List of Experiments

Sr. No.	Name of Experiment	DOP	DOC	Marks	Sign
Basic Experiments					
1	To implement the basic data types and control structures in python.				
2	To implement functions and object oriented concepts in python.				
3	To implement File Handling in Python..				
4	To create a GUI with python containing different widgets.				
5	To implement menudriven programs for Link List, Stack and Queue in python				
6	To demonstrate CRUD(create,read,update,delete)operation on database using python.				
7	To create a web application using Django framework to demonstrate user login and registration.				
8	Write a program to implement Threading in python				
9	Write a program to demonstrate different numpy array creation techniques and different numpy methods				
10	Program to demonstrate use of numpy array for working with images				
11	Program to demonstrate Data Series using Pandas				
12	Program to demonstrate DataFrame using Pandas				
Project / Assignment					
8	Course Project: Project Title				
9	Assignment 1: Python basics				
10	Assignment 2: Advanced Python				
11	Assignment 3: Data Structure in Python				
12	Assignment 4: Python Integration Primer				

13	Assignment 5: Multithreading				
14	Assignment 6: NumPy and Pandas				
Formative Assessment					
15	Th - Quiz 1: Python basics				
16	Th - Quiz 2: Advanced Python				
17	Th - Quiz 3: Data Structure in Python				
18	Th - Quiz 4: Python Integration Primer				
19	Th - Quiz 5: Multithreading				
20	Th - Quiz 6: NumPy and Pandas				
21	Pr - Quiz 1 :Datatypes,functions				
22	Pr - Quiz 2: GUI				
22	Pr - Quiz 3: Linked List,Stack				
23	Pr - Quiz 4: Python with Django				
24	Pr - Quiz 5: Numpy				
25	Pr - Quiz 6: Pandas				

D.O.P: Date of performance

D.O.C : Date of correction

Experiment No 1:

Aim: To implement the basic data types and control structures in python.

Theory:

Python has the following data types built-in by default, in these categoriesText Type: Str

Numeric Types: int, float, complex

Sequence Types: list, tuple, range

Mapping Type: Dict

Set Types: set, frozenset

Boolean Type: Bool

Binary Types: bytes, bytearray, memoryview

Getting the Data Type

You can get the data type of any object by using the type() function:

Print the data type of the variable x:

```
x = 5
print(type(x))
```

Casting

There can be two types of Type Casting in Python –

- Implicit Type Casting
- Explicit Type Casting

Implicit Type Conversion

In this, methods, Python converts data type into another data type automatically. In this process, users don't have to involve in this process.

```
# Python program to demonstrate
# implicit type Casting
# Python automatically converts
# a to int


a = 7
print(type(a))
# Python automatically converts
# b to float
b = 3.0
print(type(b))
# Python automatically converts
# c to float as it is a float addition
c = a + b
print(c)
print(type(c))
# Python automatically converts
# d to float as it is a float multiplication
d = a * b
print(d)
print(type(d))
```

Output:

```
<class 'int'>
<class 'float'>
10.0
<class 'float'>
21.0
<class 'float'>
```

Explicit Type Casting

In this method, Python need user involvement to convert the variable data type into certain data type in order to the operation required.

Mainly in type casting can be done with these data type function:

• **Int() :** Int() function take float or string as an argument and return int type object. • **float() :** float() function take int or string as an argument and return float type object.

• **str() :** str() function take float or int as an argument and return string type object.

Let's see some example of type casting:

Type Casting int to float:

Here, we are casting integer object to float object with **float()** function.

```
# Python program to demonstrate

# type Casting

# int variable

a = 5

 # typecast to float

n = float(a)

print(n)

print(type(n))
```

Output:

```
5.0

<class 'float'>
```

Sequence data types

Python has 4 built in in data types used to store collections of data, the List,Tuple, Set, and Dictionary, all with different qualities and usage.

1.List: Lists are used to store multiple items in a single variable.

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

2.Tuple:A tuple is a collection which is ordered and **unchangeable**.

Tuples are written with round brackets.

thistuple = ("apple", "banana", "cherry")

print(thistuple)

3.Dictionary:A dictionary is a collection which is ordered*, changeable and do not allow duplicates.Dictionaries are written with curly brackets, and have keys and values:

```
thisdict = {

 "brand": "Ford",

 "model": "Mustang",

 "year": 1964

}

print(thisdict)
```

```
# Python3 program for
explaining # use of list, tuple,
set and

# dictionary

# Lists

l = []

 # Adding Element into list

l.append(5)

l.append(10)

print("Adding 5 and 10 in
 list", l) # Popping Elements
 from list

l.pop()

print("Popped one element from
list", l) print()
```

```
 # Set

s = set()

 # Adding element into set

s.add(5)

s.add(10)

print("Adding 5 and 10 in
 set", s) # Removing element
 from set

s.remove(5)

print("Removing 5 from
set", s) print()

 # Tuple

t = tuple(l)

 # Tuples are immutable

print("Tuple", t)

print()

 # Dictionary

d = {}

 # Adding the key value pair

d[5] = "Five"

d[10] = "Ten"

print("Dictionary", d)

# Removing key-value pair

del d[10]

print("Dictionary", d)
```

Control Structures in Python

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

1.While Loop

Syntax :

while expression:

statement(s)

2.For in Loop

Syntax:

for iterator_var in sequence:

statements(s)

3.Nested Loops

Syntax:

for iterator_var in sequence:

for iterator_var in sequence:

statements(s)

statements(s)

The syntax for a nested while loop statement in Python programming language is as follows:

while expression:

while expression:

statement(s)

statement(s)

Control Statements

1.Continue Statement

It returns the control to the beginning of the loop.

for i in range(0,10):

if (i==5)

break

print (i)

2. Break Statement

It brings control out of the loop

```
for i in range(0,10):
 if (i==5):
 break
 print (i)
```

2.Pass Statement

We use pass statement to write empty loops. Pass is also used for empty control statement, function and classes.

PROGRAM:

```
print("-----Program for Student Information-----")
D = dict()
n = int(input('How many student record you want to store?? '))
# Add student information
# to the dictionary
for i in range(0,n):
 x, y = input("Enter the complete name (First and last name) of student: ").split()
z = input("Enter contact number: ")
 m = input('Enter Marks: ')
 D[x, y] = (z, m)
# define a function for shorting
# names based on first name
def sort():
 ls = list()
 # fetch key and value using
 # items() method
 for sname,details in D.items():
 # store key parts as an tuple
 tup = (sname[0],sname[1])

 # add tuple to the list

 ls.append(tup)

 # sort the final list of tuples

ls = sorted(ls)

 for i in ls:

 # print first name and second name
print(i[0],i[1])

 return


# define a function for

# finding the minimum marks

# in stored data

def minmarks():

 ls = list()

 # fetch key and value using

 # items() methods

 for sname,details in D.items():

 # add details second element

 # (marks) to the list

 ls.append(details[1])


 # sort the list elemnts

 ls = sorted(ls)

 print("Minimum marks: ", min(ls))

 return


# define a function for searching
# student contact number

def searchdetail(fname):

 ls = list()

 for sname,details in D.items():

 tup=(sname,details)

 ls.append(tup)

 for i in ls:

 if i[0][0] == fname:

 print(i[1][0])

 return


# define a function for

# asking the options

def option():

 choice = int(input('Enter the operation detail: \n
\  1: Sorting using first name \n \

 2: Finding Minimum marks \n \

 3: Search contact number using first name: \n
\  4: Exit\n \

 Option: '))

if choice == 1:

 # function call

 sort()

 print('Want to perform some other operation??? Y or N: ')

 inp = input()

 if inp == 'Y':

 option()
```

```
# exit function call
exit()
  elif choice == 2:
minmarks()
print('Want to perform some other operation??? Y or N: ')
 inp = input()
if inp == 'Y':
option()
exit()
 elif choice == 3:
first = input('Enter first name of student: ')
searchdetail(first)
 print('Want to perform some other operation??? Y or N: ')
inp = input()
if inp == 'Y':
option()
exit()
else:
print('Thanks for executing me!!!!')
exit()
 ption()
```

Output

```
====RESTART: C:/Users/admin/AppData/Local/Programs/Python/Python310/exp1.py ===
-----Program for Student Information-----

How many student record you want to store?? 3

Enter the complete name (First and last name) of student: Ram Sharma

Enter contact number: 9850883323
Enter Marks: 87

Enter the complete name (First and last name) of student: Geeta
Varma Enter contact number: 9920771234

Enter Marks: 67

Enter the complete name (First and last name) of student: Seeta
Shukla Enter contact number: 8856646725

Enter Marks: 56

Enter the operation detail:

 1: Sorting using first name

 2: Finding Minimum marks

 3: Search contact number using first name:

 4: Exit

 Option: 1

Geeta Varma

Ram Sharma

Seeta Shukla

Want to perform some other operation??? Y or N:

Y

Enter the operation detail:

 1: Sorting using first name

 2: Finding Minimum marks

 3: Search contact number using first name:

 4: Exit

 Option: 2

Minimum marks: 56

Want to perform some other operation??? Y or N:

Y

Enter the operation detail:
```

```
1: Sorting using first name

2: Finding Minimum marks

3: Search contact number using first name:

4: Exit

Option: 3

Enter first name of student: Seeta

8856646725

Want to perform some other operation??? Y or N:N
```

Conclusion: the experiment effectively showcased the integration of essential data types and control structures within Python. By engaging in practical exercises, participants acquired proficiency in manipulating variables, employing loops, leveraging conditionals, and utilizing fundamental data structures.

Experiment No 2:

Aim: To implement functions and object oriented concepts in python.

Theory:

Python allows us to divide a large program into the basic building blocks known as a function. The function contains the set of programming statements enclosed in a block. A function can be called multiple times to provide reusability and modularity to the Python program.

Creating a Function

Python provides the **def** keyword to define the function. The syntax of the define function is given below.

Syntax:

1. **def** my_function(parameters):
2. function_block
3. **return** expression

Function Calling

In Python, after the function is created, we can call it from another function. A function must be defined before the function call; otherwise, the Python interpreter gives an error. To call the function, use the function name followed by the parentheses.

Recursive function

A function that calls itself is a recursive function. This method is used when a certain problem is defined in terms of itself. Although this involves iteration, using an iterative approach to solve such a problem can be tedious.

Classes and Objects

Classes are used to create user-defined data structures. Classes define functions called **methods**, which identify the behaviors and actions that an object created from the class can perform with its data.

While the class is the blueprint, an **instance** is an object that is built from a class and contains real data

Instantiate an Object in Python

Instance_name=ClassName()

Instance attributes and class attributes

Attributes created in .__init__() are called **instance attributes**. An instance attribute's value is specific to a particular instance of the class.

On the other hand, **class attributes** are attributes that have the same value for all class instances. You can define a class attribute by assigning a value to a variable name outside of .__init__().

The self

Class methods must have an extra first parameter in the method definition. We do not give a value for this parameter when we call the method, Python provides it. If we have a method that takes no arguments, then we still have to have one argument.

__init__ method

The __init__ method is similar to constructors in C++ and Java. Constructors are used to initializing the object's state.

Inheritance in Python:

Inheritance is the capability of a class to inherit all properties and methods of base class from which it is derived and can add new features to the class without modifying it. The syntax to define a derived class when one or more base classes are to be inherited is as follows:

class derivedClassName(baseClassName,…):

<statement 1>.
<statement n>

Different forms of Inheritance:

1. Single inheritance:

When a child class inherits from only one parent class, it is called single inheritance. We saw an example above.

2. Multiple inheritance:

When a child class inherits from multiple parent classes, it is called multiple inheritance.
Unlike java, python shows multiple inheritance.

3.**Multilevel inheritance**:

When we have a child and grandchild relationship.

4.Hierarchical inheritance

More than one derived classes are created from a single base.

PROGRAM:

```
# This is simplest Student data management program in python

# Create class "Student"

class Student:

 # Constructor

 def __init__(self, name, rollno, m1, m2):

 self.name = name

 self.rollno = rollno

 self.m1 = m1

 self.m2 = m2

 #ob = Student(name, rollno, m1, m2 )this will give error

   # Function to create and append new student

 def accept(self, Name, Rollno, marks1, marks2 ):

 # use ' int(input()) ' method to take input from user

 ob = Student(Name, Rollno, marks1, marks2 )

 ls.append(ob)

# Function to display student details

 def display(self, ob):

 print("Name : ", ob.name)

 print("RollNo : ", ob.rollno)

print("Marks1 : ", ob.m1)
print("Marks2 : ", ob.m2)

print("\n")

 # Search Function

def search(self, rn):

for i in range(ls.__len__()):

if(ls[i].rollno == rn):

return i

  # Delete Function

def delete(self, rn):

i = obj.search(rn)

del ls[i]

  # Update Function

def update(self, rn, No):

i = obj.search(rn)

roll = No

ls[i].rollno = roll;

 # Create a list to add Students

ls =[]

# an object of Student class

obj = Student('', 0, 0, 0)

 print("\nOperations used, ")

print("\n1.Accept Student details\n2.Display Student
Details\n"   "3.Search Details of a Student\n4.Delete Details of
Student"   "\n5.Update Student Details\n6.Exit")

 # ch = int(input("Enter choice:"))

# if(ch == 1):

obj.accept("A", 1, 100, 100)

obj.accept("B", 2, 90, 90) obj.accept("C", 3, 80, 80)
```

```
# elif(ch == 2):
print("\n")
print("\nList of Students\n")
for i in range(ls.__len__()):
 obj.display(ls[i])
  # elif(ch == 3):
print("\n Student Found, ")
s = obj.search(2)
obj.display(ls[s])
  # elif(ch == 4):
obj.delete(2)
print(ls.__len__())
print("List after deletion")
for i in range(ls.__len__()):
 obj.display(ls[i])
 # elif(ch == 5):
obj.update(3, 2)
print(ls.__len__())
print("List after updation")
for i in range(ls.__len__()):
 obj.display(ls[i])
  # else:
print("Thank You !")
```

OUTPUT

```
=RESTART:
C:\Users\admin\AppData\Local\Programs\Python\Python310\student_managment.py
Operations used,
1.Accept Student details

2.Display Student Details

3.Search Details of a Student

4.Delete Details of Student

5.Update Student Details

6.Exit

List of Students

Name : A

RollNo : 1

Marks1 : 100

Marks2 : 100

Name : B

RollNo : 2

Marks1 : 90

Marks2 : 90

Name : C

RollNo : 3

Marks1 : 80

Marks2 : 80

 Student
Found,
Name : B

RollNo : 2

Marks1 : 90

Marks2 : 90


2
```

```
List after
deletion
Name : A

RollNo : 1

Marks1
: 100
Marks2
: 100

Name : C

RollNo : 3

Marks1 : 80

Marks2 : 80

2

List after
updation
Name : A

RollNo : 1

Marks1
:    100
Marks2
:    100
Name  :
C

RollNo : 2

Marks1 : 80

Marks2 : 80

Thank You !
```

Conclusion: the implementation of functions and object-oriented concepts in Python proved to be essential in enhancing code modularity, reusability, and maintainability. By effectively utilizing these programming paradigms, we were able to achieve greater flexibility and scalability in our software design, laying a solid foundation for future development endeavors.

Experiment No 3:

Aim: To implement File Handling in Python.

Theory:

There are four different methods (modes) for opening a file:

" r" - Read - Default value. Opens a file for reading, error if the file does not

exist "a" - Append - Opens a file for appending, creates the file if it does not

exist "w" - Write - Opens a file for writing, creates the file if it does not exist "x"

- Create - Creates the specified file, returns an error if the file exists In addition

you can specify if the file should be handled as binary or text mode "t" - Text -

Default value. Text mode

"b" - Binary - Binary mode (e.g. images)

Python has a set of methods available for the file object.

Method Description

close() Closes the file

detach() Returns the separated raw stream from the buffer

fileno() Returns a number that represents the stream, from the operating system's perspective

flush() Flushes the internal buffer

isatty()Returns whether the file stream is interactive or not

read() Returns the file content

readable() Returns whether the file stream can be read or not

readline() Returns one line from the file

readlines() Returns a list of lines from the file

seek() Change the file position

seekable() Returns whether the file allows us to change the file position

tell() Returns the current file position truncate() Resizes the file to a specified size

writable() Returns whether the file can be written to or not

write() Writes the specified string to the file

writelines() Writes a list of strings to the file

PROGRAM:

Program 3 : Python program to copy odd noline from one file to other

```
# open file in read mode
fn = open('bcd.txt', 'r')


# open other file in write mode
fn1 = open('nfile.txt', 'w')


# read the content of the file line by line
cont = fn.readlines()
print(len(cont))
type(cont)
for i in range(0, len(cont)):
 if(i % 2 != 0):
 fn1.write(cont[i])
 else:
 pass


# close the file
fn1.close()


# open file in read mode
fn1 = open('nfile.txt', 'r')
```

```
# read the content of the file
cont1 = fn1.read()

# print the content of the file
print(cont1)




# close all files
fn.close()
fn1.close()
```

bcd.txt

```
hello how are you Line1
I am fine line2
Python Programming Line 3
Numpy
Pandas line5
```

OUTPUT:

```
I am fine line2
Numpy
```

Conclusion: The experiment successfully demonstrated the implementation of File Handling in Python, showcasing its versatility in reading, writing, and manipulating various file formats.

Experiment No:4

Aim: To create a GUI with python containing different widgets.

Theory:

. 1. Tkinter:

Tkinter is a standard package used for GUI programming in Python. This is built on top of the Tk interface.

2. PyQt:

PyQt is a Python toolkit binding of the Qt toolkit. Qt is a C++ framework that is used by Python to implement a cross-platform PyQt toolkit as a plug-in.

3. wxPython:

wxPython is also a cross-platform GUI toolkit. It is a wrapper for the API wxWidgets.

Python Tkinter Module

Tkinter is a standard Python library used for GUI programming. It provides an object-oriented interface to build the Tk GUI toolkit. It is a faster and easier way to build a GUI in Python.

An empty Tkinter top-level window can be created by using the following steps.

1. import the Tkinter module.
2. Create the main application window.
3. Add the widgets like labels, buttons, frames, etc. to the window.
4. Call the main event loop so that the actions can take place on the user's computer screen.

```
from tkinter import *

#creating the application main window.

top = Tk()

#Entering the event main loop

top.mainloop()
```

Python Tkinter Geometry

The Tkinter geometry specifies the method by using which, the widgets are represented on display. The python Tkinter provides the following geometry methods.

1. The pack() method

syntax

1. widget.pack(options)

A list of possible options that can be passed in pack() is given below.

- **expand:** If the expand is set to true, the widget expands to fill any space. o **Fill:** By default, the fill is set to NONE. However, we can set it to X or Y to determine whether the widget contains any extra space.
- **size:** it represents the side of the parent to which the widget is to be placed on the window.

2. The grid() method

The grid() geometry manager organizes the widgets in the tabular form. We can specify the rows and columns as the options in the method call. We can also specify the column span (width) or rowspan(height) of a widget.

This is a more organized way to place the widgets to the python application. The syntax to use the grid() is given below.

Syntax

1. widget.grid(options)

A list of possible options that can be passed inside the grid() method is given below.

- **Column**

  The column number in which the widget is to be placed. The leftmost column is represented by 0.

- **Columnspan**

  The width of the widget. It represents the number of columns up to which, the column is expanded.

- **ipadx, ipady**

  It represents the number of pixels to pad the widget inside the widget's border.

3. The place() method

The place() geometry manager organizes the widgets to the specific x and y coordinates.

Syntax

1. widget.place(options)

A list of possible options is given below.

- **Anchor:** It represents the exact position of the widget within the container. The default value (direction) is NW (the upper left corner)
- **bordermode:** The default value of the border type is INSIDE that refers to ignore the parent's inside the border. The other option is OUTSIDE.
- **height, width:** It refers to the height and width in pixels.
- **relheight, relwidth:** It is represented as the float between 0.0 and 1.0 indicating the fraction of the parent's height and width.
- **relx, rely:** It is represented as the float between 0.0 and 1.0 that is the offset in the horizontal and vertical direction.
- **x, y:** It refers to the horizontal and vertical offset in the pixels.**Tkinter widgets**

There are various widgets like button, canvas, checkbutton, entry, etc. that are used to build the python GUI applications.

	Widget					
	Button	Frame	Menu		PanedWindow	

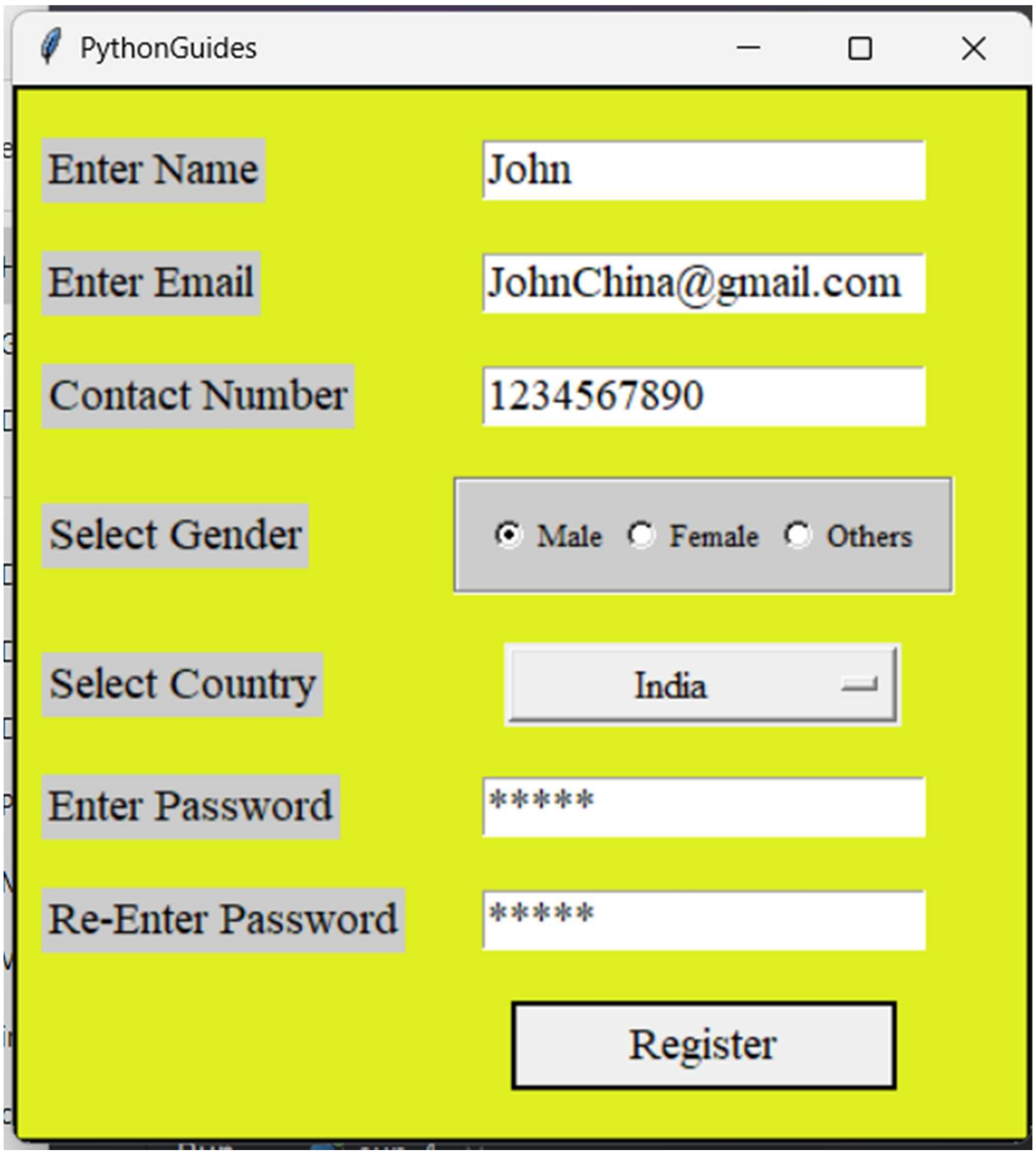
	Canvas	Label	Message	Text	LabelFrame	
	Checkbutton	ListBox	Radiob			
	Entry	Menubutton	Scale	Spinbo x		

PROGRAM

Program: To create registration form using tkniter module

```
from tkinter import *
ws = Tk()
ws.title('PythonGuides')
ws.config(bg='#0B5A81')
f = ('Times', 14)
var = StringVar()
var.set('male')
countries = []
variable = StringVar()
world = open('countries.txt', 'r')
for country in world:
 country = country.rstrip('\n')
 countries.append(country)
variable.set(countries[2])
right_frame = Frame(
 ws,
 bd=2,
 bg='#CCCCCC',
 relief=SOLID,
 padx=10,
 pady=10
 )
Label(
right_frame,
 text="Enter Name",
 bg='#CCCCCC',
 font=f
 ).grid(row=0, column=0, sticky=W,
pady=10) Label(
 right_frame,
 text="Enter Email",
 bg='#CCCCCC',
 font=f
 ).grid(row=1, column=0, sticky=W,
pady=10) Label(
 right_frame,
 text="Contact Number",
 bg='#CCCCCC',
 font=f
 ).grid(row=2, column=0, sticky=W, pady=10)
Label(
 right_frame,
 text="Select Gender",
 bg='#CCCCCC',
 font=f
 ).grid(row=3, column=0, sticky=W, pady=10)
Label(
 right_frame,
 text="Select Country",
 bg='#CCCCCC',
 font=f
 ).grid(row=4, column=0, sticky=W, pady=10)
Label(
 right_frame,
 text="Enter Password",
 bg='#CCCCCC',
 font=f
 ).grid(row=5, column=0, sticky=W, pady=10)
Label(
 right_frame,
 text="Re-Enter Password",
 bg='#CCCCCC',
 font=f
).grid(row=6, column=0, sticky=W, pady=10)
gender_frame = LabelFrame(
 right_frame,
 bg='#CCCCCC',
 padx=10,
 pady=10,
 )
register_name = Entry(
 right_frame,
 font=f
 )
register_email = Entry(
 right_frame,
 font=f
 )
register_mobile = Entry(
 right_frame,
 font=f
 )
male_rb = Radiobutton(
 gender_frame,
 text='Male',
 bg='#CCCCCC',
 variable=var,
 value='male',
 font=('Times', 10),
 )
female_rb = Radiobutton(
 gender_frame,
 text='Female',
 bg='#CCCCCC',
 variable=var,
 value='female',
 font=('Times', 10),
 )
others_rb = Radiobutton(
 gender_frame,
 text='Others',
 bg='#CCCCCC',
 variable=var,
 value='others',
 font=('Times', 10)
 )

register_country = OptionMenu(
 right_frame,
 variable,
 *countries)
register_country.config(
 width=15,
 font=('Times', 12)
)
register_pwd = Entry(
 right_frame,
 font=f,
 show='*'
)
pwd_again = Entry(
 right_frame,
 font=f,
 show='*'
)
register_btn = Button(
 right_frame,

 width=15,

 text='Register',

 font=f,

 relief=SOLID,

 cursor='hand2',

 command=None)

register_name.grid(row=0, column=1, pady=10, padx=20)
register_email.grid(row=1, column=1, pady=10, padx=20)
register_mobile.grid(row=2, column=1, pady=10,
padx=20) register_country.grid(row=4, column=1,
pady=10, padx=20) register_pwd.grid(row=5, column=1,
pady=10, padx=20) pwd_again.grid(row=6, column=1,
pady=10, padx=20) register_btn.grid(row=7, column=1,
pady=10, padx=20) right_frame.pack()

gender_frame.grid(row=3, column=1, pady=10,
padx=20) male_rb.pack(expand=True, side=LEFT)

female_rb.pack(expand=True, side=LEFT)

others_rb.pack(expand=True, side=LEFT)

ws.mainloop()
```

OUTPUT:


PythonGuides

Enter Name	John
Enter Email	JohnChina@gmail.com
Contact Number	1234567890
Select Gender	[✓ Male] Female Others
Select Country	India
Enter Password	*****
Re-Enter Password	*****

Conclusion: Through the implementation of various widgets in Python's GUI, we have successfully crafted an interactive user interface. This experiment underscores the versatility and functionality of Python for developing intuitive graphical applications

Experiment No:5

Aim: To implement menu driven programs for Link List, Stack and Queue in python

Theory:

A linked list is a sequential collection of data elements, which are connected together via links. A linked list consists of independent nodes containing any type of data and each node holds a reference or a link to the next node in the list.

The beginning node of a linked list is called the **head** and the end node is called the **tail.** All nodes of a linked list are independent and are not stored contagiously in memory.

Types of Linked Lists

There are 4 types of linked lists that can be created in python.

Singly Linked List

Circular Singly Linked List

Doubly Linked List

Circular Doubly Linked List

Stack Operations

Various operations can be performed on a stack in python.

Create Stack

Push

Pop

Peek

isEmpty

isFull

deleteStack

Queue Operations

Various operations can be performed on a queue in python.

Create Queue .

Enqueue

Dequeue

Peek

isEmpty

isFull

deleteQueue

PROGRAM

Program 5.1: Stack

```
# Program introduction statement
print("Simple STACK Data Structure Program")
# Initial empty STACK
stack = []
# Display Menu with Choices
while True:
 print("\nSELECT APPROPRIATE CHOICE")
 print("1. PUSH Element into the Stack")
 print("2. POP Element from the Stack")
 print("3. Display Elements of the Stack")
 print("4. Exit")
 choice = int(input("Enter the Choice:")) # Taking input from the user regarding choice  #
USER enter option 1 then PUSH elements into the STACK
 if choice == 1:
 # append() function to PUSH elements into the STACK
 stack.append("Monday") # PUSH element Monday
 stack.append("Tuesday") # PUSH element Tuesday
 stack.append("Wednesday") # PUSH element Wednesday
 stack.append("Thursday") # PUSH element Thursday
 stack.append("Friday") # PUSH element Friday
 stack.append("Saturday") # PUSH element Saturday

 stack.append("Sunday") # PUSH element Sunday

 stack.append('8') # PUSH element 8

 print('\nTotal 8 elements PUSH into the STACK')

 # USER enter option 2 then POP one element from the STACK

 elif choice == 2:

 if len(stack) == 0: # Check whether STACK is Empty or not  print('The
STACK is EMPTY No element to POP out')

 # Display this ERROR message if STACK is Empty

 else:

 # pop() function to POP element from the STACK in LIFO order
print('\nElement POP out from the STACK is:')

 print(stack.pop()) # Display the element which is POP out from the STACK

  # USER enter option 3 then display the STACK

 elif choice == 3:

 if len(stack) == 0: # Check whether STACK is Empty or not  print('The STACK is initially
EMPTY') # Display this message if STACK is Empty

else:

 print("The Size of the STACK is: ",len(stack)) # Compute the size of the STACK
print('\nSTACK elements are as follows:')

 print(stack) # Display all the STACK elements

 # User enter option 4 then EXIT from the program

 elif choice == 4:

 break

 # Shows ERROR message if the choice is not in between 1 to 4

 else:

 print("Oops! Incorrect Choice")
```

OUTPUT:

```
RESTART: C:/Users/admin/AppData/Local/Programs/Python/Python310/stack.py
============================================

Simple STACK Data Structure Program

SELECT APPROPRIATE CHOICE

1. PUSH Element into the Stack

2. POP Element from the Stack

3. Display Elements of the Stack

4. Exit

Enter the Choice:3

The STACK is initially EMPTY

SELECT APPROPRIATE CHOICE

1. PUSH Element into the Stack

2. POP Element from the Stack

3. Display Elements of the Stack

4. Exit

Enter the Choice:1

Total 8 elements PUSH into the STACK

SELECT APPROPRIATE CHOICE

1. PUSH Element into the Stack

2. POP Element from the Stack

3. Display Elements of the Stack

4. Exit
```

Program 5.2:Queue

```
# Import Python Package

from queue import Queue

# Program introduction statement
```

```
print("Simple QUEUE Data Structure Program")
# Initial empty QUEUE
queue = Queue();
# Display Menu with Choices
while True:
 print("\nSELECT APPROPRIATE CHOICE")
 print("1. PUT Element into the Queue")
 print("2. GET Element from the Queue")
 print("3. Display Elements of the Queue")
 print("4. Exit")
 choice = int(input("Enter the Choice:")) # Taking input from the user regarding choice
 # USER enter option 1 then PUT elements into the QUEUE
 if choice == 1:
 # put() function to PUT elements into the QUEUE
 queue.put("Monday") # PUT element Monday
 queue.put("Tuesday") # PUT element Tuesday
 queue.put("Wednesday") # PUT element Wednesday
 queue.put("Thursday") # PUT element Thursday
 queue.put("Friday") # PUT element Friday
 queue.put("Saturday") # PUT element Saturday
 queue.put("Sunday") # PUT element Sunday
 queue.put('8') # PUT element 8
 print('\nTotal 8 elements PUT into the QUEUE')
 # USER enter option 2 then GET one element from the QUEUE
elif choice == 2:
 if (queue.empty() == True): # Check whether QUEUE is Empty or not
print('The QUEUE is EMPTY No element to GET out')
 # Display this ERROR message if QUEUE is Empty  else:
```

```
 # get() function to GET element out from the QUEUE in FIFO order
print('\nElement GET out from the QUEUE is:')

 print(queue.get()) # Display the element which is GET out from the QUEUE   #
USER enter option 3 then display the QUEUE

 elif choice == 3:

 if (queue.empty() == True): # Check whether QUEUE is Empty or not  print('The
QUEUE is initially EMPTY')

 # Display this message if QUEUE is Empty

else:

 print("The Size of the QUEUE is: ",queue.qsize()) # Compute the size of the QUEUE
print('\nQUEUE elements are as follows:')

 print(list(queue.queue)) # Display all the QUEUE elements   # User enter option
4 then EXIT from the program

 elif choice == 4:

 break

 # Shows ERROR message if the choice is not in between 1 to 4

 else:

 print("Oops! Incorrect Choice")
```

OUTPUT:

```
=== RESTART: C:\Users\admin\AppData\Local\Programs\Python\Python310\Exp5.2.py ==
Simple QUEUE Data Structure Program

SELECT APPROPRIATE CHOICE

 1. PUT Element into the Queue

2. GET Element from the Queue

3. Display Elements of the Queue

4. Exit

Enter the Choice:3

The QUEUE is initially EMPTY

SELECT APPROPRIATE CHOICE

 1. PUT Element into the Queue
```

```
2. GET Element from the Queue
3. Display Elements of the Queue
4. Exit
Enter the Choice:1
Total 8 elements PUT into the QUEUE
SELECT APPROPRIATE CHOICE
1. PUT Element into the Queue
2. GET Element from the Queue
3. Display Elements of the Queue
4. Exit
```

Program 5.3:Linked List

```
# importing module
import collections
# Program introduction statement
print("Simple LINKED LIST Data Structure Program")
# initialising a deque() of arbitrary length to create Linked List
linked_lst = collections.deque()
# Display Menu with Choices
while True:
 print("\nSELECT APPROPRIATE CHOICE")
 print("1. INSERT elements into Linked List")
 print("2. INSERT elemnt at a Specific Position")
 print("3. Display all the elements of the Linked List ")
 print("4. DELETE the last element from the Linked List")
 print("5. DELETE the specific element from the Linked List")
 print("6. Exit")
 choice = int(input("Enter the Choice:")) # Taking input from the user regarding choice  #
```

```
USER enter option 1 then INSERT element in the Linked List

 if choice == 1:

 # append() function to fill deque() with elements and inserting into the Linked List
linked_lst.append("Monday") # INSERT element Monday

 linked_lst.append("Tuesday") # INSERT element Tuesday

 linked_lst.append("Wednesday") # INSERT element Wednesday
linked_lst.append("Sunday") # INSERT element Sunday

 print('\nTotal 4 elements INSERTED into the Linked List')

 # USER enter option 2 then INSERT element at a specific position in the Linked List
if choice == 2:

 # insert() function add element after the specified position in the Linked List
linked_lst.insert(3, 'Thursday')

 # INSERT element Thursday after 3rd element of Linked List  linked_lst.insert(5,
'Saturday')

 # INSERT element Saturday after 5th element of Linked List  linked_lst.insert(4,
'Friday')

 # INSERT element Friday after 4th element of Linked List  print('\nTotal 3 new
elements INSERTED at specific position in the Linked List')

 # USER enter option 3 then display the Linked List

 elif choice == 3:

 if len(linked_lst) == 0: # Check whether Linked List is Empty or not
print('The Linked List is initially EMPTY')

 # Display this message if Linked List is Empty  else:





print("The Size of the Linked List is: ",len(linked_lst))

 # Compute the size of the Linked List

 print('\nLinked List elements are as follows:')

 print(linked_lst)

 # USER enter option 4 then DELETE last element in the Linked List
elif choice == 4:
 if len(linked_lst) == 0: # Check whether Linked List is Empty or not  print('The
Linked List is EMPTY No element to DELETE')

 # Display this ERROR message if Linked List is Empty  else:

 # pop() function to DELETE last element in the Linked List

 print('\nLast element DELETED from the Linked List is:')

 print(linked_lst.pop())

 # Display the element which is Deleted from the Linked List

 # USER enter option 5 then DELETE the specific element from the Linked List
elif choice == 5:

 if len(linked_lst) == 0: # Check whether Linked List is Empty or not  print('The
Linked List is EMPTY No element to DELETE')

 # Display this ERROR message if Linked List is Empty  else:

 # remove() function to DELETE the specific element from the Linked List
print('\nSpecific element Monday DELETED from the Linked List')
linked_lst.remove('Monday') # Remove Monday from the Linked List

 # User enter option 6 then EXIT from the program

 elif choice == 6:

 break
```

OUTPUT:

```
===RESTART: C:/Users/admin/AppData/Local/Programs/Python/Python310/Exp5.3.py ==
Simple LINKED LIST Data Structure Program

SELECT APPROPRIATE CHOICE

1. INSERT elements into Linked List

2. INSERT elemnt at a Specific Position

3. Display all the elements of the Linked List

4. DELETE the last element from the Linked List

5. DELETE the specific element from the Linked List

6. Exit

Enter the Choice:1
```

```
Total 4 elements INSERTED into the Linked List
SELECT APPROPRIATE CHOICE
1. INSERT elements into Linked List
2. INSERT elemnt at a Specific Position
3. Display all the elements of the Linked List
4. DELETE the last element from the Linked List
5. DELETE the specific element from the Linked List
6. Exit
Enter the Choice:3
The Size of the Linked List is: 4
Linked List elements are as follows:
deque(['Monday', 'Tuesday', 'Wednesday', 'Sunday'])
SELECT APPROPRIATE CHOICE
1. INSERT elements into Linked List
2. INSERT elemnt at a Specific Position
3. Display all the elements of the Linked List
4. DELETE the last element from the Linked List
5. DELETE the specific element from the Linked List
6. Exit
```

Conclusion: the implementation of menu-driven programs for linked lists, stacks, and queues in Python has demonstrated their versatility and efficiency in managing data structures. Through this experiment, we have gained insights into the practical applications of these fundamental data structures, paving the way for further exploration and optimization in programming solutions.

Experiment No:6

Aim: To demonstrate CRUD(create,read,update,delete)operation on database using python.

Theory:

Python can be used to connect the Database.

MySQL is one of the most popular Databases.

Steps to work with the MySQL using Python.

1. Install MySQL Driver
2. Create a connection Object
3. Create a cursor Object
4. Execute the Query

Install MySQL Driver

1. python -m pip install mysql-connector-python

Create a Connection Object

The mysql.connector provides the **connect()** method used to create a connection between the MySQL database and the Python application. The syntax is given below.

Syntax:

1. Conn_obj= mysql.connector.connect(host = <hostname>, user = <username>, passwd = <password>,database=<database>)

Create a Cursor Object

The connection object is necessary to create because it provides the multiple working environments the same connection to the database. The **cursor()** function is used to create the cursor object. It is import for executing the SQL queries. The syntax is given below.

Syntax:

1. cursorobj= conn.cursor()

Execute the Query

Use the execute() method of the cursor object to execute the

query Cursorobj.execute(SQL statement)

Methods

Following are the various methods provided by the Cursor class/object. 1 callproc() :

2 close():

3 Info():

4 executemany():

5 execute():

6 fetchall()

7 fetchone()

8 fetchmany()

9 etchwarnings()

Properties

Following are the properties of the Cursor class –

1 column_names

2 description

3 lastrowid

4 rowcount

5 statement

PROGRAM

1. To create a database

```
import mysql.connector

mydb = mysql.connector.connect(
 host="localhost",
 user="myusername",
 password="mypassword"
)

mycursor = mydb.cursor()

mycursor.execute("CREATE DATABASE mydatabase")

#If this page is executed with no error, you have successfully created a database.
```

2.**To display Databases**

```
import mysql.connector

mydb = mysql.connector.connect(
 host="localhost",
 user="myusername",
 password="mypassword"
)

mycursor = mydb.cursor()

mycursor.execute("SHOW DATABASES")

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

OUTPUT

```
('information_scheme',)

('mydatabase',)

('performance_schema',)

('sys',)
```

3.Create table and insert values and update, delete and read the contents.

```
import mysql.connector

mydb = mysql.connector.connect(

 host="localhost",

 user="yourusername",

 password="yourpassword",

 database="mydatabase"

)

mycursor = mydb.cursor()

mycursor.execute("CREATE TABLE customers (id INT AUTO_INCREMENT PRIMARY
KEY, name VARCHAR(255), address VARCHAR(255))")


mycursor = mydb.cursor()

sql = "INSERT INTO customers (name, address) VALUES (%s,
%s)" val = [

  ('Peter', 'Lowstreet 4'),

 ('Amy', 'Apple st 652'),

   ('Vicky', 'Yellow Garden 2'),

 ('Ben', 'Park Lane 38'),

  ('William', 'Central st 954'),

 ('Chuck', 'Main Road 989'),

  ('Viola', 'Sideway 1633')

]

mycursor.executemany(sql, val)

mydb.commit()

print(mycursor.rowcount, "was inserted.")
```

```
sql1 = "SELECT * FROM customers WHERE address ='Park Lane

38'" mycursor.execute(sql1)


myresult = mycursor.fetchall()


for x in myresult:
  print(x)

sql2 = "UPDATE customers SET address = 'Canyon 123' WHERE address = 'Valley 345'"

mycursor.execute(sql2)


mydb.commit()

print(mycursor.rowcount, "record(s) affected")


sql3 = " DELETE FROM customers WHERE address = 'Mountain
21'" mycursor.execute(sql3)

mydb.commit()

print(mycursor.rowcount, "record(s) deleted")
```

OUTPUT

```
13 record was inserted.

(11, 'Ben', 'Park Lane 38')

1 record(s) affected

1 record(s) deleted
```

Conclusion: the experiment successfully showcased the fundamental CRUD operations - create, read, update, and delete - on a database using Python. Through systematic execution and analysis, it was evident that Python's intuitive syntax and powerful libraries offer efficient means to interact with databases, enabling seamless manipulation of data for various applications.

Experiment No 7

Aim: To create a web application using Django framework to demonstrate user login and registration.

Theory:

Django is a Python-based web framework which allows you to quickly create web application without all of the installation or dependency problems that you normally will find with other frameworks.

When you're building a website, you always need a similar set of components: a way to handle user authentication (signing up, signing in, signing out), a management panel for your website, forms, a way to upload files, etc. Django gives you ready-made components to use.

Django is based on **MVT (Model-View-Template)** architecture. MVT is a software design pattern for developing a web application.

MVT Structure has the following three parts –

Model: The model is going to act as the interface of your data. It is responsible for maintaining data. It is the logical data structure behind the entire application and is represented by a database (generally relational databases such as MySql, Postgres).

View: The View is the user interface — what you see in your browser when you render a website. It is represented by HTML/CSS/Javascript and Jinja files.

Template: A template consists of static parts of the desired HTML output as well as some special syntax describing how dynamic content will be inserted.

Project Structure :

A Django Project when initialized contains basic files by default such as manage.py, view.py, etc. A simple project structure is enough to create a single-page application. Here are the major files and their explanations. Inside the website folder (project folder) there will be the following files

manage.py- This file is used to interact with your project via the command line(start the server, sync the database… etc). For getting the full list of commands that can be executed by manage.py type this code in the command window

```
python manage.py help
```

folder (website) – This folder contains all the packages of your project. Initially, it contains four files –

- **_init_.py –** It is a python package. It is invoked when the package or a module in the package is imported. We usually use this to execute package initialization code, for example for the initialization of package-level data.
- **settings.py –** As the name indicates it contains all the website settings. In this file, we register any applications we create, the location of our static files, database configuration details, etc.
- **urls.py –** In this file, we store all links of the project and functions to call. • **wsgi.py** – This file is used in deploying the project in WSGI. It is used to help your Django application communicate with the webserver.

Installation of Django

Use command**:** pip install Django

Wait for the django to be downloaded and installed at the same time. Then After that type "python -m django version" and hit enter to check if django is installed and what version of django is.

```
python -m django version
```

Creating the App

After setting django we will now create the web app for the web server. First create a new folder named " RegistrationAndLogin", then cd to a newly created folder, then type "**django-admin startproject website**" and hit enter. A new folder will be created on the directory named 'website'

Running The Server

After creating a project, cd to the newly created directory, then type "manage.py runserver" and hit enter to start the server running. The "manage.py" is a command of django-admin that utilize the administrative tasks of python web framework.

Type '127.0.0.1:8000' in the url browser to view the server. When there is code changes in the server just (ctrl + C) to command prompt to stop the server from running, then start again to avoid errors.

Creating The Website

This time will now create the web app to display the web models. First locate the directory of the app via command prompt cd, then type "manage.py startapp web" and hit enter. A new directory will be create inside the app named "web". **Setting up The URL**

This time will now create a url address to connect the app from the server. First Go to website directory, then open urls via Python IDLE's or any text editor. Then import "include" module beside the url module and import additional module to make a redirect url to your site "from . import views". After that copy/paste the code below inside the urlpatterns.

```
url(r'^$', views.index_redirect, name='index_redirect'),
url(r'^web/', include('web.urls')),
```

It will be look like this:

```
from django.urls import include,re_path
from django.contrib import admin
from . import views

urlpatterns = [
re_path(r'^$', views.index_redirect, name='index_redirect'),
re_path(r'^web/', include('web.urls')),
re_path(r'^admin/', admin.site.urls),
]
```

Then after that create a view that will catch the redirect url. To do that create a file "views.py" then copy/paste the code below and save it as "views.py".

```
from django.shortcuts import redirect
from . import views

def index_redirect(request):
return redirect('/web/')
```

Creating The Path For The Pages

Now that we set the connect we will now create a path for the web pages. All you have to do first is to go to web directory, then copy/paste the code below and save it inside "web" directory named 'urls.py' The file name must be urls.py or else there will be an error in the code.

```
from django.urls import include,re_path
from . import views

urlpatterns = [
re_path(r'^$', views.index, name='index'),
re_path(r'^login/$', views.login, name='login'),
re_path(r'^home/$', views.home, name='home'),
]
```

Creating A Static Folder

This time we will create a directory that store an external file. First go to the web directory then create a directory called "static", after that create a sub directory called "web". You'll notice that it is the same as your main app directory name, to assure the absolute link. This is where you import the css, js, etc directory.

Creating The Views

The views contains the interface of the website. This is where you assign the html code for rendering it to django framework and contains a methods that call a specific functions. To do that first open the views.py, the copy/paste the code below.

```
from django.shortcuts import render, redirect, HttpResponseRedirect
from .models import Member
# Create your views here.

def index(request):
if request.method == 'POST':
member = Member(username=request.POST['username'],
password=request.POST['password'], firstname=request.POST['firstname'],
lastname=request.POST['lastname'])
member.save()
return redirect('/')
else:
return render(request, 'web/index.html')

def login(request):
return render(request, 'web/login.html')

def home(request):
if request.method == 'POST':
if Member.objects.filter(username=request.POST['username'],
password=request.POST['password']).exists():
member = Member.objects.get(username=request.POST['username'],
password=request.POST['password'])
return render(request, 'web/home.html', {'member': member})
else:
context = {'msg': 'Invalid username or password'}
return render(request, 'web/login.html', context)
```

Creating The Models

Now that we're done with the views we will then create a models. Models is module that will store the database information to django. To do that locate and go to web directory, then open the "models.py" after that copy/paste the code.

```
from django.db import models

# Create your models here


class Member(models.Model):
firstname=models.CharField(max_length=30)
lastname=models.CharField(max_length=30)
username=models.CharField(max_length=30)
password=models.CharField(max_length=12)

def __str__(self):
return self.firstname + " " + self.lastname
```

Registering The App To The Server

Now that we created the interface we will now then register the app to the server. To do that go to the website directory, then open "settings.py" via Python IDLE's or any text editor. Then copy/paste this script inside the INSTALLED_APP variables 'web'. It will be like this:

```
INSTALLED_APPS = [
'web',
'django.contrib.admin',
'django.contrib.auth',
'django.contrib.contenttypes',
'django.contrib.sessions',
'django.contrib.messages',
'django.contrib.staticfiles',
]
```

Creating The Mark up Language

Now we will create the html interface for the django framework. First go to web directory, then create a directory called "templates" and create a sub directory on it called web.

base.html

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8" name="viewport" content="width=device-width, initial scale=1"/>
{% load static %}
<link rel="stylesheet" type="text/css" href="{% static 'web/css/bootstrap.css'  %}" />
</head>
<body>
<nav class="navbar navbar-default">
<div class="container-fluid">
<a class="navbar-brand" href="give any link ">Experiment No 7</a>
   </div>
</nav>
<div class="col-md-3"></div>
<div class="col-md-6 well">
<h3 class="text-primary">Python - Simple Registration & Login Form</h3> <hr
style="border-top:1px dotted #000;"/>
{% block body %}
{% endblock %}
</div>
</body>
</html>
```

Save it as "base.html" inside the web directory "sub directory of templates". **index.html**

```
{% extends 'web/base.html' %}
{% block body %}
<form method="POST">
{% csrf_token %}
<div class="col-md-2">
<a href="{% url 'login' %}">Login</a>
</div>
<div class="col-md-8">
<div class="form-group">
<label for="username">Username</label>
<input type="text" name="username" class="form-control" required="required">
</div>
<div class="form-group">
<label for="password">Password</label>
<input type="password" name="password" class="form-control" required="required"/>
</div>
<div class="form-group">
<label for="firstname">Firstname</label>
<input type="text" name="firstname" class="form-control" required="required"/>
</div>
<div class="form-group">
<label for="lastname">Lastname</label>
<input type="text" name="lastname" class="form-control" required="required"/>
```

```
</div>
<br />
<div class="form-group">
<button type="submit" class="btn btn-primary form-control"><span class="glyphicon glyphicon-save"></span> Submit</button>
</div>
</div>
</form>
{% endblock %}
```

Save it as "index.html" inside the web directory "sub directory of

templates". **login.html**

```
{% extends 'web/base.html' %}
{% block body %}
<form method="POST" action="{% url 'home' %}">
{% csrf_token %}
<div class="col-md-2">
<a href="{% url 'index' %}">Signup</a>
</div>
<div class="col-md-8">
<div class="form-group">
<label for="username">Username</label>
<input type="text" name="username" class="form-control" required="required"/>
</div>
<div class="form-group">
<label for="password">Password</label>
<input type="password" name="password" class="form-control" required="required"/>
```

16. </div>

17. <center><label class="text-danger">{{ msg }}</label></center> 18.

19. <div class="form-group">

20. <button class="btn btn-primary form-control" type=”submit”><span
class="glyphicon glyphicon-log-in"></span> Login</button>

21. </div>

22. </div>

23. </form>

24.

25. {% endblock %}

Save it as "login.html" inside the web directory "sub directory of templates".

home.html

1. {% extends 'web/base.html' %}

2. {% block body %}

3. <h2>Welcome</h2>

4. {{ member.firstname }}

5. {% endblock %}

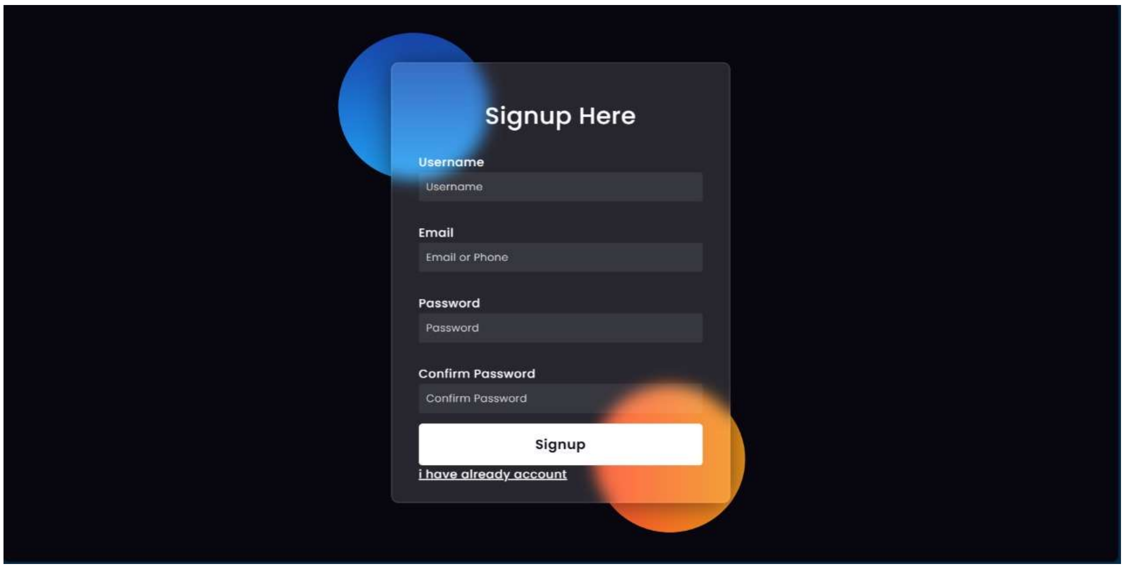
Save it as "home.html" inside the web directory "sub directory of

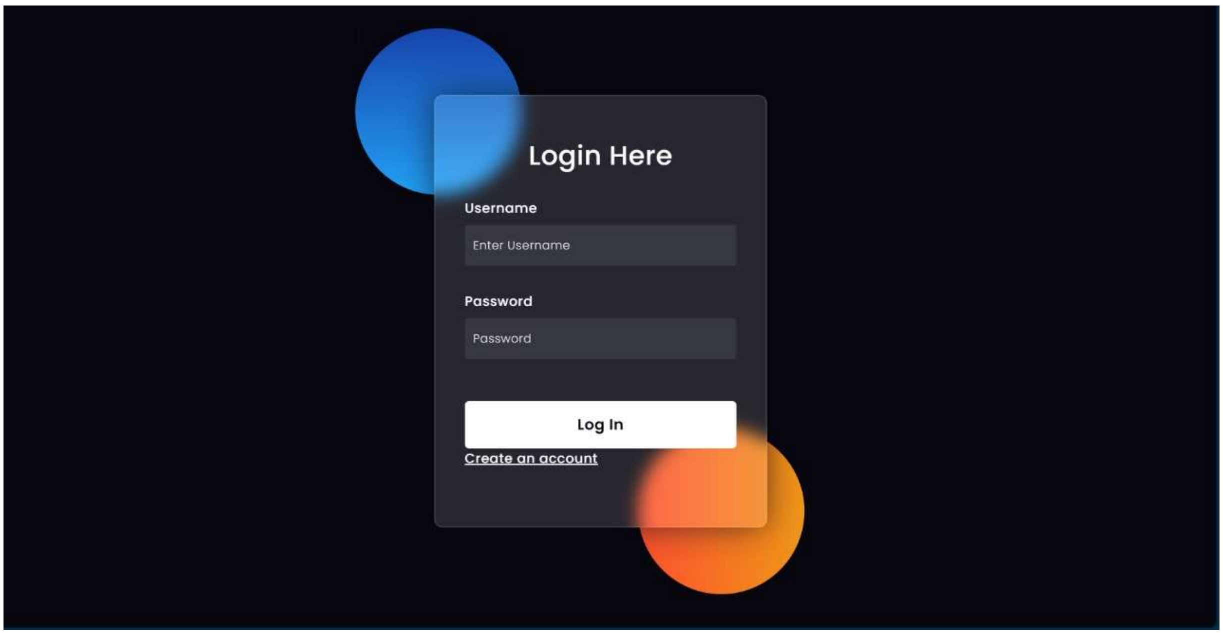
templates". **Migrating The App To The Server**

Now that we done in setting up all the necessary needed, we will now then make a migration and migrate the app to the server at the same time. To do that open the command prompt then cd to the "website" directory, then type "manage.py makemigrations" and hit enter. After that type again "manage.py migrate" then hit enter.

Now try to run the server again, using manage.py runserver and see if all things are done.

OUTPUT





Conclusion: the successful development of a web application using Django framework for user login and registration showcases its robustness and efficiency in handling authentication processes. This experiment underscores Django's capability in providing secure and user-friendly solutions for web development, highlighting its relevance in modern application development paradigms.

Experiment No: 8

Aim: Write a program to implement Threading in python.

Theory:

Multithreading is a threading technique in Python programming to run multiple threads concurrently by rapidly switching between threads with a CPU help (called context switching).
.

There are two main modules of multithreading used to handle threads in
Python. The thread module

The threading module

Thread modules

It is started with Python 3, designated as obsolete, and can only be accessed with **_thread** that supports backward compatibility.

Syntax:

1. thread.start_new_thread (function_name, args[, kwargs])

To implement the thread module in Python, we need to import a **thread** module and then define a function that performs some action by setting the target with a variable.

Threading Modules

The threading module is a high-level implementation of multithreading used to deploy an application in Python. To use multithreading, we need to import the threading module in Python Program.

Thread Class Methods

Methods Description	
start()	A start() method is used to initiate the activity of a thread. And it calls only once for each thread so that the execution of the thread can begin.

run()	A run() method is used to define a thread's activity and can be overridden by a class that extends the threads class.
join()	A join() method is used to block the execution of another code until the thread terminates.

Follow the given below steps to implement the threading module in Python Multithreading:

1. Import the threading module

Create a new thread by importing the **threading** module, as shown.

Syntax:

1. **import** threading

A **threading** module is made up of a **Thread** class, which is instantiated to create a Python thread.

2. Declaration of the thread parameters: It contains the target function, argument, and **kwargs** as the parameter in the **Thread()** class.

- **Target**: It defines the function name that is executed by the thread.
- **Args**: It defines the arguments that are passed to the target function name.

Start a new thread: To start a thread in Python multithreading, call the thread class's object. The start() method can be called once for each thread object; otherwise, it throws an exception error.

Syntax:

1. t1.start()
2. t2.start()

4. Join method: It is a join() method used in the thread class to halt the main thread's execution and waits till the complete execution of the thread object. When the thread object is completed, it starts the execution of the main thread in Python.

5. Synchronizing Threads in Python

It is a thread synchronization mechanism that ensures no two threads can simultaneously execute a particular segment inside the program to access the shared resources. The situation may be termed as critical sections. We use a race condition to avoid the critical section condition, in which two threads do not access resources at the same time.

PROGRAM

```
import threading
import time
exitFlag = 0
class myThread (threading.Thread):
 def __init__(self, threadID, name, counter):
 threading.Thread.__init__(self)
 self.threadID = threadID
 self.name = name
 self.counter = counter
 def run(self):
 print ("Starting " + self.name)
 print_time(self.name, self.counter, 5)
 print ("Exiting " + self.name)
def print_time(threadName, delay, counter):
 while counter:
 if exitFlag:
 threadName.exit()
 time.sleep(delay)
 print ("%s: %s" % (threadName, time.ctime(time.time())))
 counter -= 1
# Create new threads
```

```
thread1 = myThread(1, "Thread-1", 1)

thread2 = myThread(2, "Thread-2", 2)

# Start new Threads

thread1.start()

thread2.start()

thread1.join()

thread2.join()

print ("Exiting Main Thread")
```

OUTPUT:

```
= RESTART: C:/Users/admin/AppData/Local/Programs/Python/Python310/threadingexp.py
Starting Thread-1Starting Thread-2

Thread-1: Fri Apr 22 23:33:53 2022

Thread-2: Fri Apr 22 23:33:54 2022

Thread-1: Fri Apr 22 23:33:54 2022

Thread-1: Fri Apr 22 23:33:55 2022

Thread-2: Fri Apr 22 23:33:56 2022

Thread-1: Fri Apr 22 23:33:56 2022

Thread-1: Fri Apr 22 23:33:57 2022

Exiting Thread-1

Thread-2: Fri Apr 22 23:33:58 2022

Thread-2: Fri Apr 22 23:34:00 2022

Thread-2: Fri Apr 22 23:34:02 2022

Exiting Thread-2

Exiting Main Thread
```

Conclusion: The experiment successfully demonstrated the implementation of threading in Python, showcasing its ability to execute multiple tasks concurrently and improve program efficiency. Through this exercise, the benefits of threading in enhancing performance and resource utilization were clearly evident, emphasizing its importance in modern software development.

Experiment No: 9

Aim: Write a program to demonstrate different numpy array creation techniques and different numpy methods

NumPy stands for Numerical Python. It is a Python library used for working with an array. In Python, we use the list for purpose of the array but it's slow to process. NumPy array is a powerful N-dimensional array object and its use in linear algebra, Fourier transform, and random number capabilities. It provides an array object much faster than traditional Python lists.

Types of Array:

1. One Dimensional Array
2. Multi-Dimensional Array

One Dimensional Array:

A one-dimensional array is a type of linear array.

```
# importing numpy module
import numpy as np
# creating list
list = [1, 2, 3, 4]
# creating numpy array
sample_array = np.array(list1)
print("List in python : ", list)
print("Numpy Array in python :",
 sample_array)
```

Output:

```
List in python : [1, 2, 3, 4]
Numpy Array in python : [1 2 3 4]
```

Check data type for list and array:

```
print(type(list_1))
print(type(sample_array))
```

Output:

```
<class 'list'>
<class 'numpy.ndarray'>
```

Multi-Dimensional Array:

Data in multidimensional arrays are stored in tabular form.

Two Dimensional Array

```
# importing numpy module
import numpy as np
# creating list
list_1 = [1, 2, 3, 4]
list_2 = [5, 6, 7, 8]
list_3 = [9, 10, 11, 12]
# creating numpy array
sample_array = np.array([list_1,
 list_2,
 list_3])
print("Numpy multi dimensional array in python\n",
 sample_array)
```

Output:

```
Numpy multi dimensional array in python
[[ 1 2 3 4]
```

```
 [ 5 6 7 8]

 [ 9 10 11 12]]
```

Note: use **[ ]** operators inside numpy.array() for multi-dimensional

Anatomy of an array :

1. Axis: The Axis of an array describes the order of the indexing into the array. *Axis 0 = one dimensional*

Axis 1 = Two dimensional

Axis 2 = Three dimensional

2.Shape: The number of elements along with each axis. It is from a tuple. **3. Rank:** The rank of an array is simply the number of axes (or dimensions) it has. **The one-dimensional array has rank 1.**

Rank 1

The two-dimensional array has rank 2.

Rank 2

3. **Data type objects (dtype):** Data type objects (dtype) is an instance of **numpy.dtype** class. It describes how the bytes in the fixed-size block of memory corresponding to an array item should be interpreted.

Some different way of creating Numpy Array :

1. numpy.array(): The Numpy array object in Numpy is called **ndarray.** We can create ndarray using **numpy.array()** function.

Syntax: *numpy.array(parameter)*

```
import numpy as np

arr = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

print(arr)
```

2. numpy.fromiter(): The fromiter() function create a new one-dimensional array from an iterable object.

Syntax: *numpy.fromiter(iterable, dtype, count=-1)*

3. numpy.arange(): This is an inbuilt NumPy function that returns evenly spaced values within a given interval.

Syntax: *numpy.arange([start,]stop, [step,]dtype=None)*

numpy.linspace(): This function returns evenly spaced numbers over a specified between two limits.

Syntax: *numpy.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None, axis=0)*

5. numpy.empty(): This function create a new array of given shape and type, without initializing value.

Syntax: *numpy.empty(shape, dtype=float, order='C')*

6. numpy.ones(): This function is used to get a new array of given shape and type, filled with ones(1).

Syntax: *numpy.ones(shape, dtype=None, order='C')*

7. numpy.zeros(): This function is used to get a new array of given shape and type, filled with zeros(0).

Syntax: *numpy.ones(shape, dtype=None)*

PROGRAM:

```
# Python program to demonstrate

# array creation techniques and different array methods.

import numpy as np


# Creating array from list with type float

a = np.array([[1, 2, 4], [5, 8, 7]], dtype = 'float')

print ("Array created using passed list:\n", a)


# Creating array from tuple

b = np.array((1 , 3, 2))

print ("\nArray created using passed tuple:\n", b)


# Creating a 3X4 array with all zeros
```

```
c = np.zeros((3, 4))

print ("\nAn array initialized with all zeros:\n", c)


# Create a constant value array of complex type

d = np.full((3, 3), 6, dtype = 'complex')

print ("\nAn array initialized with all 6s."

 "Array type is complex:\n", d)


# Create an array with random values

e = np.random.random((2, 2))

print ("\nA random array:\n", e)




# Create a sequence of integers

# from 0 to 30 with steps of 5

f = np.arange(0, 30, 5)

print ("\nA sequential array with steps of
5:\n", f)

# Create a sequence of 10 values in range 0
to 5 g = np.linspace(0, 5, 10)

print ("\nA sequential array with 10 values
between"  "0 and 5:\n", g)

# Reshaping 3X4 array to 2X2X3 array

arr = np.array([[1, 2, 3, 4],

 [5, 2, 4, 2],

 [1, 2, 0, 1]])


newarr = arr.reshape(2, 2, 3)
```

```
print ("\nOriginal array:\n", arr)

print ("Reshaped array:\n", newarr)


# Flatten array

arr = np.array([[1, 2, 3], [4, 5, 6]])

flarr = arr.flatten()


print ("\nOriginal array:\n", arr)

print ("Fattened array:\n", flarr)

a = np.array([[1, 4, 2],

 [3, 4, 6],

 [0, -1, 5]])


# sorted array

print ("Array elements in sorted
order:\n",  np.sort(a, axis = None))


# sort array row-wise

print ("Row-wise sorted array:\n",

 np.sort(a, axis = 1))


# specify sort algorithm

print ("Column wise sort by applying merge-
sort:\n",  np.sort(a, axis = 0, kind = 'mergesort'))

arr = np.array([1, 2, 3, 4, 5])

x = arr.copy()

y = arr.view()
```

```
print(x.base)

print(y.base)
```

OUTPUT

```
= RESTART: C:/Users/admin/AppData/Local/Programs/Python/Python310/numpyexp1.py =
Array created using passed list:

 [[1. 2. 4.]

 [5. 8. 7.]]

Array created using passed tuple:

 [1 3 2]

An array initialized with all zeros:

 [[0. 0. 0. 0.]

 [0. 0. 0. 0.]

 [0. 0. 0. 0.]]

An array initialized with all 6s.Array type is complex:

 [[6.+0.j 6.+0.j 6.+0.j]

 [6.+0.j 6.+0.j 6.+0.j]

 [6.+0.j 6.+0.j 6.+0.j]]

A random array:

 [[0.80415936 0.89933488]

 [0.4431535 0.83914627]]

A sequential array with steps of 5:

 [ 0 5 10 15 20 25]

A sequential array with 10 values between0 and 5:

 [0. 0.55555556 1.11111111 1.66666667 2.22222222 2.77777778

 3.33333333 3.88888889 4.44444444 5. ]

Original array:
```

```
[[1 2 3 4]
[5 2 4 2]
[1 2 0 1]]
Reshaped array:
[[[1 2 3]
[4 5 2]]
[[4 2 1]
[2 0 1]]]
Fattened array:
[1 2 3 4 5 6]
Array elements in sorted order:
[-1 0 1 2 3 4 4 5 6]
Row-wise sorted array:
[[ 1 2 4]
[ 3 4 6]
[-1 0 5]]
Column wise sort by applying merge-
sort: [[ 0 -1 2]
[ 1 4 5]
[ 3 4 6]]
None
[1 2 3 4 5]
```

Conclusion: We successfully showcased diverse techniques for creating numpy arrays and employing various numpy methods. This demonstration not only illustrated the flexibility and efficiency of numpy in array manipulation but also highlighted its utility in facilitating complex mathematical operations and data analysis tasks with ease.

Experiment No 10:

Aim: Program to demonstrate use of numpy array for working with

images

Theory

Images are an easier way to represent the working model. In Machine Learning, Python uses the image data in the format of Height, Width, Channel format. i.e. Images are converted into Numpy Array in Height, Width, Channel format.

Modules Needed:

NumPy: By default in higher versions of Python like 3.x onwards, NumPy is available and if not available(in lower versions), one can install by using

```
pip install numpy
```

Pillow: This has to be explicitly installed in later versions too. It is a preferred image manipulation tool. In Python 3, Pillow python library which is nothing but the upgradation of PIL only. It can be installed using

```
pip install Pillow
```

One can easily check the version of installed Pillow by using the below code

```
  import PIL


print('Installed Pillow Version:', PIL.__version__)
```

Output:

```
Installed Pillow Version: 7.2.0
```

Loading the images via Pillow Library

Let us check for an image that is in the PNG or JPEG format. The image can be referred via its path. Image class is the heart of PIL. It has open() function which opens up an image and digital file format can be retrieved as well as pixel format.

Converting an image into NumPy Array

Python provides many modules and API's for converting an image into a NumPy array.
Let's discuss a few of them in detail.

Using NumPy module

Numpy module in itself provides various methods to do the same. These methods are –

Method 1: Using asarray() function

asarray() function is used to convert PIL images into NumPy arrays. This function converts the input to an array

Method 2: Using numpy.array() function

By using numpy.array() function which takes an image as the argument and converts to NumPy array

In order to get the value of each pixel of the NumPy array image, we need to print the retrieved data that got either from asarray() function or array() function.

PROGRAM:

```
from PIL import Image

from numpy import as array

# load the image

image = Image.open('demopic.jpg')

# convert image to numpy array

data = asarray(image)

print(type(data))

# summarize shape

print(data.shape)

print(data)

image2 = Image.fromarray(data)

print(type(image2))

# summarize image details

print(image2.mode)
print(image2.size)
```

OUTPUT:

```
=RESTART: C:\Users\admin\AppData\Local\Programs\Python\Python310\img2.py
==============================================

<class 'numpy.ndarray'>

(377, 271, 3)

[[[169 180 114]

 [167 177 114]

 [167 173 113]

 ...

 [152 133 93]

 [151 132 92]

 [150 130 93]]

 [[161 171 110]

 [160 170 110]

 [158 166 109]

 ...

 [151 132 92]

 [149 130 90]

 [148 128 91]]

 [[150 159 106]

 [148 156 105]

 [147 153 105]

 ...

 [152 133 93]

 [150 131 91]

 [149 131 93]]
...

 [[184 124 72]
 [182 125 72]
 [180 122 72]
 ...
 [241 221 13]
 [240 222 16]
 [241 223 19]]

 [[181 121 69]
 [225 208 4]
 [222 203 3]]

 [[182 122 70]
 [181 124 71]
 [180 122 72]
 ...
 [222 205 4]
 [222 206 5]
 [219 201 5]]]
<class
'PIL.Image.Image'>
JPEG

RGB

(271, 377)
```

Conclusion: , the experiment effectively showcased the practical application of numpy arrays in image manipulation, underscoring their efficiency and versatility in processing visual data. By implementing fundamental numpy functions, the experiment demonstrated how these arrays can be harnessed to perform various image operations with ease and precision, providing a solid foundation for further exploration and development in the field of image processing.

Experiment No: 11

Aim: Program to demonstrate Data Series using Pandas

Theory:

Pandas is an open-source library that is made mainly for working with relational or labeled data both easily and intuitively. It provides various data structures and operations for manipulating numerical data and time series. This library is built on top of the NumPy library. Pandas is fast and it has high performance & productivity for users.

After the pandas have been installed into the system, you need to import the library. This module is generally imported as:

import pandas as pd

Here, pd is referred to as an alias to the Pandas.

Pandas generally provide two data structures for manipulating data, They are:

- **Series**
- **DataFrame**

Series:

Pandas Series is a one-dimensional labelled array capable of holding data of any type (integer, string, float, python objects, etc.).

The axis labels are collectively called indexes.

Pandas Series is nothing but a column in an excel sheet.

Creating an empty Series :

A basic series, which can be created is an Empty Series.

```
# import pandas as pd

import pandas as pd

# Creating empty series

ser = pd.Series()

print(ser)
```

Creating a series from array:

In order to create a series from array, we have to import a numpy module and have to use array() function.

```
# import pandas as pd
import pandas as pd
 # import numpy as np
import numpy as np
# simple array
data = np.array(['g', 'e', 'e', 'k', 's'])
  ser = pd.Series(data)
print(ser)
```

Creating a series from array with index :

In order to create a series from array with index, we have to provide index with same number of element as it is in array.

```
# import pandas as pd
import pandas as pd
# import numpy as np
import numpy as np
# simple array
data = np.array(['g', 'e', 'e', 'k', 's'])
# providing an index
ser = pd.Series(data, index =[10, 11, 12, 13, 14])
print(ser)
```

Creating a series from Lists:

In order to create a series from list, we have to first create a list after that we can create a series from list.

```
import pandas as pd
```

```
 # a simple list
list = ['g', 'e', 'e', 'k', 's']


# create series form a list
ser = pd.Series(list)
print(ser)
```

Creating a series from Dictionary:

In order to create a series from dictionary, we have to first create a dictionary after that we can make a series using dictionary. Dictionary key are used to construct a index.

```
import pandas as pd


# a simple dictionary
ict = {'Geeks' : 10,
 'for' : 20,
 'geeks' : 30}


# create series from dictionary
ser = pd.Series(dict)


print(ser)
```

Creating a series from Scalar value:

In order to create a series from scalar value, an index must be provided. The scalar value will be repeated to match the length of index.

```
import pandas as pd
import numpy as np
```

```
# giving a scalar value with index

ser = pd.Series(10, index =[0, 1, 2, 3, 4, 5])

print(ser)
```

Creating a series using NumPy functions :

In order to create a series using numpy function, we can use different function of numpy like numpy.linspace(), numpy.random.radn().

```
# import pandas and numpy

import pandas as pd

import numpy as np


# series with numpy linspace()

ser1 = pd.Series(np.linspace(3, 33, 3))

print(ser1)


# series with numpy linspace()

ser2 = pd.Series(np.linspace(1, 100, 10))

print("\n", ser2)
```

PROGRAM:

```
import pandas as pd

import matplotlib.pyplot as plt


author = ['Jitender', 'Purnima', 'Arpit',
'Jyoti'] article = [210, 211, 114, 178]


auth_series = pd.Series(author)

article_series = pd.Series(article)
```

```
frame = { 'Author': auth_series, 'Article':
article_series }

result = pd.DataFrame(frame)

age = [21, 21, 24, 23]


result['Age'] = pd.Series(age)


result.plot.bar()

plt.show()
```

Conclusion: the experiment successfully showcased the utility of Pandas in handling and manipulating data series effectively. Through various operations and analyses, Pandas demonstrated its capability to streamline data management tasks, providing researchers and analysts with a powerful tool for data exploration and interpretation.

Experiment No 12:

Aim: Program to demonstrate DataFrame using Pandas

Theory:

Pandas DataFrame is a two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns). A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns. Pandas DataFrame consists of three principal components, the data, rows, and columns.

Creating an empty dataframe :

A basic DataFrame, which can be created is an Empty Dataframe. An Empty Dataframe is created just by calling a dataframe constructor.

```
# import pandas as pd

import pandas as pd

# Calling DataFrame constructor

df = pd.DataFrame()

print(df)
```

Output :

```
Empty DataFrame

Columns: []

Index: []
```

Creating a dataframe using List:

DataFrame can be created using a single list or a list of lists.

```
# import pandas as pd

import pandas as pd
```

```
# list of strings
lst = ['Geeks', 'For', 'Geeks', 'is',
 'portal', 'for', 'Geeks']
# Calling DataFrame constructor on list
df = pd.DataFrame(lst)
print(df)
```

Creating DataFrame from dict of ndarray/lists:

To create DataFrame from dict of narray/list, all the narray must be of same length. If index is passed then the length index should be equal to the length of arrays. If no index is passed, then by default, index will be range(n) where n is the array length.

```
# Python code demonstrate creating
# DataFrame from dict narray / lists
# By default addresses.
import pandas as pd
# initialise data of lists.
data = {'Name':['Tom', 'nick', 'krish', 'jack'], 'Age':[20, 21, 19, 18]}
# Create DataFrame
df = pd.DataFrame(data)
# Print the output.
print(df)
```

Create pandas dataframe from lists using dictionary:

Creating pandas data-frame from lists using dictionary can be achieved in different ways. We can create pandas dataframe from lists using dictionary using pandas.DataFrame. With this method in Pandas we can transform a dictionary of list to a dataframe.

```
# importing pandas as pd
import pandas as pd
# dictionary of lists
dict = {'name':["aparna", "pankaj", "sudhir", "Geeku"],
 'degree': ["MBA", "BCA", "M.Tech", "MBA"],
 'score':[90, 40, 80, 98]}
df = pd.DataFrame(dict)
print(df)
```

Dataframe methods

Few methods of Dataframe are mentioned below:

1. Pandas **head()** method is used to return top n (5 by default) rows of a data frame or series.

2. Pandas **describe()** is used to view some basic statistical details like percentile, mean, std etc. of a data frame or a series of numeric values.

3.Pandas **tail()** method is used to return bottom n (5 by default) rows of a data frame or series

4.query():Pandas provide many methods to filter a Data frame and **Dataframe.query()** is one of them.

5. Pandas provide a unique method to retrieve rows from a Data frame.DataFrame.loc[] method is used to retrieve rows from Pandas DataFrame. Rows can also be selected by passing integer location to an iloc[] function.

6.drop() method is used to delete columns or rows of a dataframe.

PROGRAM:

Program 12.1:Program to query dataframe

```
import pandas as pd


df = pd.DataFrame([[10, 20, 30, 40], [70, 14, 21, 80],
 [55, 15, 80, 12]],


 columns=['GFG_USER_1', 'GFG_USER_2',
 'GFG_USER_3', 'GFG_USER_4'],


 index=['Practice1', 'Practice2', 'Practice3'])


print(df, "\n")


# Filter data using query method
df1 = df.loc[df.query(
 'GFG_USER_1 <= 80 & GFG_USER_2 > 10 & \
 GFG_USER_3 < 50 & GFG_USER_4 == 80').index]


print(df1)
```

Conclusion: The experiment successfully showcased the versatility and efficiency of Pandas DataFrame for data manipulation and analysis. Through its intuitive functionality and comprehensive features, Pandas proved to be an indispensable tool for handling structured data, offering researchers and analysts powerful capabilities for exploring and visualizing datasets with ease.