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and
improvements: https://aka.ms/powershell

PS D:\A2_41\Python> python .\control_statements.py
Welcome!!
1) Addition
2) Subtraction
3) Division
4) Multiplication
5) Modulus
6) Exponent
7) Floor Division
Enter the choice: 3
Enter the first number: 50
Enter the second number: 4
The multiplication of 50.0 and 4.0 is: 200.0
PS D:\A2_41\Python> |

```

B.1 Observations and learning:

The data types are automatically assigned by the Python programming language. User did not have to assign the data types to the variables manually.

B.2 Conclusion:

The data types are automatically assigned by the Python programming language. Python programming language supports five primitive data types. The logical and arithmetic operations can be carried out easily in Python.

B.3 Question of Curiosity

Q.1] Differentiate between List and Tuple?

Ans]

List	Tuple
<ul style="list-style-type: none"> • Lists are mutable. • Syntax: my_list = [1, 2, 3] • Can be modified • Slightly slower compared to tuples. • More memory usage due to flexibility 	<ul style="list-style-type: none"> • Tuples are immutable • Syntax: my_tuple = (1, 2, 3) • Cannot be modified after creation • Generally faster compared to lists. • Less memory usage due to immutability

Q.2] Use if elif control statement for Percentage of marks for student?

Ans]

```
marks = float(input("Enter the percentage of marks: "))
```

```
if marks >= 90:
```

```
    grade = 'A'
```

```
elif 80 <= marks < 90:
```

```
    grade = 'B'
```

```
elif 70 <= marks < 80:
```

```
    grade = 'C'
```

```
elif 60 <= marks < 70:
```

```
    grade = 'D'
```

```
else:
```

```
    grade = 'F'
```

```
print("The marks of student are {0}, hence the grade acquired by the student is {1}".format(marks, grade))
```