Lovely Professional University

CA1

School: Mittal School of Business **Lovely Faculty of Business and Arts**
Name of the Faculty Member: Logesh Kumar, Rupesh
Course Code: MGNM801 **Course Title:** Business Analytics-I
Section: Q2240
Date of Allotment: 22/12/2022 **Date of Submission:** 26/12/2022
Attempt (Group/Individual): Individual **Max Marks** :30

Name: Nowneesh T **Roll No:** RQ2240A19
Reg No: 12202342

S No.	Objectives	Task	Marks	Evaluation Parameters
1	Understanding Basics of Python by applying simple examples	What is python? Explain the following concepts with examples: • Variables • Keywords • Identifiers • Operators	10	Research Capability – 3 marks Ability to describe – 2 marks Clean Coding – 5 marks
2	Applying the basic concepts of programming	Show 1 to 10 counting with the help of for and while loop	10	Ability to elaborate – 5 marks Ability to write code – 5 marks
3	Understanding of basic python datatypes	Write a program to describing the major methods we can apply on the following datatypes: 1. Lists 2. Strings 3. Tuples 4. Dictionaries Also describe each datatypes in brief	10	Understanding of Dictionaries – 3 marks understanding of lists – 3 marks Understanding of Tuples – 1 mark understanding of Strings – 3 marks

1. **<u>What is python? Explain the following concepts with examples:</u>**

- Variables
- Keywords
- Identifiers
- Operators

Python is an interpreted, device, high-level, dynamically semantic programming language. It is particularly desirable for Rapid Application Development as well as for usage as a scripting or glue language to tie existing components together due to its high-level built-in data structures, dynamic typing, and dynamic binding. Python's straightforward syntax prioritises readability and makes it simple to learn, which lowers the cost of programme maintenance. Python's support for modules and packages promotes the modularity and reuse of code in programmes. For all popular systems, the Python interpreter and the comprehensive standard library are freely distributable and accessible in source or binary form.

Variables

A symbolic name that serves as a reference or pointer to an object is called a variable in Python. You can use the variable name to refer to an object once it has been allocated to it.

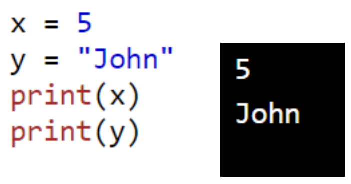
Identifiers

The term "Python Identifiers" refers to all of the variables, classes, objects, functions, lists, dictionaries, etc. in Python. The foundation of any Python programme is its identifiers. Nearly every Python programme makes use of identifiers in some way.

Python identifier and variable usage guidelines:

- A name for an identifier shouldn't be a keyword.
- Only a letter or an underscore may come before an identifier name.
- An identification name may include underscores (A-z, 0-9, and _) as well as numbers and letters.
- Python is case-sensitive, therefore sum and Sum are two different identifier names.

Example:



```python
x = 5
y = "John"
print(x)
print(y)
```

```
5
John
```

Keywords:

The following reserved words cannot be used as variable names, function names, or any other identifiers in the Python programming language:

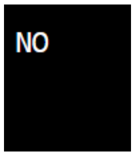
Keyword	Description
and	A logical operator
as	To create an alias
assert	For debugging
break	To break out of a loop
class	To define a class
continue	To continue to the next iteration of a loop
def	To define a function
del	To delete an object
elif	Used in conditional statements, same as else if
else	Used in conditional statements
except	Used with exceptions, what to do when an exception occurs
False	Boolean value, result of comparison operations
finally	Used with exceptions, a block of code that will be executed no matter if there is an exception or not
for	To create a for loop
from	To import specific parts of a module
global	To declare a global variable
if	To make a conditional statement
import	To import a module

in	To check if a value is present in a list, tuple, etc.
is	To test if two variables are equal
lambda	To create an anonymous function
None	Represents a null value
nonlocal	To declare a non-local variable
not	A logical operator
or	A logical operator
pass	A null statement, a statement that will do nothing

raise	To raise an exception
return	To exit a function and return a value
True	Boolean value, result of comparison operations
try	To make a try...except statement
while	To create a while loop
with	Used to simplify exception handling
yield	To end a function, returns a generator

Example:

```
x = 2
if x > 3:
 print("YES")
else:
 print("NO")
```



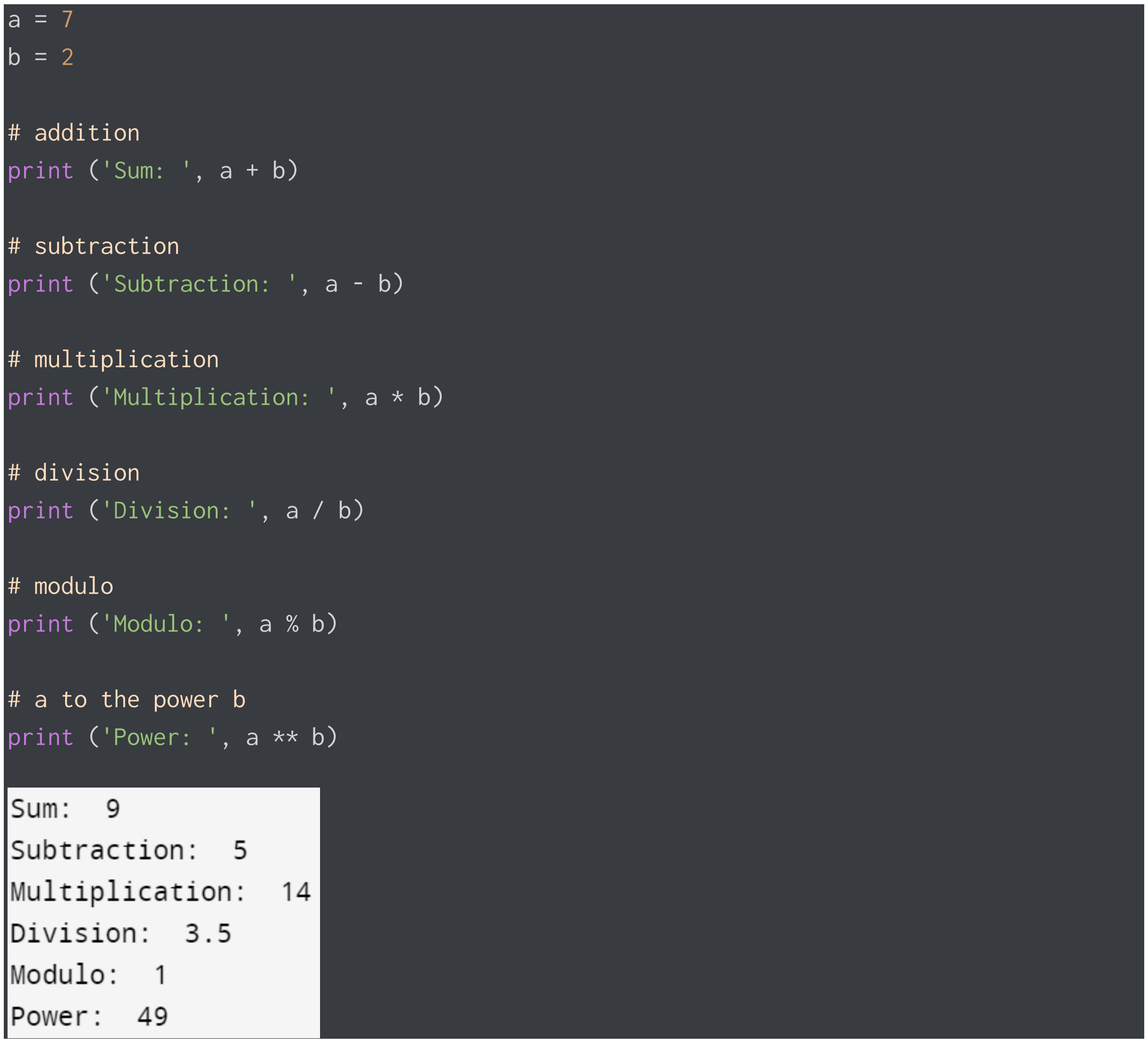
```
NO
```

Operators:

Operators are used to perform operations on variables and values.

Python divides the operators in the following groups:

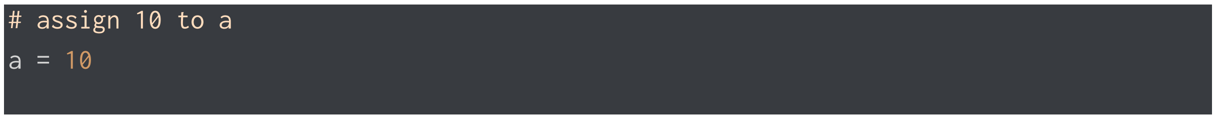
- Arithmetic operators - They used with numeric values to perform common mathematical operations



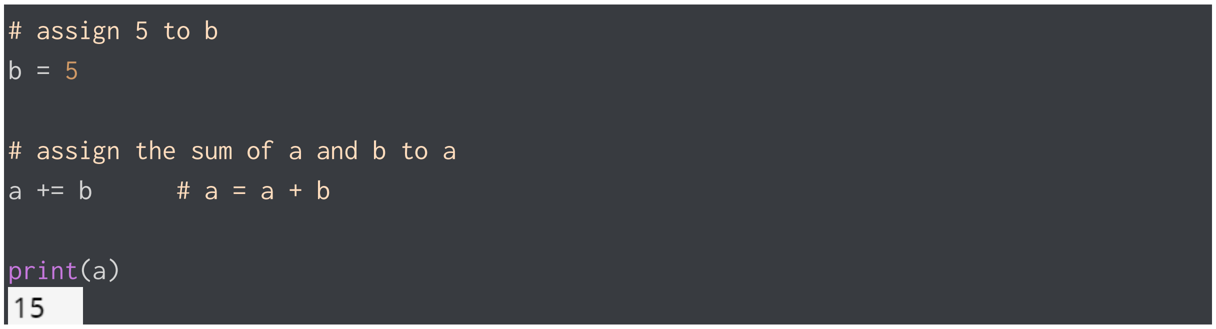
```python
a = 7
b = 2

# addition
print ('Sum: ', a + b)

# subtraction
print ('Subtraction: ', a - b)

# multiplication
print ('Multiplication: ', a * b)

# division
print ('Division: ', a / b)

# modulo
print ('Modulo: ', a % b)

# a to the power b
print ('Power: ', a ** b)
```

```
Sum:   9
Subtraction:   5
Multiplication:   14
Division:   3.5
Modulo:   1
Power:   49
```

- Assignment operators – They are used to assign values to variables



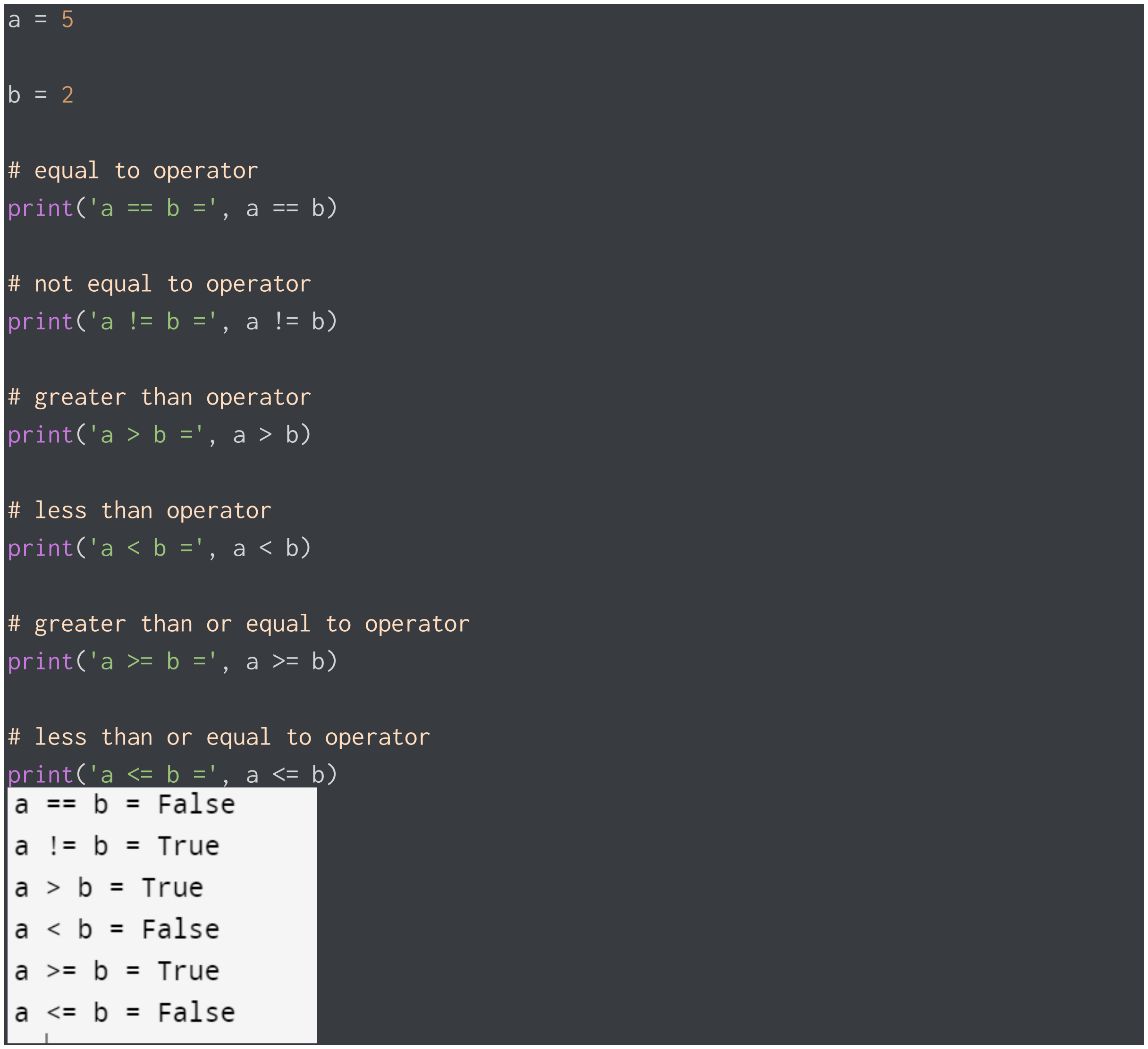
```python
# assign 10 to a
a = 10
```



```python
# assign 5 to b
b = 5

# assign the sum of a and b to a
a += b      # a = a + b

print(a)
```

```
15
```

- Comparison operators – They are used to compare two values



```python
a = 5

b = 2

# equal to operator
print('a == b =', a == b)

# not equal to operator
print('a != b =', a != b)

# greater than operator
print('a > b =', a > b)

# less than operator
print('a < b =', a < b)

# greater than or equal to operator
print('a >= b =', a >= b)

# less than or equal to operator
print('a <= b =', a <= b)
```

```
a == b = False
a != b = True
a > b = True
a < b = False
a >= b = True
a <= b = False
```

- Logical operators – They are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location

```python
# logical AND
print(True and True)     # True
print(True and False)    # False

# logical OR
print(True or False)     # True

# logical NOT
print(not True)          # False
```

```
True
False
True
False
```

- Identity operators – They are used to test if a sequence is presented in an object

```python
x1 = 5
y1 = 5
x2 = 'Hello'
y2 = 'Hello'
x3 = [1,2,3]
y3 = [1,2,3]

print(x1 is not y1)  # prints False

print(x2 is y2)  # prints True

print(x3 is y3)  # prints False
```

```
False
True
False
```

- Membership operators – They are used to test if a sequence is presented in an object

```python
message = 'Hello world'
dict1 = {1:'a', 2:'b'}
```

```
# check if 'H' is present in message string
print('H' in message)  # prints True

# check if 'hello' is present in message string
print('hello' not in message)  # prints True

# check if '1' key is present in dict1
print(1 in dict1)  # prints True

# check if 'a' key is present in dict1
print('a' in dict1)  # prints False
```

```
True
True
True
False
```

- Bitwise operators - They are used to compare (binary) numbers

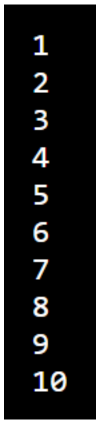
```
a = 10
b = 4

# Print bitwise AND operation
print("a & b =", a & b)

# Print bitwise OR operation
print("a | b =", a | b)

# Print bitwise NOT operation
print("~a =", ~a)

# print bitwise XOR operation
print("a ^ b =", a ^ b)
```

```
a & b = 0
a | b = 14
~a = -11
a ^ b = 14
```

2. <u>Show 1 to 10 counting with the help of for and while loop</u>

For loop:

```
for x in range(1,11):
  print(x)
```



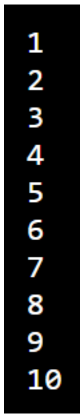
```
1
2
3
4
5
6
7
8
9
10
```

This loop will help us to understand about the range. In range, I give 11 instead of 10 because for loop will take upto previous value. So it won't consider 11.

It will increment value in the specified range.

While loop:

```
i = 1
while i < 11:
  print(i)
  i += 1
```



```
1
2
3
4
5
6
7
8
9
10
```

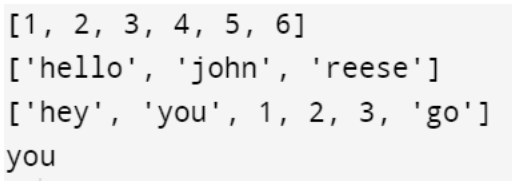
This loop will help us to understand about the condition. In condition, I give 11 instead of 10 because while loop will take upto previous value based on condition. So it won't consider 11. It will increment value with the condition specified.

3. **<u>Write a program to describing the major methods we can apply on the following datatypes:</u>**
<u>1. Lists</u>
<u>2. Strings</u>
<u>3. Tuples</u>
<u>4. Dictionaries</u>
<u>Also describe each datatypes in brief.</u>

The classification or categorising of data elements is known as data types. It represents the type of value that specifies the operations that can be carried out on a given set of data. In Python programming, everything is an object, hence variables and data types are both instances (or objects) of the same classes.

1. Lists

```
#list of having only integers
a= [1,2,3,4,5,6]
print(a)

#list of having only strings
b=["hello","john","reese"]
print(b)

#list of having both integers and strings
c= ["hey","you",1,2,3,"go"]
print(c)

#index are 0 based. this will print a single character
print(c[1]) #this will print "you" in list c
```



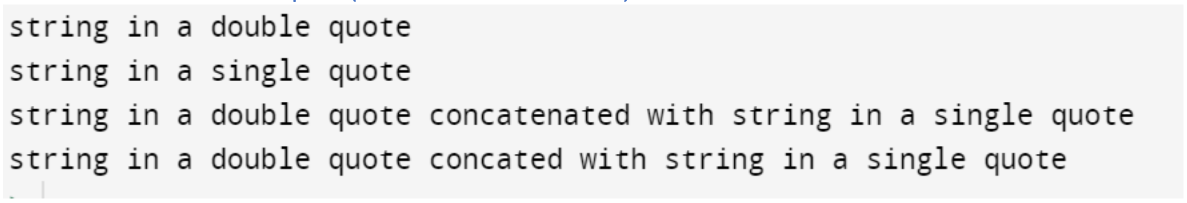
```
[1, 2, 3, 4, 5, 6]
['hello', 'john', 'reese']
['hey', 'you', 1, 2, 3, 'go']
you
```

The list is a flexible data type that is only available in Python. It resembles the array in C/C++ in certain ways. However, the list in Python is noteworthy because it can store multiple sorts of data at once. Formally, a list is an ordered collection of information that is written with commas and square brackets ([]). (,).

2. Strings

```
a = "string in a double quote"
b= 'string in a single quote'
print(a)
print(b)

# using ',' to concatenate the two or several strings
print(a,"concatenated with",b)
```

```
#using '+' to concate the two or several strings
print(a+" concated with "+b)
```

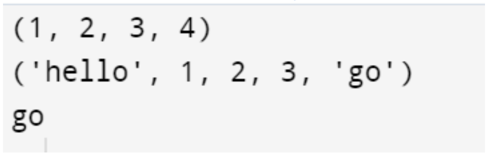


```
string in a double quote
string in a single quote
string in a double quote concatenated with string in a single quote
string in a double quote concated with string in a single quote
```

A series of characters make up the string. Python can handle characters in Unicode. Typically, single or double quotations are used to denote strings.

3. Tuples

```
#tuple having only integer type of data.
a=(1,2,3,4)
print(a) #prints the whole tuple

#tuple having multiple type of data.
b=("hello", 1,2,3,"go")
print(b) #prints the whole tuple

#index of tuples are also 0 based.

print(b[4]) #this prints a single element in a tuple, in this case "go"
```



```
(1, 2, 3, 4)
('hello', 1, 2, 3, 'go')
go
```

The above data is a tuple, which is a list-like sequence of data. But it cannot change. This indicates that a tuple's data is write-protected. A tuple's data is expressed using parenthesis and commas.

4. Dictionaries

```
#a sample dictionary variable

a = {1:"first name",2:"last name", "age":33}

#print value having key=1

print(a[1])

#print value having key=2

print(a[2])

#print value having key="age"

print(a["age"])
```

```
first name
last name
33
```

A key-value pair-formatted unordered series of data is a Python dictionary. It resembles the hash table kind. Dictionary entries take the key:value format and are enclosed in curly brackets. The ability to efficiently obtain info from a vast volume of data is tremendously helpful.