INDEX

Date: 16/1/25 Page No. 1

Assignment No.: 1

Assignment Topic: Repetition in python

1. Explain Repetition in python using suitable example?

A. Python has two primitive loop commands:-

- while loops
- for loops.

The while loop :-

With the while loop we can execute a set of statements as long as a condition is true.

Syntax for while loop:-

```
while condition:
    statements....
```

Ex:- i=1

```
while i<4:
    Print (i).
    i+=1
```

output:-

1
2
3

The break statement:-

With the break statement we can stop the loop even if the while condition is true.

Ex:- i=0

```
while i<6:
    Print (i)
    i+=1
    if i==4:
        break
```

output:-

0
1
2
3

The continue Statement:-

With the continue statement we can stop the current

iteration, and continue with the next:

```
i=0
while i<6:
    i+1=1
    if i==3:
        continue.
        Print (i)
```

output :-

1
2
4
5
6

The else statement :-

With the else statement we can run a block of code once when the condition no longer is true:

Ex:- i=0

```
while i<3:
    Print(i)
    i+=1
else:
    Print("i is no longer less than 3")
```

output :-

0

output :-

0
1
2

i is no longer less than 3

for loop :-

for loop are used when you have a block of code which you want to repeat a fixed no. of times. A for loop is used for iterating over a sequence. This is less like the for keyword in other programming languages, and works more like an iterator method as found in others object-oriented programming

languages.

with the for loop we can execute a set of statements, once for each item in a list, tuple, set etc..

Syntax for for loop :-

```
For val in sequence :
    statements...
```

Ex :- fruits = ["apple", "banana", "cherry"]

```
for x in fruits :
    print(x)
```

output :-

apple
banana
cherry

The break statement :-

with the break statement we can stop the loop before it has looped through all the items.

Ex :- fruits = ["apple", "banana", "cherry"]

```
for x in fruits :
    print(x)
    if x == "banana" :
        break
```

output :-

apple
banana.

The continue statement :-

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

Ex : fruits = ["apple", "banana", "cherry"]

```
For x in fruits:
    if x == "banana":
        continue
    print(x)
```

output:-

apple
cherry.

The range() function:-

To loop through a set of code a specified no. of times, we can use the range() function, The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 and ends at a specified number.

Ex:- for x in range(3):
    print(x)

output:-

0
1
2

Else in for loop:-

The else keyword in a for loop specifies a block of code to be executed when the loop is finished:

Ex:- For x in range(3):
    print(x)
else:
    print("finally finished!")

output:-

0
1
2
Finally finished!

Nested loops :-

A nested loop is a loop inside a loop.
The "inner loop" will be executed one time for each iteration of the "outer loop".

Ex :-

```
adj = ["red", "big"]
fruits = ["apple", "banana"]
for x in adj :
    for y in fruits :
        print (x, y)
```

output :-

red apple
red banana
big apple
big banana

The pass Statement :-

For loops cannot be empty, but if you for some reason have a for loop with no content, put in the pass statement to avoid getting an error.

Ex :-

```
For x in [0, 1, 2]:
    pass
```

Date: 27/1/25 Page No. 7

Assignment No.: 02

Assignment Topic: string operations

2. Discuss python strings operations with example?

A. Strings in python are surrounded by either single quotation marks, or double quotation marks. 'hello' is same as "hello". you can display a string literal with the print() function:

```
a="Hello"
print(a)
```

output: Hello

Multiline Strings:-

you can assign a multiline string to a variable by using three quotes:

```
a= """ we are using
      python strings"""
print(a)
```

output:- We are using
python strings

Strings are Arrays:-

```
a="Hello, World!"
print(a[1])
```

output: e

Looping Through a string:-

since strings are arrays, we can loop through the characters in a string, with a for loop.

```
for x in "loop":
    print(x)
```

output: l
o
o
p

String length :-

To get the length of a string, use the len() function

```
a = "Hello, world!"
print(len(a))
```

Output : 13

Check string :-

To check if a certain phrase or character is present in a string, we can use the keyword in.

```
txt = "the best things in life are free!"
print("free" in txt)
```

output :- True

Use it in an if statement :-

```
txt = "The best things in life are free!"
if "free" in txt:
    print("yes, 'free' is present.")
```

output :- yes, 'free' is present.

Slicing Strings :-

You can return a range of characters by using the slice syntax.

Specify the start index and the end index, seperated by a colon, to return a part of the string.

```
b = "Hello, world!"
print(b[2:5])
```

output : "Hello, world!"

Python - modify Strings :-

Upper case :

The upper() method returns the string in upper case :

```
a = 'Hello, world!"
```

print(a.upper())
output: HELLO, WORLD!

Lower case :-
The lower() method returns the string in lower case:
a = "Hello, world"
print(a.lower())
output: hello, world

Remove whitespace :-
white space is the space before and/or after the actual text, and very often you want to remove this space.
The strip() method removes any whitespaces from the beginning or the end:
a = " Hello, world "
print(a.strip())
Output: Hello, world

Replace string :-
the replace() method replaces a string with another string:
a = "Tom and Jerry"
print(a.replace("T", "J"))
output: Jom and Jerry

Split string :-
The split() method returns a list where the text between the specified seperator becomes the list items.

```
a = "Hello, world!"
print(a.split(", "))
['Hello', 'world!']
```

String concatenation:-

To concatenate, or combine, two strings you can use the + operator.

```
a='Hello'
b='world'
c= a+b
print(c)
```

output: Hellowworld

Date: 5/2/25 Page No. 11

Assignment No.: 3

Assignment Topic: class & Inheritance in python.

3. what is class? Explain inheritance in python.

A. Python is an object oriented programming language. Almost everything in python is an object, with its properties and methods. A class is like an object constructor or a "blueprint" for creating objects.

Create class:-

To create a class, use the keyword class:

Create a class named myclass, with a property named x:

```
class myclass:
  x=5
print(myclass)
```

output: <class'__main__.myclass'>

Create object:-

Now we can use the class named myclass to create objects:

Create an object named p1, and print the value of x:

```
Class myclass:
  x=5
p1= myclass()
print(p1.x)
```

output: 5

Inheritance in python:-

Inheritance allows us to define a class that inherit all the methods and properties from another class. parent class is the class being inherited from, also

base class. child class is the class that inherit from another class, also called derived class.

Create a parent class:

Any class can be a parent class, so the syntax is the same as creating any other class:

create a class person, with firstname and last name properties, and a printname method:

Create a child class:-

To create a class that inherits the functionality from another class, send the parent class as a parameter when create the child class:

Ex:-

```
Class person:
    def __init__(self, fname, lname):
        self.firstname = fname
        self.lastname = lname
    def printname(self):
        print(self.firstname, self.lastname)
class student(person):
    pass
x = Student("mike", "olsen")
x.printname()
```

output:- mike olsen

Date: 12/2/25 Page No. 13

Assignment No.: 4

Assignment Topic: Exceptions

4. Define Exceptions? Explain it's types?

A. Python Exception handling handles errors that occurs during the execution of a program exception handling allows to respond to the error, instead of crashing the running program.

→ Try block let you test a block of code for errors

→ The except block lets you handle the error.

→ The else block lets you execute code when there is no error.

→ The finally block lets you execute code, regardless of the result of the try and except blocks.

Exception Name	Description
Base Exception	The base class for all built-in exceptions.
Exception	The base class for all non-exit exceptions.
Arithmetic Error	Base class for all errors related to arithmetic operations.
Zero Division Error	Raised when a division or modulo operation is performed with zero as the divisor.
Overflow Error	Raised when a numerical operation exceeds the maximum limit of a data type.
Floating point Error	Raised when a floating-point operation fails.
Assertion Error	Raised when an assert statement fails.
Attribute Error	Raised when an attribute reference or assignment fails.
Index Error	Raised when a sequence subscript is out of range.

KeyError	Raised when a dictory key is not found.
memoryError	Raised when an operation runs out of memory
NameError	Raised when a local or global name is not found.
OSError	Raised when a system-related operation (like file I/O) fails.
TypeError	Raised when an operation or function is applied to an object of inappropriate type.
ValueError	Raised when a function receives an argument of the right type but inappropriate value.
importError	Raised when an import statement has issues.
moduleNotFound Error	Raised when a module cannot be found.

Date: 17/2/25 Page No. 15

Assignment No.: 05

Assignment Topic: List operations

5. what is list? Explain different list operations and including the list traversing?

A. List are used to store multiple items in a single variable. List are one of 4 built-in-data types in python used to store collections of data, the other 3 are Tuple, Set, and Dictionary, all with different qualities and usage.

List are created using square brackets:

Creating List:

```
Mylist = [1,2,"apple"]        output:- [1, 2, 'apple']
print (mylist)
```

List operations:-

Change item value:-

To change the value of a specific item, refer to the index number:

Ex:- thislist = ["apple", "banana", "cherry"]

```
thislist [1] = "blackcurrent"
print (thislist)
```

output:- ['apple', 'blackcurrant', 'cherry']

Insert Items:-

To insert a new list item, without replacing any of the existing values, we can use the insert() method.

The insert() method insert an item at the specified index:

insert "watermelon" as the third item:

Ex:- Thislist = ["apple", "Banana", "cherry"]

```
thislist.append("orange")
print(thislist)
```

output :- ['apple', 'banana', 'cherry', 'orange']

Extend list :-

To append elements from another list to the current list, use the extend() method.

Add the elements of tropical to thislist:

Ex:-

```
thislist = ["apple", "banana", "cherry"]
tropical = ["mango", "pineapple", "papaya"]
thislist.extend(tropical)
print(thislist)
```

output :- ['apple', 'banana', 'cherry', 'mango', 'pineapple', 'papaya']

Remove list Items :-

The remove() method removes the specified item.

Remove "banana".

Ex:-

```
thislist = ["apple", "banana", "cherry"]
thislist.remove("banana")
print(thislist)
```

output :- ['apple', 'cherry']

The del keyword :-

The del keyword can also delete the list completely.

Delete the entire list:

Ex:-

```
thislist = ["apple", "banana", "cherry"]
```

del thislist

clear the list :-

The clear() method empties the list

The list still remains, but it has no content

clear the list content:

Ex :-

```
thislist = ["apple", "banana"]
thislist.clear()
print(thislist)
```

Output : []

loop through a list : [traversing a list]

you can loop through the list items by using a for loop :

print all items in the list, one by one.

Ex :-

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

output :-

apple
banana
cherry.

Date: 24/2/25 Page No. 19

Assignment No.: 06

Assignment Topic: Dictionary

6. Define Dictionary? Explain different operations on Dictionary?

A. Dictionaries are used to store data values in key: value pairs. A dictionary is a collection, which is

- Ordered
- changeble
- do not allows duplicates.

Dictionary operations:-

Accessing items:-

you can access the items of a dictionary by refering to its key name, inside square brackets.

Ex:-

```
thisdict = {
"brand": "Ford",
"model": "mustang" }
x = thisdict["model"]; print(x)
```

output:- mustang.

get():-

This method is used to access the elements present in the dictionary

Ex:-

```
thisdict = {
"brand": "ford",
"model": "mustang" }
x = thisdict.get("model")
print(x)
```

output:-

mustang.

Get keys :-

The keys() method will return a list of all the keys in the dictionary. Add a new item to the original dictionary.

Ex:- car = {

"brand": "Ford",

"model": "mustang" }

x = car.keys()

print(x)

car["color"] = "white"

print(x)

Output:-

dict_keys(['brand', 'model', 'year'])

dict_keys(['brand', 'model', 'color'])

Adding items :-

Adding an item to the dictionary is done by using a new index key and assigning a value to it.

Ex:-

thisdict = { "brand": "Ford",

"model": "mustang" }

thisdict["color"] = "red"

print(thisdict)

output:- {'brand': 'Ford', 'model': 'mustang', 'color': 'red'}

Update Dictionary :-

The update() method will update the dictionary with the items from a given argument. If the item does not exist, the item will be added.

The argument must be a dictionary, or an iterable

Object with key : value pairs.

```
this dict = {
  "brand" : "ford",
  "model" : "mustang" }
thisdict.update({"model" : "car x"}); print(thisdict)
```

Output :-

```
{'brand' : 'Ford', 'model' : 'car x'}
```

Removing items :-

There are several methods to remove items from a dictionary.

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

```
thisdict = {"brand" : "ford",
"model" : "mustang" }
thisdict.pop("model")
print(thisdict)
```

output :-

```
{'brand' : 'Ford'}
```

The popitem() method removes the last inserted item.

```
thisdict = {'brand" : "ford",
"model" : "mustang" }
thisdict.popitem()
print(thisdict)
```

output :-

```
{'brand' : 'ford'}
```

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, print all key names in the dictionary, one by one:

```
thisdict = { "brand": "Ford",
"model": "mustang" }
for x in thisdict:
    print(x)
```

output:-

```
brand
model
```

You can also use the values() method to return values of a dictionary.

```
For X in thisdict.values():
    print(x)
```

output:-

```
Ford
Mustang.
```

Date: 1/3/25 Page No. 23

Assignment No.: 7

Assignment Topic: Numpy Array

7. What is Numpy Array? Explain different Numpy array operators?

A. * Numpy is a Python library.
* Numpy is used for working with arrays.
* Numpy is short for "Numerical Python".
* In Python we have lists that serve the purpose of arrays, but they are slow to process.
* Numpy aims to provide an array object that is up to 50x faster than traditional python lists.
* Numpy arrays are stored at one continuous place in memory unlike lists, so processes can access and manipulate them very efficiently.
* We can create a numpy array object by using the array() function.

Operation on Numpy Array:-

Numpy array is a powerful N-dimensional array object which is in the form of rows and columns. We can initialize Numpy arrays from nested python lists and access it elements.

Numpy is an open source mathematical python. the Numpy library offers a collection of high-level mathematical functions including support for multi-dimentional arrays, masked arrays and matrices.

A Numpy array on a structural level is made up of a combination of:

- The Data pointer indicates the memory address of the first byte in the array.
- The Data type or dtype pointer describes the kind of elements that are contained within the array.
- The shape indicates the shape of the array.
- The strides are the number of bytes that should be stipped in memory to go to the next element.

```
import numpy as np
arr1 = np.array([1,2,3,4,5])
arr2 = np.array([6,7,8,9,10])
newarr = np.add(arr1, arr2)
print(newarr)
newarr = np.subtract(arr1,arr2)
print(newarr)
newarr = np.multiply(arr1,arr2)
print(newarr)
newarr = np.divide(arr1,arr2)
print(newarr)
newarr = np.power(arr1,arr2)
print(newarr)
print(main(arr1))
print(max(arr1))
```

Output:-

```
[7 9 11 13 15]
[-5 -5 -5 -5 -5]
[6 14 24 36 50]
[0.16666667 0.28571429 0.375 0.44444444 0.5 ]
[1 128 6561 262144 9765625]
```

Date: 8/3/25 Page No. 25

Assignment No.: 8

Assignment Topic: Series & Data frames

8. Explain Series & Data frames in python?

A. Series:-

A panda series is like a column in a table. It is a one-dimensional array holding data of any type.

```
import pandas as pd
a= [1,7,2]
myvar = Pd. Series (a)
print (myvar)
```

Output: 0 1
1 7
2 2

Labels:-

Label can be used to access a specific value. The value are labeled with their index numbers. first value has index 0, second value has index 1 etc.

```
print (myvar[0])
```

output: 2

Create Labels:-

with the index argument, we can name our import pandas as pd.

```
a= [1,7,2]
myvar = Pd. series (a, index = ["x", "y", "z"])
print (myvar) own labels.
```

Output:- x 1
y 7
z 2

When you have created labels, you can access an item by referring to the label.

print(myvar["y"])

Key-value objects as series:-

We can also use a key/value object, like a dictionary, when creating a series.

```
import pandas as pd
calories = {"day1": 420, "day2": 380, "day3": 390}
myvar = pd.Series(calories)
print(myvar)
```

Output:-

day1 420
day2 380
day3 390

Data frames:-

A Pandas Data frame is a 2 dimensional data structure, like a 2 dimentional array, or a table with rows and columns.

```
import pandas as pd
data = {
"calories": [420, 380, 390],
"duration": [50, 40, 45] }
df = pd.DataFrame(data)
print(df)
```

output :-

	Calories	duration
0	320	50
1	380	40
2	390	45

The DataFrame is like a table with rows and columns. Pandas use the loc attribute to return one or more specified row(s)

refer to the row index :-

```
print(df.loc[0])
```

Output : Calories 420
duration 50

use list of indexes :

```
print(df.loc[[0,1]])
```

output :-

	Calories	duration
0	420	50
1	380	40

Named Index : With the index argument, we can name our own indexes.

```
import pandas as pd
data = { "calories" : [420, 380, 390],
         "duration" : [50, 40, 45] }
df = pd.DataFrame(data, index = ["day1", "day2", "day3"])
print(df)
```

output :-

	Calories	duration
day1	420	50
day2	380	40
day3	390	45

Locate named indexes :- Use the named index in the loc attribute to return the specified rows.

```
print(df.loc["day2"])
```

Output :-

```
Calories  380
duration  40
Name: day2
```

Date: 17/3/25 Page No. 29

Assignment No.: 9

Assignment Topic: matplotlib

Q. Explain plotting using matplotlib?

A. Matplotlib is a popular python library used for creating static, animated and interactive visualization. It provides a flexible framework to generate various types of plots, such as charts, bargraphs histograms and scatter plots.

(1) Install matplotlib:-

If you haven't already installed matplotlib. you can do so using

```
python
pip install matplotlib
```

(2) Basic structure of a plot:-

A basic matplotlib plot consist of figure:
The entire image or canvas
Axes: the plotting area inside the figure
plot elements: Titles, lables, legends, grid; lines etc.

(3). Basic line plot:-

Here's a simple example of plotting a line graph.

```
import matplotlib.python as plt
x = [1,2,3,4,5]
y = [10,20,25,30,35]
plt.plot(x,y, marker='o', linestyle="--", color
```

```
='b', label=["line"]
plt.xlabel("x-axis")
plt.ylabel("y-axis")
plt.title("Basic line plot")
plt.legend()
plt.show()
```

(4) Common types of plots:-

A. Scatter plot

used to show relationship between two variables

Python

```
plt.scatter(x,y, color='r', marker='s')
plt.xlabel("x-axis")
plt.ylabel("y-axis")
plt.title("scatter plot")
plt.show()
```

Bar chart:-

```
categories = ["A", "B", "C", "D"]
values = [5, 7, 3, 8]
plt.bar(categories, values, color='g')
plt.xlabel("categories")
plt.ylabel("values")
plt.title("Barchart")
plt.show()
```

Date : .....

Page No. 31

Assignment No. : .....

Assignment Topic : .....

→ Histogram :-

Used to show the distribution of numerical data.

```
import numpy as np
data = np.random.random(100)
plt.hist(data, bins=30, color='purple', edgecolor="black")
plt.xlabel('black')
plt.ylabel("value")
plt.title("histogram")
plt.show()
```

Date: 25/3/2025 Page No. 33

Assignment No.: 10

Assignment Topic: Python to mySQL Query

10. Explain how to connect python to mysql & use some queries?

A. The mysql.connector provides the connect() method used to create a connection between the mySQL database and the Python application.

Syntax:-

```
conn-obj = mysql.connector.connect (host = <hostname>,
  user = <username>, passwd = <password>)
```

Program:-

```
import mysql.connector
conn = mysql.connector.Connect (user = 'username',
                                host = 'localhost',
                                password = 'password',)
print (conn)
Conn.close()
```

We can use connection mySQL connection() class instead of connect().

Create a Database:-

```
import mysql.connector
mydb = mysql.connector.connect(
host = "localhost",
user = "yourusername",
password = "yourpassword")
```

```
mycursor = mydb.cursor()
mycursor.execute("create database mydatabase")
# to show databases
mycursor.execute("SHOW DATABASES")
for x in mycursor:
    print(x)
```

To create a table in MySQL
use the "CREATE TABLE" statement.

```
import mysql.connector
mydb = mysql.connector.connect(
host = "localhost",
user = "yourusername",
password = "yourpassword",
database = "mydatabase")
mycursor = mydb.cursor()
mycursor.execute("CREATE TABLE customer (name
    varchar(255), address varchar(255))")
# to show tables
# mycursor.execute("SHOW TABLES")
# for x in mycursor:
    # print(x)
```

To insert into a table :-
To fill a table in MySQL, use the "INSERT INTO" statement

```
import mysql.connector
```

```
mydb= mysql.connector.connect(
host = "localhost",
user = "your username",
password = "your password",
database = "mydatabase"
)

mycursor = mydb.cursor()
sql = "INSERT INTO customers (name, address) VALUES
    ('%s', '%s')"
val = ("John", "Highway 21")
mycursor.execute(sql, val)
mydb.commit()
print(mycursor.rowcount, "record inserted")
```

to select from a table:-

to select from a table in MySQL, use the "SELECT" statement.

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="your user name",
  password="your password",
  database="mydatabase")
mycursor = mydb.cursor()
```

```
mycursor.execute("SELECT * FROM customers")
myresult = mycursor.fetchall()
for x in myresult:
    print(x)
```