Name: Meet Raut
Batch: S21
Roll number: 2201084

PYTHON

EXPERIMENT: 1-A

Aim: To write a program to implement compound structures like Lists, Tuples, Dictionaries and Sets.

Objective Mapped: LO1

Theory:

In Python, compound data types are fundamental structures that allow programmers to organize, manipulate, and represent data in various ways. These data types, including lists, dictionaries, tuples, and sets, provide versatile tools for handling complex datasets and solving a wide range of programming problems.

1. Lists:

Lists are one of the most commonly used data structures in Python. They are ordered collections of items, where each item can be of any data type, and they can even contain other lists (nested lists). Lists are mutable, meaning their elements can be modified after creation. This flexibility makes lists ideal for tasks such as storing sequences of data, managing dynamic collections, and facilitating iteration and manipulation of elements. With operations like appending, extending, slicing, and sorting, lists provide powerful tools for data organization and manipulation.

2. Dictionaries:

Dictionaries are another essential compound data type in Python. Unlike lists, which are indexed by integer values, dictionaries are indexed by keys, allowing for fast lookup and retrieval of values based on unique identifiers. Each element in a dictionary is a key-value pair, where the key is immutable (typically a string or number) and the value can be of any data type. Dictionaries are highly efficient for mapping relationships between items, representing structured data, and implementing associative arrays. They support operations such as adding, updating, and deleting key-value pairs, making them invaluable for tasks like data modelling, configuration management, and rapid information retrieval.

3. Tuples:

Tuples are ordered collections similar to lists but with one crucial difference: they are immutable, meaning their elements cannot be modified after creation. Tuples are typically used to group together related data elements that should not be changed, such as coordinates, records, or function arguments. While tuples lack the extensive mutability of lists, their immutability provides benefits such as improved performance, data integrity, and protection against unintended modifications. Tuples support operations like indexing, slicing, and unpacking, making them versatile for tasks like data serialization, function return values, and representing fixed sequences of data.

4. Sets:

Sets are unordered collections of unique elements, designed to support mathematical set operations like union, intersection, and difference. Sets are mutable, allowing for dynamic modification of elements, but each element within a set must be hashable and immutable to ensure uniqueness. Sets are particularly useful for tasks requiring membership testing, duplicate removal, or filtering unique elements from a dataset. With operations like adding, removing, and testing for membership, sets provide efficient tools for data deduplication, filtering, and set-based computations.

Code:

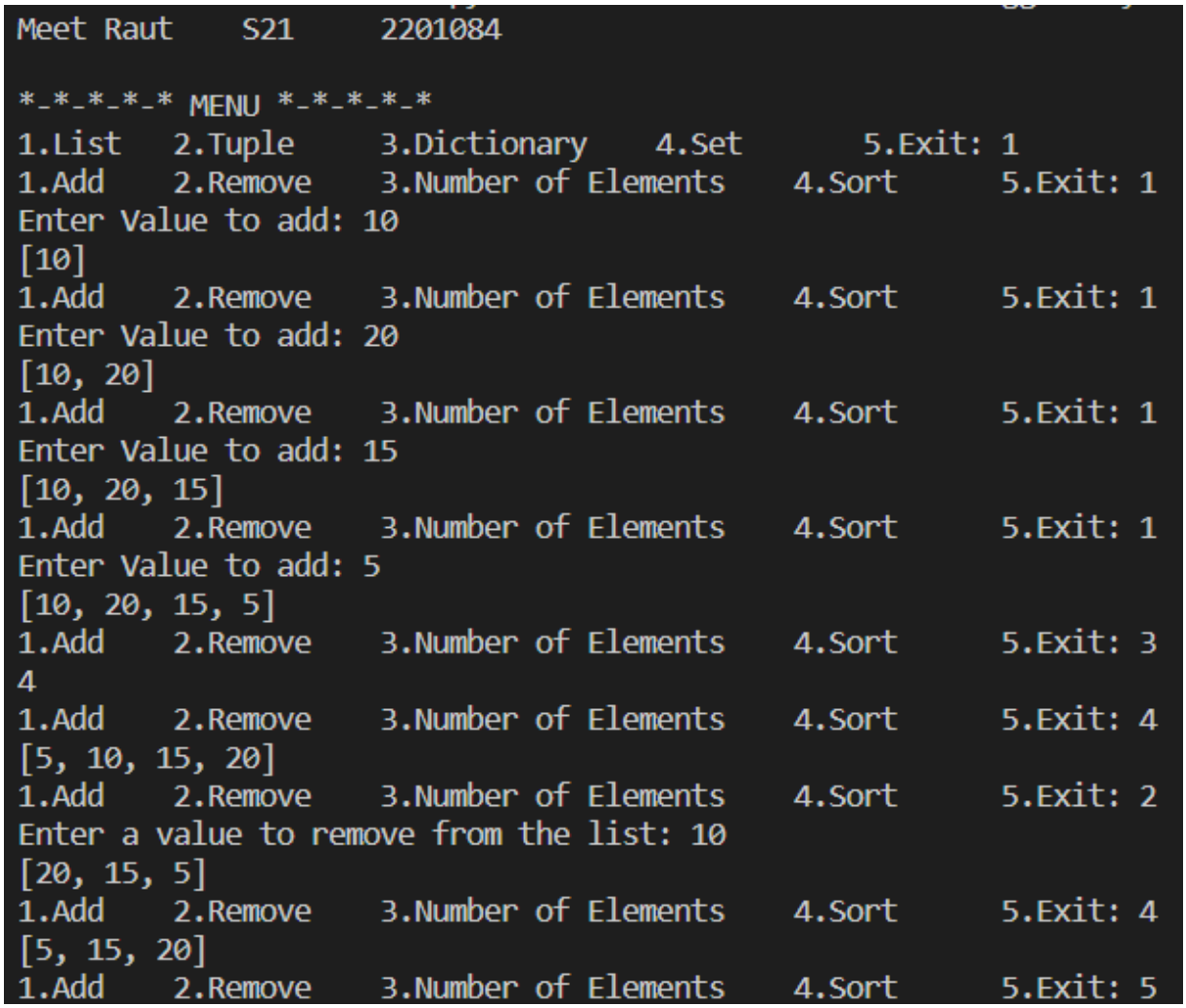
```
while True:
    print("Meet Raut    S21     2201084")
    print("\n*-*-*-*-* MENU *-*-*-*-*")
    n = int(input("1.List   2.Tuple     3.Dictionary    4.Set       5.Exit:
"))

    if n == 1:
        l = []
        while True:
            a = int(input("1.Add    2.Remove    3.Number of
Elements    4.Sort      5.Exit: "))
            if a == 1:
                l.append(int(input("Enter Value to add: ")))
                print(l)
            elif a == 2:
                try:
                    value = int(input("Enter a value to remove from the list:
"))
                    l.remove(value)
                    print(l)
                except ValueError:
                    print("Value not found in the list.")
            elif a == 3:
                print(len(l))
            elif a == 4:
                print(sorted(l))
            else:
                break

    elif n == 2:
        t = tuple(map(int, input("Enter values separated by space:
").split()))
        while True:
            a = int(input("1.Print      2.Number of
Elements    3.Sort      4.Exit: "))
            if a == 1:
                print(t)
            elif a == 2:
                print(len(t))
            elif a == 3:
                print(tuple(sorted(t)))
            else:
                break

    elif n == 3:
        d = {}
```

```
        while True:
            a = int(input("1.Add    2.Remove    3.Number of
Elements    4.Sort      5.Exit: "))
            if a == 1:
                key, value = input("Enter key value pair separated by space:
").split()
                d[key] = value
                print(d)
            elif a == 2:
                try:
                    key = input("Enter a key to remove from the dictionary: ")
                    d.pop(key)
                    print(d)
                except KeyError:
                    print("Key not found in the dictionary.")
            elif a == 3:
                print(len(d))
            elif a == 4:
                print(sorted(d.items()))
            else:
                break

    elif n == 4:
        s = set()
        while True:
            a = int(input("1.Add    2.Remove    3.Number of
Elements    4.Sort      5.Exit: "))
            if a == 1:
                s.add(int(input("Enter Value to add: ")))
                print(s)
            elif a == 2:
                try:
                    value = int(input("Enter a value to remove from the set:
"))
                    s.remove(value)
                    print(s)
                except KeyError:
                    print("Value not found in the set.")
            elif a == 3:
                print(len(s))
            elif a == 4:
                print(sorted(s))
            else:
                break

    elif n == 5:
        break
```

OUTPUT:

Below are the screenshots that show the output of each operation of the data types.

1. LISTS



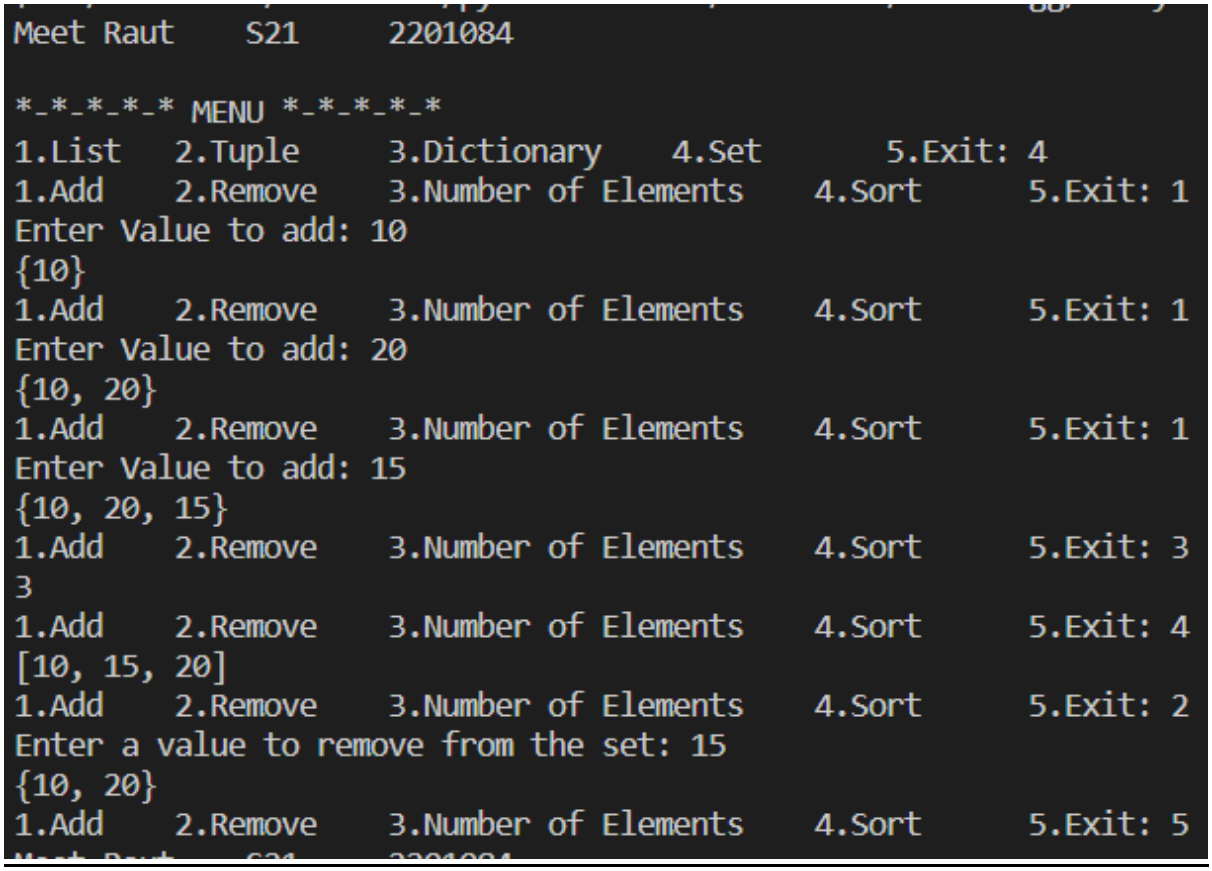
2. TUPLES

```
Meet Raut    S21     2201084

*_*_*_*_* MENU *_*_*_*_*
1.List   2.Tuple     3.Dictionary    4.Set       5.Exit: 2
Enter values separated by space: 10 20 15 5 35 12
1.Print      2.Number of Elements    3.Sort      4.Exit: 1
(10, 20, 15, 5, 35, 12)
1.Print      2.Number of Elements    3.Sort      4.Exit: 3
(5, 10, 12, 15, 20, 35)
1.Print      2.Number of Elements    3.Sort      4.Exit: 2
6
1.Print      2.Number of Elements    3.Sort      4.Exit: 4
```

3. DICTIONARY

```
Meet Raut    S21     2201084

*_*_*_*_* MENU *_*_*_*_*
1.List   2.Tuple     3.Dictionary    4.Set       5.Exit: 3
1.Add    2.Remove    3.Number of Elements    4.Sort      5.Exit: 1
Enter key value pair separated by space: 1 Apple
{'1': 'Apple'}
1.Add    2.Remove    3.Number of Elements    4.Sort      5.Exit: 1
Enter key value pair separated by space: 2 Banana
{'1': 'Apple', '2': 'Banana'}
1.Add    2.Remove    3.Number of Elements    4.Sort      5.Exit: 3
2
1.Add    2.Remove    3.Number of Elements    4.Sort      5.Exit: 4
[('1', 'Apple'), ('2', 'Banana')]
1.Add    2.Remove    3.Number of Elements    4.Sort      5.Exit: 2
Enter a key to remove from the dictionary: 1
{'2': 'Banana'}
```

4. SETS



CONCLUSION:

In conclusion, compound data types in Python are essential building blocks for organising and manipulating data effectively. Whether you're working with lists, dictionaries, tuples, or sets, understanding their characteristics, capabilities, and appropriate use cases empowers you to write cleaner, more efficient, and more expressive Python code.

Name: Meet Raut
Batch: S21
Roll Number: 2201084

Python Lab

Experiment 1-B

AIM:- To study basic looping and if statements and implement calculator, age sorter and implement formulae.

THEORY:-

If Statements:

In Python, an if statement is used for conditional execution. It allows you to execute a block of code only if a certain condition is true. The general syntax of an if statement in Python is:

```
if condition:
        # code block to execute if condition is True
```

The condition can be any expression that evaluates to either True or False. If the condition is True, the code block following the if statement is executed. If the condition is False, the code block is skipped.

Branching:

Branching occurs when the flow of execution in a program diverges based on certain conditions. In Python, branching is commonly achieved using if, elif (short for "else if"), and else statements. Multiple conditions can be evaluated using a series of if-elif-else statements. The syntax looks like:

```
 if condition1:
  # code block to execute if condition1 is True
elif condition2:
  # code block to execute if condition1 is False and condition2 is True
else:
  # code block to execute if all conditions are False
```

Python evaluates the conditions in order. If a condition evaluates to True, the corresponding code block is executed, and the rest of the conditions are skipped. If none of the conditions are True, the code block under the else statement (if present) is executed.

Looping:

Looping is the process of repeatedly executing a block of code until a certain condition is met. Python provides two main loop constructs: the for loop and the while loop.

- For Loop: The for loop iterates over a sequence (such as a list, tuple, string, or range) and executes the code block for each item in the sequence. Here's the syntax:

```
for item in sequence:
    # code block to execute for each item in the sequence
```

- While Loop: The while loop repeats a block of code as long as a specified condition is True. Here's the syntax:

```
while condition:
    # code block to execute while the condition is True
```

It's important to ensure that the condition within a while loop eventually becomes False; otherwise, the loop will continue indefinitely, resulting in an infinite loop.

For this part of the experiment, we are implementing these control statements with three programs- a calculator, age sorter and an equation solver.

CODE:

1. Calculator

```
#Calculator
print("Meet Raut    S21     2201084")
print("*-*-*-* MENU *-*-*-*")
print("*****MENU*****\n")
print("1. Addition 2. Subtraction 3. Multiplication 4. Division 5. Exit\n")
choice = 0

while(choice != 5):

    choice = int(input("Enter your choice: "))

    if(choice == 1):
        a = int(input("Enter first value: "))
        b = int(input("Enter second value: "))
        print("Answer: ", a + b)

    elif(choice == 2):
        a = int(input("Enter first value: "))
        b = int(input("Enter second value: "))
        print("Answer: ", a - b)

    elif(choice == 3):
        a = int(input("Enter first value: "))
        b = int(input("Enter second value: "))
        print("Answer: ", a * b)

    elif(choice == 4):
        a = int(input("Enter first value: "))
        b = int(input("Enter second value: "))
        while(b == 0):
            print("Are you dumb!")
            b = int(input("Enter second value: "))
        print("Answer: ", a / b)

    elif(choice == 4):
        a = int(input("Enter value: "))
        print("Answer: ", a ** 2 )

    elif(choice == 5):
        break

    else:
        print("Invalid choice")
```

```
print("*****Exited*****")
```

OUTPUT:-

```
Meet Raut    S21     2201084
*-*-*-* MENU *-*-*-*
*****MENU*****

1. Addition 2. Subtraction 3. Multiplication 4. Division 5. Exit

Enter your choice: 1
Enter first value: 10
Enter second value: 20
Answer:  30
Enter your choice: 2
Enter first value: 10
Enter second value: 20
Answer:  -10
Enter your choice: 3
Enter first value: 10
Enter second value: 20
Answer:  200
Enter your choice: 4
Enter first value: 10
Enter second value: 20
Answer:  0.5
Enter your choice: 5
*****Exited*****
```

2. Age Sorter

CODE:

```
#Age
print("Meet Raut    S21     2201084")
print("*-*-*-* MENU *-*-*-*")
age = int(input("Enter your age: "))
if(age <= 0):
    print("INVALID")
elif(age <= 12):
    print("You are a kid")
```

```
elif(age <= 20):
    print("You are a teenager")
elif(age <= 50):
    print("You are an adult")
elif(age <= 100):
    print("You are old")
else:
    print("Invalid")
```

OUTPUT:

```
meetr@HP MINGW64 /d/Documents/Meet Engg/2nd year/Sem 4/PYTHON LAB
$ D:/Downloads/Softwares/python.exe "d:/Documents/Meet Engg/2nd year/Sem 4/PYTHON LAB/age.py"
Meet Raut    S21     2201084
*-*-*-* MENU *-*-*-*
Enter your age: 10
You are a kid

meetr@HP MINGW64 /d/Documents/Meet Engg/2nd year/Sem 4/PYTHON LAB
$ D:/Downloads/Softwares/python.exe "d:/Documents/Meet Engg/2nd year/Sem 4/PYTHON LAB/age.py"
Meet Raut    S21     2201084
*-*-*-* MENU *-*-*-*
Enter your age: 15
You are a teenager

meetr@HP MINGW64 /d/Documents/Meet Engg/2nd year/Sem 4/PYTHON LAB
$ D:/Downloads/Softwares/python.exe "d:/Documents/Meet Engg/2nd year/Sem 4/PYTHON LAB/age.py"
Meet Raut    S21     2201084
*-*-*-* MENU *-*-*-*
Enter your age: 20
You are a teenager

meetr@HP MINGW64 /d/Documents/Meet Engg/2nd year/Sem 4/PYTHON LAB
$ D:/Downloads/Softwares/python.exe "d:/Documents/Meet Engg/2nd year/Sem 4/PYTHON LAB/age.py"
Meet Raut    S21     2201084
*-*-*-* MENU *-*-*-*
Enter your age: 45
You are an adult

meetr@HP MINGW64 /d/Documents/Meet Engg/2nd year/Sem 4/PYTHON LAB
$ D:/Downloads/Softwares/python.exe "d:/Documents/Meet Engg/2nd year/Sem 4/PYTHON LAB/age.py"
Meet Raut    S21     2201084
*-*-*-* MENU *-*-*-*
Enter your age: 65
You are old
```

3. Equation Solver

CODE:

```
#equation
print("Meet Raut    S21     2201084")
```

```
print("*-*-*-* MENU *-*-*-*")
print("1. x^2+2xy+y^2  2. x^3+xyz+z^2  3. xyz+xy^2+xz^2\n")
choice = int(input("Enter the equation which you want to solve: "))

if (choice == 1):
    x=int(input("Enter the value of x: "))
    y=int(input("Enter the value of y: "))
    print( x**2+2*x*y+y**2)

if (choice == 2):
    x=int(input("Enter the value of x: "))
    y=int(input("Enter the value of y: "))
    z=int(input("Enter the value of z: "))
    print( x**3+x*y*z+z**2)


if (choice == 3):
    x=int(input("Enter the value of x: "))
    y=int(input("Enter the value of y: "))
    z=int(input("Enter the value of z: "))
    print( x*y*z+x*(y**2)+x*(z**2))
```

OUTPUT:

```
meetr@HP MINGW64 /d/Documents/Meet Engg/2nd year/Sem 4/PYTHON LAB
$ D:/Downloads/Softwares/python.exe "d:/Documents/Meet Engg/2nd year/Sem 4/PYTHON LAB/equation.py"
Meet Raut    S21     2201084
*-*-*-* MENU *-*-*-*
1. x^2+2xy+y^2  2. x^3+xyz+z^2  3. xyz+xy^2+xz^2

Enter the equation which you want to solve: 1
Enter the value of x: 10
Enter the value of y: 20
900

meetr@HP MINGW64 /d/Documents/Meet Engg/2nd year/Sem 4/PYTHON LAB
$ D:/Downloads/Softwares/python.exe "d:/Documents/Meet Engg/2nd year/Sem 4/PYTHON LAB/equation.py"
Meet Raut    S21     2201084
*-*-*-* MENU *-*-*-*
1. x^2+2xy+y^2  2. x^3+xyz+z^2  3. xyz+xy^2+xz^2

Enter the equation which you want to solve: 2
Enter the value of x: 10
Enter the value of y: 20
Enter the value of z: 30
7900

meetr@HP MINGW64 /d/Documents/Meet Engg/2nd year/Sem 4/PYTHON LAB
$ D:/Downloads/Softwares/python.exe "d:/Documents/Meet Engg/2nd year/Sem 4/PYTHON LAB/equation.py"
Meet Raut    S21     2201084
*-*-*-* MENU *-*-*-*
1. x^2+2xy+y^2  2. x^3+xyz+z^2  3. xyz+xy^2+xz^2

Enter the equation which you want to solve: 3
Enter the value of x: 10
Enter the value of y: 20
Enter the value of z: 30
19000
```

CONCLUSION:

In conclusion, if statements, branching, and looping are fundamental control flow mechanisms in Python that allow you to make decisions and repeat actions based on conditions. Understanding these concepts is crucial for writing clear, efficient, and flexible code.

Name: Meet Raut
Batch: S21
Roll number: 2201084

PYTHON LAB

EXPERIMENT 1-C

AIM:

To study about sequences and apply it to writing three programs:

1. Find the sum of series : 1+½+⅓ +....+1/N
2. To determine all the Pythagorean triplets between 1 and 50.
3. Find the sum of series : $1+x^2/2 +x^3/3+....+ x^N/N$

THEORY:

Python's features like if-else, for-loops and nested loops help a lot for finding and simplifying problems on sequencing. These offer a robust framework for working with mathematical series. These tools empower you to explore, analyse, and manipulate series, gaining insights into diverse mathematical concepts and phenomena.

With Python's readability and simplicity, implementing mathematical algorithms becomes straightforward. Whether you're calculating Euler's series to understand exponential growth, generating Fibonacci numbers to model natural patterns, or finding Pythagorean triplets to explore geometric relationships, Python provides an intuitive environment for mathematical exploration and experimentation.

CODE:

```
# Write a program to find sum of series
print("Meet Raut    S21     2201084")
n = int(input("Enter number: "))
sum = 0
for i in range (1, n+1):
    sum = sum + (1/i)
    i = i + 1


print("Summation of series is: ", sum)
```

OUTPUT :

```
Meet Raut    S21     2201084
Enter number: 10
Summation of series is:  2.9289682539682538
```

CODE:

```
print("Meet Raut    S21     2201084")
triplets = []
count = 0
print("1. With duplicates     2. Without duplicates")
choice = int(input("Your choice: "))

if(choice == 1):
     for a in range (1, 51):
          for b in range (1, 51):
               for c in range (1, 51):
                    if(a**2 + b**2 == c**2):
                         count += 1
                         triplets.append((a,b,c))
elif(choice == 2):
     for a in range (1, 51):
          for b in range (a, 51):
               for c in range (1, 51):
                    if(a**2 + b**2 == c**2):
                         count += 1
                         triplets.append((a,b,c))
else:
     print("Invalid input")

if(choice == 1 or choice == 2):
     print("total number of triplets = ", count, "\nTriplets are: ")
     print(triplets)
```

OUTPUT :

```
meetr@HP MINGW64 /d/Documents/Meet Engg/2nd year/Sem 4/PYTHON LAB
$ D:/Downloads/Softwares/python.exe "d:/Documents/Meet Engg/2nd year/Sem 4/PYTHON LAB/pythagoreanTriplets.py"
Meet Raut    S21     2201084
1. With duplicates     2. Without duplicates
Your choice: 1
total number of triplets =  40
Triplets are:
[(3, 4, 5), (4, 3, 5), (5, 12, 13), (6, 8, 10), (7, 24, 25), (8, 6, 10), (8, 15, 17), (9, 12, 15), (9, 40, 41), (10, 24, 26), (12, 5, 13), (12, 9, 15), (12, 16, 20), (12, 35,
37), (14, 48, 50), (15, 8, 17), (15, 20, 25), (15, 36, 39), (16, 12, 20), (16, 30, 34), (18, 24, 30), (20, 15, 25), (20, 21, 29), (21, 20, 29), (21, 28, 35), (24, 7, 25), (24,
 10, 26), (24, 18, 30), (24, 32, 40), (27, 36, 45), (28, 21, 35), (30, 16, 34), (30, 40, 50), (32, 24, 40), (35, 12, 37), (36, 15, 39), (36, 27, 45), (40, 9, 41), (40, 30, 50)
, (48, 14, 50)]

meetr@HP MINGW64 /d/Documents/Meet Engg/2nd year/Sem 4/PYTHON LAB
$ D:/Downloads/Softwares/python.exe "d:/Documents/Meet Engg/2nd year/Sem 4/PYTHON LAB/pythagoreanTriplets.py"
Meet Raut    S21     2201084
1. With duplicates     2. Without duplicates
Your choice: 2
total number of triplets =  20
Triplets are:
[(3, 4, 5), (5, 12, 13), (6, 8, 10), (7, 24, 25), (8, 15, 17), (9, 12, 15), (9, 40, 41), (10, 24, 26), (12, 16, 20), (12, 35, 37), (14, 48, 50), (15, 20, 25), (15, 36, 39), (1
6, 30, 34), (18, 24, 30), (20, 21, 29), (21, 28, 35), (24, 32, 40), (27, 36, 45), (30, 40, 50)]
```

CODE:

```python
# Write a program to print the sine and cosine series
import math
print("Meet Raut    S21     2201084")
print("***MENU***")
print("1. Sine series   2. Cosine series")
choice = int(input("Enter your choice: "))
x = int(input("Enter x: "))


if(choice == 1):
    sum = x
    num = 1
    print("This is Sine serie")
    for i in range (0, 84):
        #print(sum, end = " ")

        #sum = sum + ((-1)**i) * ((x**num)/ math.factorial(num))
        print(((-1)**i) * ((x**num)/ math.factorial(num)), end = " ")
        num = num + 2
elif(choice == 2):
    print("This is Cosine series")
    sum = 1
    num = 0

    for i in range (0, 84):

        #print(sum, end = " ")

        #sum = sum + ((-1)**i) * ((x**num)/ math.factorial(num))
        print(((-1)**i) * ((x**num)/ math.factorial(num)), end = " ")
        num = num + 2

else:
```

```
    print("Invalid choice")
```

OUTPUT :

```
Meet Raut    S21     2201084
***MENU***
1. Sine series   2. Cosine series
Enter your choice: 1
Enter x: 12
This is Sine serie
12.0 -288.0 2073.6 -7109.4857142857145 14218.971428571429 -18613.926233766233 17182.085754245752 -11782.001660054231 6237.530290616946 -2626.32854341766l4 900.4555006003411 -2
56.25611084278484 61.50146660226836 -12.615685456875S6 2.2372644159976365 -0.3464151353802792 0.04723842755185625 -0.005716246695350672 0.000617972615713586Z -6.004592217459946
65e-05 5.272324873867270Se-06 -4.203847075508787e-07 3.057343327642754e-08 -2.036343381963721Ge-09 1.2467408461002378e-10 -7.040418895624871Se-12 3.6785933271770Ole-13 -1.7835
604010555157e-14 8.0461371476188686-16 -3.3858671807630535e-17 1.3321444645625l3e-18 -4.9111316665899096e-20 1.7000071153580454e-21 -5.5359797515051696-23 1.6990219186205123e-
24 -4.9227194422807595e-26 1.3486902581591122e-27 -3.4993044536020206e-29 8.610728662315294e-31 -2.0122442833063978e-32 4.4716539629031O6e-34 -9.461036888892849e-36 1.90810828
01128435e-37 -3.6723816136895143e-39 6.752080597181947e-41 -1.1871790060979247e-42 1.9980572332643893e-44 -3.2219511936178284e-46 4.9823987530172604e-48 -7.395025978504282e-50
 1.0543403375293234e-51 -1.4451266762252292e-53 1.9056615510662362e-55 -2.4194609712003e-57 2.9595852858719264e-59 -3.4904199931659O8e-61 3.971400750757671e-63 -4.362179314333
368e-65 4.628306964809Ge-67 -4.7463059390490182e-69 4.7070797261059654e-71 -4.5169897411652606e-73 4.1964291788890167e-75 -3.7763142217224e-77 3.2932972863858135e-79 -2.784702
3443309287e-81 2.284103084892081e-83 -1.8181915103618555e-85 1.4052145636115Ge-87 -1.0548998913568223e-89 7.6953183564023Sle-92 -5.457135050339499e-94 3.763541414027241e-96 -
2.525160579721939Ge-98 1.6489348969706116e-100 -1.0483294709217134e-102 6.491204154314015e-105 -3.9159338006754006e-107 2.3023618622295347e-109 -1.3197201980775934e-111 7.3773
17877452385e-114 -4.0230772337845Z2e-116 2.1408836720804605e-118 -1.1120671263963145e-120
meetr@HP MINGW64 /d/Documents/Meet Engg/2nd year/Sem 4/PYTHON LAB
$ D:/Downloads/Softwares/python.exe "d:/Documents/Meet Engg/2nd year/Sem 4/PYTHON LAB/sine and cosine series.py"
Meet Raut    S21     2201084
***MENU***
1. Sine series   2. Cosine series
Enter your choice: 2
Enter x: 12
This is Cosine series
1.0 -72.0 864.0 -4147.2 10664.228571428572 -17062.765714285713 18613.926233766233 -14727.502075067789 8836.501245040674 -4158.353527077964 1575.797126050597 -491.1575457820042
 128.12805542139242 -28.38529227797001 5.406722338660955 -0.8949057663990545 0.1299056757676047 -0.016672386194772792 0.001905415565116890Z -0.000195149247067448Z7 1.801377665
237984e-05 -1.5063785353906487e-06 1.1465037478660328e-07 -7.975678246024576e-09 5.090858454909304e-10 -2.992178030640S7e-11 1.624712052836508Be-12 -8.174651838171114e-14 3.82
191514511896Ze-15 -1.664718030541834Be-16 6.771734361526108e-18 -2.578344124959702Se-19 9.208371874856O8e-21 -3.090922027923719Ze-22 9.769376032067945e-24 -2.912609003349449Se
-25 8.204532403801266e-27 -2.187065283501263e-28 5.525217558318980e-30 -1.324727486510045Ie-31 3.018366424959596Ze-33 -6.543883848150888e-35 1.351517669841326Ale-36 -2.662476669
924898e-38 5.007793109576Ole-40 -9.002774129575929e-42 1.548494355779901Ze-43 -2.550711361614114e-45 4.027438992022285Se-47 -6.100896432266033e-49 8.874031174205139e-51 -1.240
4003970933218e-52 1.667453857182956Ze-54 -2.157352699320267Ge-56 2.688289968000333Se-58 -3.228638493678465Ae-60 3.739735706963473Ge-62 -4.180421842902811e-64 4.512599290689691
5e-66 -4.706752845569431e-68 4.746305390490182e-70 -4.629914484694393e-72 4.371280394676059e-74 -3.996599217989539Ae-76 3.540294582864749Ae-78 -3.039966725894597Ze-80 2.531547
585755389Ge-82 -2.045465449157087Ae-84 1.604286626789872Ae-86 -1.221925707488319Ze-88 9.041999068772763e-91 -6.503085934987903e-93 4.547612541949583e-95 -3.093321710159376e-97
 2.047427497071842Ze-99 -1.319147917576489Ze-101 8.276285296750369e-104 -5.058081159205726e-106 3.012256769750308e-108 -1.748629262452811Ae-110 9.897901485581951e-113 -5.46467
9909223989e-115 2.943715049110633Ze-117 -1.547626750901537Ge-119
```

CONCLUSION :

In conclusion, Python's versatility as a programming language, coupled with its rich set of features for handling loops, conditional statements, and functions, provides a powerful toolkit for working with mathematical series. From Euler's series to Fibonacci sequences and Pythagorean triplets, Python enables users to explore these mathematical phenomena with ease and clarity. By leveraging Python's simplicity and readability, mathematicians, scientists, and programmers can bridge the gap between abstract mathematical concepts and practical computational solutions. Whether for research, education, or creative exploration, Python serves as an invaluable ally in the journey to unlock the secrets hidden within mathematical series.

NAME: Meet Raut
DIV: S21
ROLL NO: 2201084

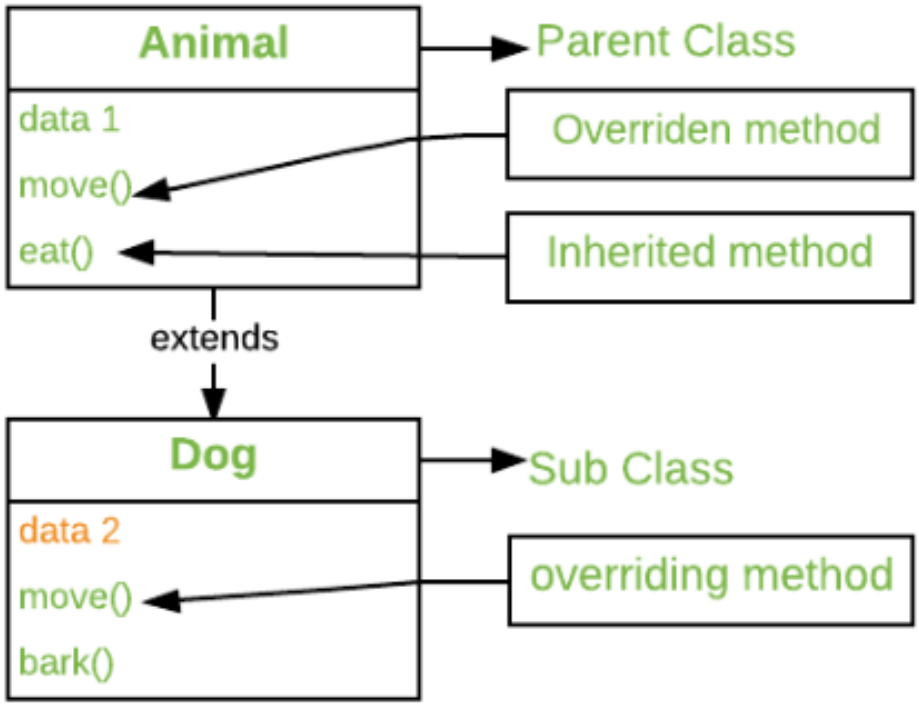
EXPERIMENT – 2:

- ***AIM:*** **To study and implement program on the concept of method overriding.**

- ***THEORY:***

Method Overriding in Python:

Any object-oriented programming language can allow a subclass or child class to offer a customized implementation of a method already supplied by one of its superclasses or parent classes. This capability is known as method overriding. The term "override" refers to a method in a subclass that replaces a method in a superclass when both methods share the same name, parameters, signature, and return type (or sub-type).



The object that calls a method will determine which version of the method is executed. When a method is called from an object of a parent class, the method's parent class version is executed; however, when a method is called from an object of a subclass, the child class version is executed. In other words, the version of an overridden method depends on the object being referenced, not the type of the reference variable.

- **Benefits of Method Overriding in Python:**

Both **overriding and overloading in Python** provide unique benefits. Let’s go through the benefits of method overriding in python.

- Overriding methods permit a user to change the existing methods' behaviour. For its implementation, a minimum of two classes are needed.
- The key benefit is that it allows the main class to declare those methods shared by all. Also, it allows subclasses to define their implementations of any or all such methods.
- When overriding a method, you need to use inheritance.
- The parent class function and child class function must have the same signature. They must have the same number of parameters.
- It allows a child class to adapt the execution of any method provided by any one of its parent classes.

➢ **PROGRAM 1:**

```
print("Meet Raut-S21-2201084")

class A:
   def sum(self, A=None, B=None, C=None):
      if (A is not None and B is not None and C is not None):
         print("Sum of A, B & C is:", A+B+C)
      elif (A is not None and B is not None and C is None):
         print("Sum of A & B is:",A+B)
      elif (A is not None and B is None and C is None):
         print("A=",A)
      else:
         print("Sum not possible!!")

ob=A()
ob.sum(12,10,20)
ob.sum(12,10)
ob.sum(12)
ob.sum()
```

- **OUTPUT:**

```
meetr@HP MINGW64 /d/Documents/Meet Engg/2nd year/Sem 4/PYTHON LAB
$ D:/Downloads/Softwares/python.exe "d:/Documents/Meet Engg/2nd year/Sem 4/PYTHON LAB/2.py"
Meet Raut-S21-2201084
Sum of A, B & C is: 42
Sum of A & B is: 22
A= 12
Sum not possible!!
```

➢ **PROGRAM 2:**

```
class Parent():

   def __init__(self):
      self.value = "Inside Parent"
```

```
    def show(self):
        print(self.value)

class Child(Parent):

    def __init__(self):
        self.value = "Inside Child"

    def show(self):
        print(self.value)

obj1 = Parent()
obj2 = Child()
obj1.show()
obj2.show()
```

- **OUTPUT:**

```
meetr@HP MINGW64 /d/Documents/Meet Engg/2nd year/Sem 4/PYTHON LAB
$ D:/Downloads/Softwares/python.exe "d:/Documents/Meet Engg/2nd year/Sem 4/PYTHON LAB/2_2.py"
Inside Parent
Inside Child
```

- **PROGRAM 3:**

```
class Parent1():

    def show(self):
        print ("Inside Parent1")

class Parent2():

    def display(self):
        print ("Inside Parent2")

class Child (Parent1, Parent2):

    def show(self):
        print ("Inside Child")
obj = Child ()
obj. show ()
obj. display ()
```

- **OUTPUT:**

```
meetr@HP MINGW64 /d/Documents/Meet Engg/2nd year/Sem 4/PYTHON LAB
$ D:/Downloads/Softwares/python.exe "d:/Documents/Meet Engg/2nd year/Sem 4/PYTHON LAB/2_3.py"
Inside Child
Inside Parent2
```

➢ **PROGRAM 4:**

```
class Parent ():

    def display(self):
        print ("Inside Parent")

class Child (Parent):

    def show(self):
        print ("Inside Child")

class GrandChild (Child):

    def show(self):
        print ("Inside GrandChild")


g = GrandChild ()
g.show()
g.display()
```

- **OUTPUT:**

```
meetr@HP MINGW64 /d/Documents/Meet Engg/2nd year/Sem 4/PYTHON LAB
$ D:/Downloads/Softwares/python.exe "d:/Documents/Meet Engg/2nd year/Sem 4/PYTHON LAB/2_4.py"
Inside GrandChild
Inside Parent
```

➢ **PROGRAM 5:**

```
class student:
    def __init__ (self, name, mmarks=0, dmarks=0, cmarks=0, dlmarks=0,
dsmarks=0, amarks=0):
        self.name=input("Enter Your Name: ")
        self.mmarks=int(input("Enter MATHS Marks: "))
        self.dmarks=int(input("Enter DSA Marks: "))
```

```
        self.cmarks=int(input("Enter CG Marks: "))
        self.dlmarks=int(input("Enter DLCA Marks: "))
        self.dsmarks=int(input("Enter DSGT Marks: "))

self.amarks=float(self.mmarks+self.dmarks+self.cmarks+self.dlmarks+self.dsm
arks)/5

    def result_display (self):
        print("STUDENT NAME: ",self.name)
        print("Average Marks: ",self.amarks)

        if (self.amarks >= 80):
             print("You got Distinction Class")
        elif (self.amarks >= 60):
            print("You got First Class")
        elif (self.amarks >=40):
            print("You have Passes")
        else:
            print("You are Failed")
name=""
mmarks=0
dmarks=0
cmarks=0
dlmarks=0
dsmarks=0
amarks=0
ob=student(name, mmarks, dmarks, cmarks, dlmarks, dsmarks, amarks)
print("------------YOUR RESULT ------------")
ob.result_display()
```

- **OUTPUT:**

```
meetr@HP MINGW64 /d/Documents/Meet Engg/2nd year/Sem 4/PYTHON LAB
$ D:/Downloads/Softwares/python.exe "d:/Documents/Meet Engg/2nd year/Sem 4/PYTHON LAB/2_5.py"
Enter Your Name: Meet
Enter MATHS Marks: 89
Enter DSA Marks: 97
Enter CG Marks: 88
Enter DLCA Marks: 84
Enter DSGT Marks: 95
------------YOUR RESULT -------------
STUDENT NAME:  Meet
Average Marks:  90.6
You got Distinction Class
```

- ***CONCLUSION:*** Hence, we have successfully implemented program on the concept of method overriding.

NAME: Meet Raut
DIV: S21
ROLL NO: 2201084

EXPERIMENT – 3:

- ***AIM:*** **To study and implement program on the concept of classes, functions and inheritance.**

- ***THEORY:***

➢ Classes in Python:

In Python, a class is a user-defined data type that contains both the data itself and the methods that may be used to manipulate it. In a sense, classes serve as a template to create objects. They provide the characteristics and operations that the objects will employ.
Suppose a class is a prototype of a building. A building contains all the details about the floor, rooms, doors, windows, etc. we can make as many buildings as we want, based on these details. Hence, the building can be seen as a class, and we can create as many objects of this class.

Creating Classes in Python:

In Python, a class can be created by using the keyword class, followed by the class name. The syntax to create a class is given below.

Syntax

```
class ClassName:
    #statement_suite
```

In Python, we must notice that each class is associated with a documentation string which can be accessed by using **<class-name>.__doc__.** A class contains a statement suite including fields, constructor, function, etc. definition.

➢ Objects in Python:

An object is a particular instance of a class with unique characteristics and functions. After a class has been established, you may make objects based on it. By using the class constructor, you may create an object of a class in Python. The object's attributes are initialised in the constructor, which is a special procedure with the name __init__.

Syntax:

```
# Declare an object of a class
object_name = Class_Name(arguments)
```

The self-parameter:

The self-parameter refers to the current instance of the class and accesses the class variables. We can use anything instead of self, but it must be the first parameter of any function which belongs to the class.

_ _init_ _ method:

In order to make an instance of a class in Python, a specific function called __init__ is called. Although it is used to set the object's attributes, it is often referred to as a constructor.
The self-argument is the only one required by the __init__ method. This argument refers to the newly generated instance of the class. To initialise the values of each attribute associated with the objects, you can declare extra arguments in the __init__ method.

- **PROGRAM 1:**

```
class Person:
    count = 0                    # This is a class variable

    def __init__(self, name, age):
        self.name = name        # This is an instance variable
        self.age = age
        Person.count += 1
person1 = Person("Aditya", 25)
person2 = Person("Prasad", 30)
print(Person.count)
```

- **OUTPUT:**

```
meetr@HP MINGW64 /d/Documents/Meet Engg/2nd year/Sem 4/PYTHON LAB
$ D:/Downloads/Softwares/python.exe "d:/Documents/Meet Engg/2nd year/Sem 4/PYTHON LAB/3_1.py"
Number of person:  2
```

- **PROGRAM 2:**

```
class Person:
    def __init__(self, name, age):
        self.name = name    # This is an instance variable
        self.age = age
person1 = Person("Aditya", 25)
person2 = Person("Bunty", 30)
print(person1.name)
print(person2.age)
```

- **OUTPUT:**

```
meetr@HP MINGW64 /d/Documents/Meet Engg/2nd year/Sem 4/PYTHON LAB
$ D:/Downloads/Softwares/python.exe "d:/Documents/Meet Engg/2nd year/Sem 4/PYTHON LAB/3_2.py"
Aditya
30
```

- **Python Functions:**

A collection of related assertions that carry out a mathematical, analytical, or evaluative operation is known as a function. An assortment of proclamations called Python Capabilities returns the specific errand. Python functions are necessary for intermediate-level programming and are easy to define. Function names meet the same standards as variable names do. The objective is to define a function and group-specific frequently performed actions. Instead of repeatedly creating the same code block for various input variables, we can call the function and reuse the code it contains with different variables.
Client-characterized and worked-in capabilities are the two primary classes of capabilities in Python. It aids in maintaining the program's uniqueness, conciseness, and structure.

Advantages of Python Functions:

Pause We can stop a program from repeatedly using the same code block by including functions.

- Once defined, Python functions can be called multiple times and from any location in a program.
- Our Python program can be broken up into numerous, easy-to-follow functions if it is significant.
- The ability to return as many outputs as we want using a variety of arguments is one of Python's most significant achievements.
- However, Python programs have always incurred overhead when calling functions.

However, calling functions has always been overhead in a Python program.

Syntax

```
#  An example Python Function
def function_name( parameters ):
   # code block
```

The accompanying components make up to characterize a capability, as seen previously.

- The start of a capability header is shown by a catchphrase called def.
- function_name is the function's name, which we can use to distinguish it from other functions. We will utilize this name to call the capability later in the program. Name functions in Python must adhere to the same guidelines as naming variables.
- Using parameters, we provide the defined function with arguments. Notwithstanding, they are discretionary.
- A colon (:) marks the function header's end.
- We can utilize a documentation string called docstring in the short structure to make sense of the reason for the capability.
- Several valid Python statements make up the function's body. The entire code block's indentation depth-typically four spaces-must be the same.
- A return expression can get a value from a defined function.

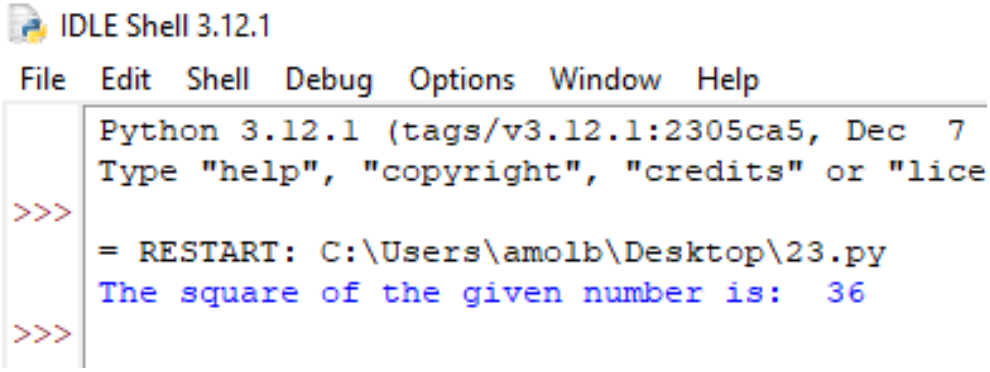
Calling a Function:

Calling a Function To define a function, use the def keyword to give it a name, specify the arguments it must receive, and organize the code block.
When the fundamental framework for a function is finished, we can call it from anywhere in the program.

- **PROGRAM 3:**

```
def square( num ):

    return num**2
object_ = square(6)
print( "The square of the given number is: ", object_ )
```

- **OUTPUT:**



- **PROGRAM 4:**

```
def a_function( string ):
    "This prints the value of length of string"
    return len(string)

# Calling the function we defined
print( "Length of the string Functions is: ", a_function( "Functions" ) )
print( "Length of the string Python is: ", a_function( "Python" ) )
```

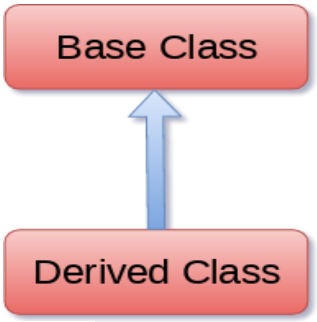
- **OUTPUT:**

```
IDLE Shell 3.12.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.12.1 (tags/v3.12.1:2305ca5, Dec  7 2023,
Type "help", "copyright", "credits" or "license()
>>>
= RESTART: C:\Users\amolb\Desktop\23.py
Length of the string Functions is:  9
Length of the string Python is:  6
>>>
```

➢ Python Inheritance:

Inheritance is an important aspect of the object-oriented paradigm. Inheritance provides code reusability to the program because we can use an existing class to create a new class instead of creating it from scratch.

In inheritance, the child class acquires the properties and can access all the data members and functions defined in the parent class. A child class can also provide its specific implementation to the functions of the parent class. In this section of the tutorial, we will discuss inheritance in detail. In python, a derived class can inherit base class by just mentioning the base in the bracket after the derived class name. Consider the following syntax to inherit a base class into the derived class.



Syntax

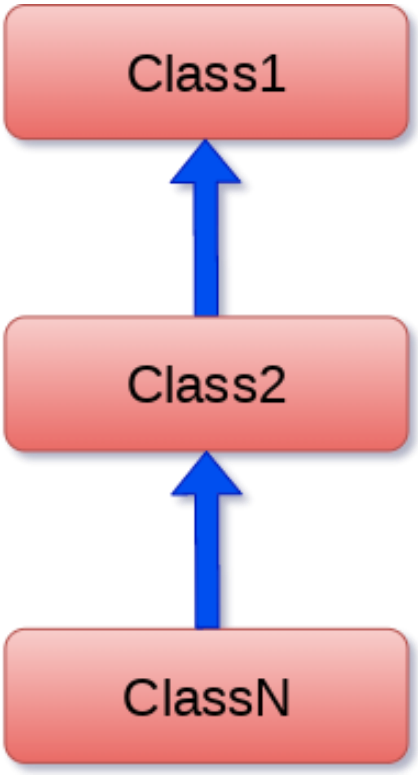
```
class derived-class(base class):
    <class-suite>
```

A class can inherit multiple classes by mentioning all of them inside the bracket. Consider the following syntax.

```
class derive-class(<base class 1>, <base class 2>, ..... <base class n>):
    <class - suite>
```

Python Multi-Level inheritance:

Multi-Level inheritance is possible in python like other object-oriented languages. Multi-level inheritance is archived when a derived class inherits another derived class. There is no limit on the number of levels up to which, the multi-level inheritance is archived in python.

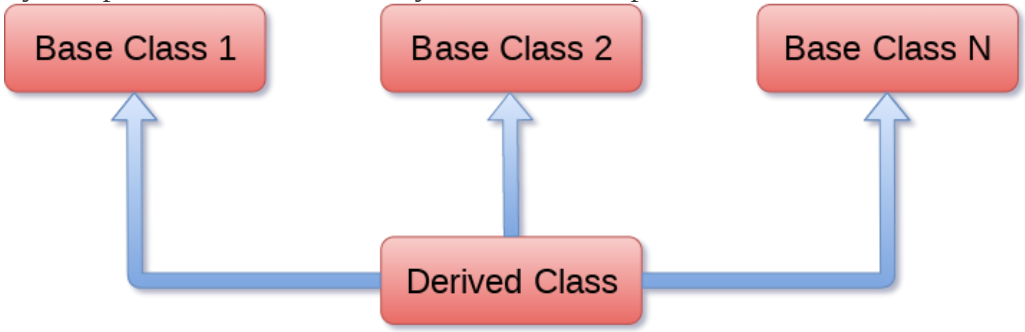


Syntax

```
class class1:
    <class-suite>
class class2(class1):
    <class suite>
class class3(class2):
    <class suite>
.
```

Python Multiple inheritance:

Python provides us the flexibility to inherit multiple base classes in the child class.



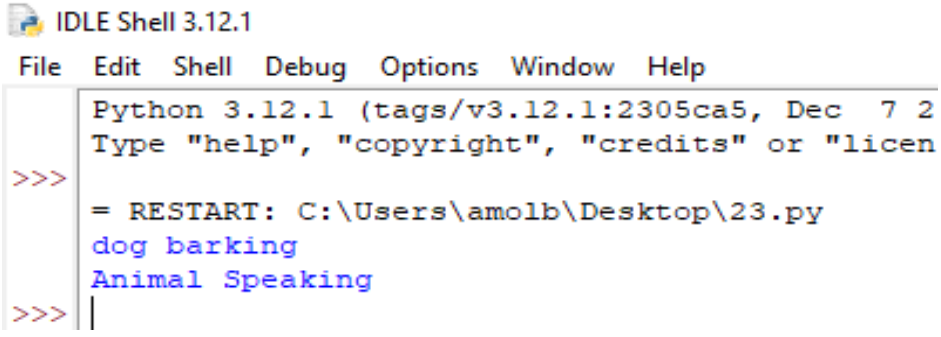
Syntax

```
class Base1:
   <class-suite>

class Base2:
   <class-suite>
.
.
.
class BaseN:
   <class-suite>

class Derived(Base1, Base2, ..... BaseN):
   <class-suite>
```

- **PROGRAM 5:**

```
class Animal:
   def speak(self):
      print("Animal Speaking")
#child class Dog inherits the base class Animal
class Dog(Animal):
   def bark(self):
      print("dog barking")
d = Dog()
d.bark()
d.speak()
```

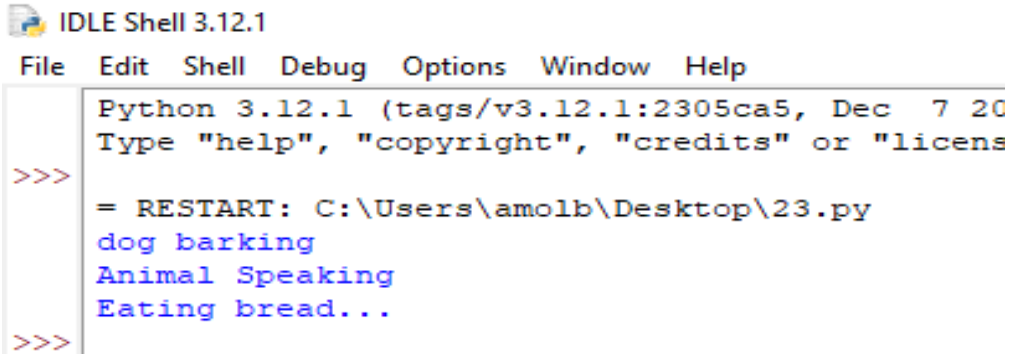
- **OUTPUT:**



- **PROGRAM 6:**

```
class Animal:
    def speak(self):
        print("Animal Speaking")
#The child class Dog inherits the base class Animal
class Dog(Animal):
    def bark(self):
        print("dog barking")
#The child class Dogchild inherits another child class Dog
class DogChild(Dog):
    def eat(self):
        print("Eating bread...")
d = DogChild()
d.bark()
d.speak()
d.eat()
```

- **OUTPUT:**

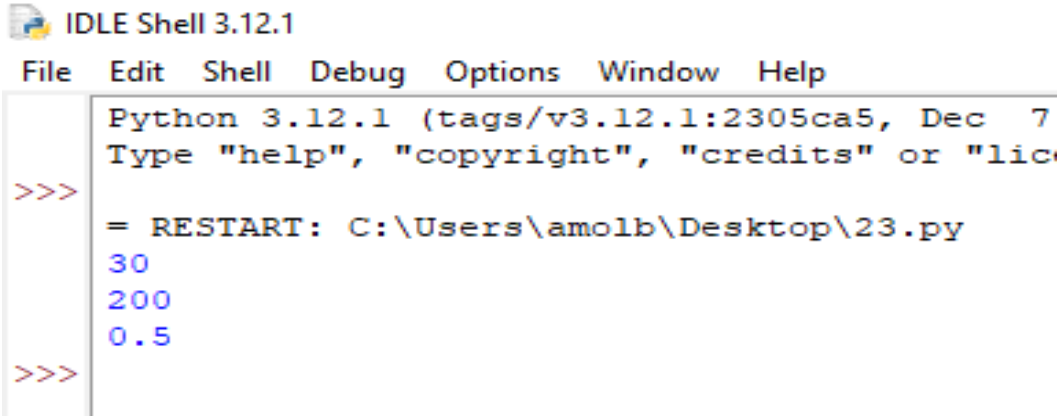


- **PROGRAM 7:**

```
class Calculation1:
    def Summation(self,a,b):
        return a+b;

class Calculation2:
    def Multiplication(self,a,b):
        return a*b;

class Derived(Calculation1,Calculation2):
    def Divide(self,a,b):
```

```
        return a/b;

d = Derived()
print(d.Summation(10,20))
print(d.Multiplication(10,20))
print(d.Divide(10,20))
```

- **OUTPUT:**



- **PROGRAM 8:**

```
#MULTILEVEL INHERITANCE

class A:
    def _init_ (self, Name, Age):
        self.Name=Name
        self.Age=Age
    def displayA (self):
        print("Parent: Name={0} || Age={1}".format(self.Name,self.Age))

class B(A):
    def _init_ (self, Name, Age, Roll):
        A._init_(self,Name,Age)
        self.Roll=Roll
    def displayB (self):
```

```
        print("Child: Name={0} || Age={1} || Roll
No={2}".format(self.Name,self.Age,self.Roll))

class C(B):
    def _init_ (self, Name, Age, Roll, Gender):
        A._init_(self,Name,Age)
        B._init_(self,Name,Age,Roll)
        self.Gender=Gender
    def displayC (self):
        print("GrandChild: Name={0} || Age={1} || Roll No={2} ||
Gender={3}".format(self.Name,self.Age,self.Roll,self.Gender))

ob=C("Meet", 20, 72, "Male")
ob.displayA()
ob.displayB()
ob.displayC()
```

- **OUTPUT:**

```
meetr@HP MINGW64 /d/Documents/Meet Engg/2nd year/Sem 4/PYTHON LAB
$ D:/Downloads/Softwares/python.exe "d:/Documents/Meet Engg/2nd year/Sem 4/PYTHON LAB/3_8.py"
Parent: Name=Meet || Age=20
Child: Name=Meet || Age=20 || Roll No=72
GrandChild: Name=Meet || Age=20 || Roll No=72 || Gender=Male
```

- ***CONCLUSION:*** Hence, we have successfully implemented program on the concept of classes, functions and inheritance.

NAME: Meet Raut

DIV: S2-1

ROLL NO: 2201084

EXPERIMENT – 4:

- ***AIM:*** **To study and implement programs on exploring Files and directories:**
  - **a) To append data to existing file and then display the entire file.**
  - **b) To count number of lines, words and characters in a file. .**
  - **c) Python program To display file available in current directory.**

- ***THEORY:***

a) Append Data to Existing File and Display the Entire File:

In Python, you can use the built-in open() function to open a file in different modes, such as 'r' for reading, 'w' for writing, and 'a' for appending. To append data to an existing file, you can open the file in append mode and then use the write() method to add new content. After appending, you can read and display the entire file.

- **PROGRAM 1:**

```
file=open('hello.txt','r')
print(file.read())

file=open('hello.txt','w')
file.write("Writing PYTHON LAB\n")

file=open('hello.txt','a')
file.write("Appending an text TSEC\n")
file.close()
```

- **OUTPUT:**

```
Writing PYTHON LAB
Appending an text TSEC
This is extra test IndiaThis is extra test India
>>>
```

- **PROGRAM 2:**

```
file1 = open("myfile.txt", "w")
L = ["This is Delhi \n", "This is Paris \n", "This is London"]
file1.writelines(L)
file1.close()

# Append-adds at last
# append mode
file1 = open("myfile.txt", "a")

# writing newline character
file1.write("\n")
file1.write("Today")

# without newline character
file1.write("Tomorrow")


file1 = open("myfile.txt", "r")
print("Output of Readlines after appending")
print(file1.read())
print()
file1.close()
```

- **OUTPUT:**

```
Output of Readlines after appending
This is Delhi
This is Paris
This is London
TodayTomorrow
```

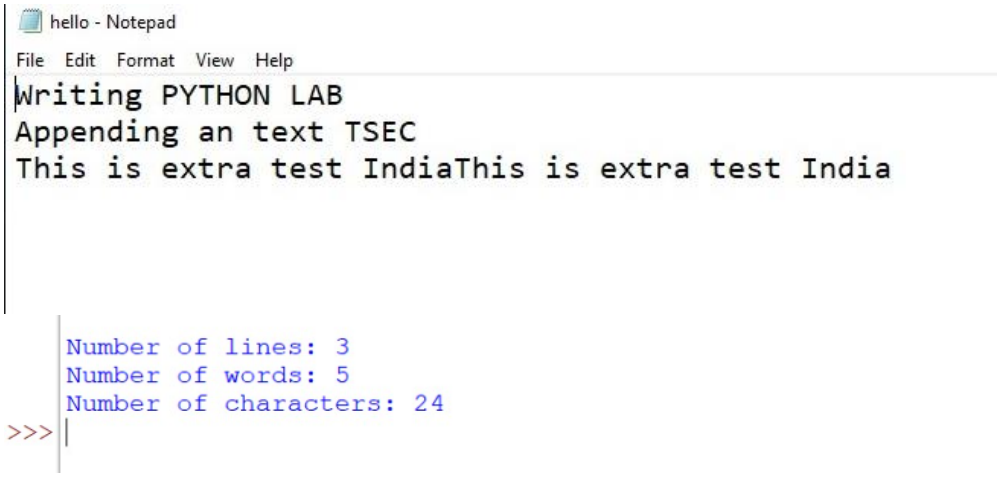
b) Count Number of Lines, Words, and Characters in a File:

To count the number of lines, words, and characters in a file, you can read the file and then use Python string and list methods to perform the required counts. For example, to count lines, you can split the content based on newline characters (\n). To count words, you can use the split() method, and to count characters, you can use the len() function.

- **PROGRAM 1:**

```
filename= r"C:\Users\Lenovo\Desktop\S21_2201072\hello.txt"
num_lines = 0
num_words = 0
num_chars = 0

with open(filename, 'r') as file:
    for line in file:
        num_lines += 1

words = line.split()
num_words += len(words)
num_chars += len(line)

file =  open(filename, 'a')
extra_text = "This is extra test India"

file.write(extra_text)
file.close()

print(f"Number of lines: {num_lines}")
print(f"Number of words: {num_words}")
print(f"Number of characters: {num_chars}")
```

- **OUTPUT:**



c) Display Files Available in the Current Directory

To display the files available in the current directory, you can use the os module, which provides a function called listdir(). This function returns a list containing the names of the entries in the directory given by the path. You can iterate over the list and print the names of the files.

What is a Directory in Python?

A Directory, sometimes known as a folder, is a unit organizational structure in a computer's file system for storing and locating files or more folders. Python now supports several APIs to list the directory contents. For instance, we can use the Path.iterdir, os.scandir, os.walk, Path.rglob, or os.listdir functions.

How to List Files in a Directory in Python

There are multiple ways of listing all the files in a directory. In this article, we will discuss the below modules and their functions to fetch the list of files in a directory. We will cover a total of 5 ways with examples to check the list of files in a directory.

1. Using **OS Module**
2. Using **glob Module**

Using os.listdir() method to get the list of files

os.listdir() method gets the list of all files and directories in a specified directory. By default, it is the current directory. Beyond the first level of folders, os.listdir() does not return any files or folders.

Syntax: *os.listdir(path)*

Parameters:

- ***Path***: *Path of the directory*

Return Type: *returns a list of all files and directories in the specified path*

Using os.walk() method to access files in a Directory tree

OS.walk() generates file names in a directory tree. This function returns a list of files in a tree structure. The method loops through all of the directories in a tree.

Syntax: *os.walk(top, topdown, onerror, followlinks)*

Parameters:

- ***top:*** *It is the top directory from which you want to retrieve the names of the component files and folders.*
- ***topdown:*** *Specifies that directories should be scanned from the top down when set to True. If this parameter is False, directories will be examined from the top down.*
- ***onerror:*** *It provides an error handler if an error is encountered*
- ***followlinks***: *if set to True, visits folders referenced by system links*

Return: *returns the name of every file and folder within a directory and any of its subdirectories.*

Using os.scandir() method to list files in a Directory

os.scandir() is an efficient version of os.listdir() function. It was later released by Python and is supported for Python 3.5 and greater.

Syntax: *os.scandir(path)*

Parameter:

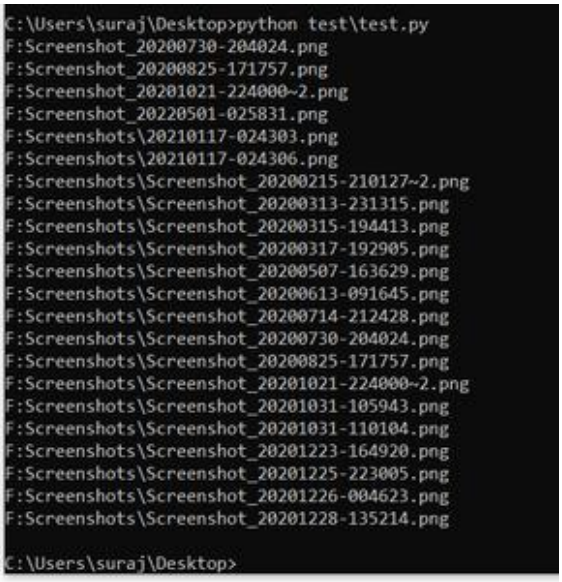
- ***Path-*** *Path of the directory.*

Return Type: *returns an iterator of os.DirEntry object.*

- **PROGRAM 1:**

```
import os
# traverse whole directory
for root, dirs, files in os.walk(r'F:'):
        # select file name
        for file in files:
                # check the extension of files
               if file.endswith('.png'):
                        # print whole path of files
                          print(os.path.join(root, file))
```

- **OUTPUT:**

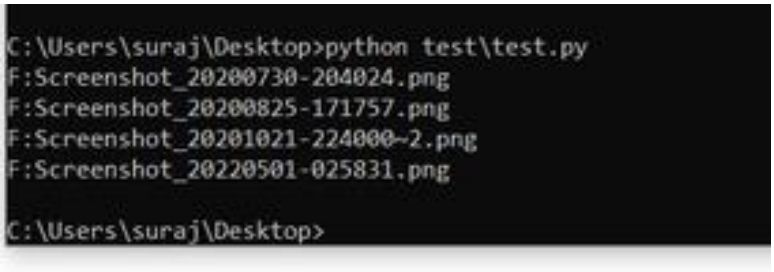


- **PROGRAM 2:**

```
import os
# return all files as a list
 for file in os.listdir(r'F:'):
            # check the files which are end with specific extension
              if file.endswith(".png"):
                        # print path name of selected files
```

```
print(os.path.join(r'F:', file))
```

- **<u>OUTPUT:</u>**



- ***<u>CONCLUSION:</u>*** **Hence, we have successfully implemented program on exploring files and directories; LO 2.**

EXPERIMENT – 5:

AIM: To study and implement program on creating GUI with python containing widgets such as labels, textbox, radio, checkboxes and custom dialog boxes.

THEORY:

Tkinter is a Python Package for creating GUI applications. Python has a lot of GUI frameworks, but Tkinter is the only framework that's built into the Python standard library.

Tkinter has several strengths; it's cross-platform, so the same code works on Windows, macOS, and Linux.

Tkinter is lightweight and relatively painless to use compared to other frameworks. This makes it a compelling choice for building GUI applications in Python, especially for applications where a modern shine is unnecessary, and the top priority is to build something functional and cross-platform quickly.

- **Use Cases of Tkinter:**

1. Creating windows and dialog boxes: Tkinter can be used to create windows and dialog boxes that allow users to interact with your program. These can be used to display information, gather input, or present options to the user.

To create a window or dialog box, you can use the *Tk()* function to create a root window, and then use functions like *Label*, *Button*, and *Entry* to add widgets to the window.

2. Building a GUI for a desktop application: Tkinter can be used to create the interface for a desktop application, including buttons, menus, and other interactive elements.

To build a GUI for a desktop application, you can use functions like *Menu*, *Checkbutton*, and *RadioButton* to create menus and interactive elements and use layout managers like *pack* and *grid* to arrange the widgets on the window.

3. Adding a GUI to a command-line program: Tkinter can be used to add a GUI to a command-line program, making it easier for users to interact with the program and input arguments.

To add a GUI to a command-line program, you can use functions like *Entry* and *Button* to create input fields and buttons, and use event handlers like *command* and *bind* to handle user input.

4. Creating custom widgets: Tkinter includes a variety of built-in widgets, such as buttons, labels, and text boxes, but it also allows you to create your own custom widgets.

To create a custom widget, you can define a class that inherits from the *Widget* class and overrides its methods to define the behavior and appearance of the widget.

5. Prototyping a GUI: Tkinter can be used to quickly prototype a GUI, allowing you to test and iterate on different design ideas before committing to a final implementation.

To prototype a GUI with Tkinter, you can use the *Tk()* function to create a root window, and then use functions like *Label*, *Button*, and *Entry* to add widgets to the window and test different layouts and design ideas.

➢ Getting Started with Tkinter:

1. Import tkinter package and all of its modules.

2. Create a root window. Give the root window a title(using **title()**) and dimension(using **geometry()**). All other widgets will be inside the root window.

3. Use **mainloop()** to call the endless loop of the window. If you forget to call this nothing will appear to the user. The window will wait for any user interaction till we close it.

➢ Some of the common widgets used in Tkinter are :

- **Frame**: serves as a holding area for other widgets and serves as a container.
- **Text**: It enables us to display and alter text in a variety of styles and offers a prepared text display.
- **Label:** Used to display text and images, but we are unable to interact with it.
- **Button:** Often used add buttons and we may add functions and methods to it.
- **Entry:** One-line string text can be entered into this widget.
- **Labelframe:** For intricate window layouts, this widget serves as a separator or container.
- **Listbox:** It just has text elements, all of which are the same colour and font.
- **Scrollbar:** This gives a sliding controller.
- **Canvas:** Custom widgets can be implemented using the canvas widget.
- **Scale**: This widget offers graphical slider items that let us choose different scale values.

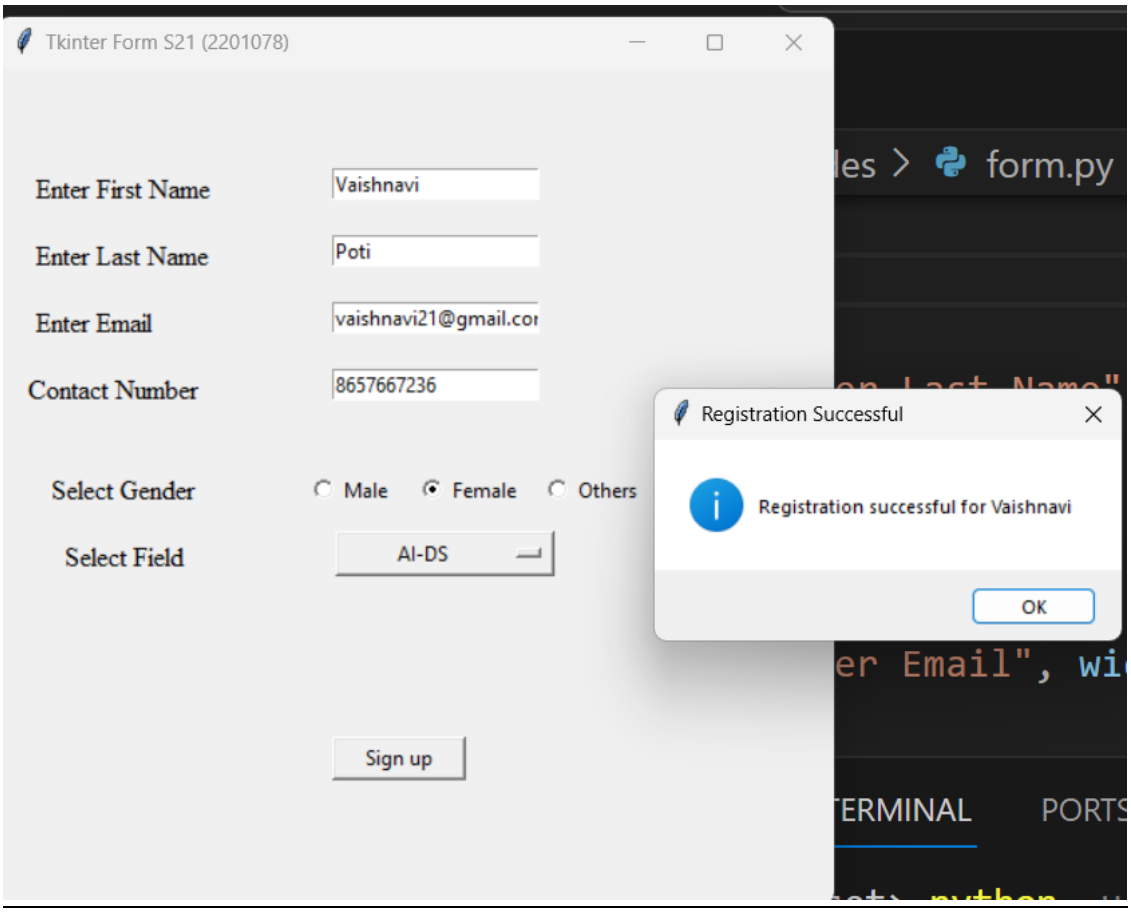
- **Radiabutton**: Use a radio button to carry out one of several choices.
- **Checkbox**: Use a checkbox to implement on-off choices.
- **Listbox**: It just has text elements, all of which are the same colour and font.

➢ PROGRAM:

```
from tkinter import *
from tkinter import messagebox

def register():
    name = en1.get()
    messagebox.showinfo("Registration Successful", f"Registration
successful for {name}")

base = Tk()
base.geometry("500x500")
base.title("Tkinter Form S21 (2201084)")

lb1 = Label(base, text="Enter First Name", font=("Times New
Roman",12))
lb1.place(x=20, y=60)
en1 = Entry(base)
en1.place(x=200, y=60)

lb2 = Label(base, text="Enter Last Name",  font=("Times New
Roman",12))
lb2.place(x=20, y=100)
en2 = Entry(base)
en2.place(x=200, y=100)

lb3 = Label(base, text="Enter Email", width=10, font=("Times New
Roman",12))
lb3.place(x=10, y=140)
en3 = Entry(base)
en3.place(x=200, y=140)

lb4 = Label(base, text="Contact Number", font=("Times New
Roman",12))
lb4.place(x=15, y=180)
en4 = Entry(base)
```

```
en4.place(x=200, y=180)

lb5 = Label(base, text="Select Gender", width=15, font=("Times New
Roman",12))
lb5.place(x=5, y=240)
vars = IntVar()
Radiobutton(base, text="Male", padx=5, variable=vars,
value=1).place(x=180, y=240)
Radiobutton(base, text="Female", padx=10, variable=vars,
value=2).place(x=240, y=240)
Radiobutton(base, text="Others", padx=15, variable=vars,
value=3).place(x=310, y=240)

list_of_fields = ("AI-DS", "IT", "Comps", "EXTC")
cv = StringVar()
drplist = OptionMenu(base, cv, *list_of_fields)
drplist.config(width=15)
cv.set("Select Field")
lb2 = Label(base, text="Select Field", width=13, font=("Times New
Roman",12))
lb2.place(x=14, y=280)
drplist.place(x=200, y=275)

Button(base, text="Sign up", width=10, command=register).place(x=200,
y=400)
base.mainloop()
```

- **OUTPUT:**



- **CONCLUSION:** Hence, we have successfully implemented program on creating GUI with python containing widgets such as labels, textbox, radio, checkboxes and custom dialog boxes; LO 1.

NAME: **Meet Raut**
DIV: **S2-1**
ROLL NO: **2201084**

EXPERIMENT – 6:

- ***AIM:*** To study and implement program on creating GUI with python containing widgets such as labels, textbox, radio, checkboxes and custom dialog boxes.

- ***THEORY:***

Tkinter is a Python Package for creating GUI applications. Python has a lot of GUI frameworks, but Tkinter is the only framework that's built into the Python standard library.

Tkinter has several strengths; it's cross-platform, so the same code works on Windows, macOS, and Linux.

Tkinter is lightweight and relatively painless to use compared to other frameworks. This makes it a compelling choice for building GUI applications in Python, especially for applications where a modern shine is unnecessary, and the top priority is to build something functional and cross-platform quickly.

- **Use Cases of Tkinter:**

1. Creating windows and dialog boxes: Tkinter can be used to create windows and dialog boxes that allow users to interact with your program. These can be used to display information, gather input, or present options to the user.

To create a window or dialog box, you can use the *Tk()* function to create a root window, and then use functions like *Label*, *Button*, and *Entry* to add widgets to the window.

2. Building a GUI for a desktop application: Tkinter can be used to create the interface for a desktop application, including buttons, menus, and other interactive elements.

To build a GUI for a desktop application, you can use functions like *Menu*, *Checkbutton*, and *RadioButton* to create menus and interactive elements and use layout managers like *pack* and *grid* to arrange the widgets on the window.

3. Adding a GUI to a command-line program: Tkinter can be used to add a GUI to a command-line program, making it easier for users to interact with the program and input arguments.

To add a GUI to a command-line program, you can use functions like *Entry* and *Button* to create input fields and buttons, and use event handlers like *command* and *bind* to handle user input.

4. Creating custom widgets: Tkinter includes a variety of built-in widgets, such as buttons, labels, and text boxes, but it also allows you to create your own custom widgets.

To create a custom widget, you can define a class that inherits from the *Widget* class and overrides its methods to define the behavior and appearance of the widget.

5. Prototyping a GUI: Tkinter can be used to quickly prototype a GUI, allowing you to test and iterate on different design ideas before committing to a final implementation.

To prototype a GUI with Tkinter, you can use the *Tk()* function to create a root window, and then use functions like *Label*, *Button*, and *Entry* to add widgets to the window and test different layouts and design ideas.

➢ Getting Started with Tkinter:

1. Import tkinter package and all of its modules.

2. Create a root window. Give the root window a title(using **title()**) and dimension(using **geometry()**). All other widgets will be inside the root window.

3. Use **mainloop()** to call the endless loop of the window. If you forget to call this nothing will appear to the user. The window will wait for any user interaction till we close it.

➢ Some of the common widgets used in Tkinter are :

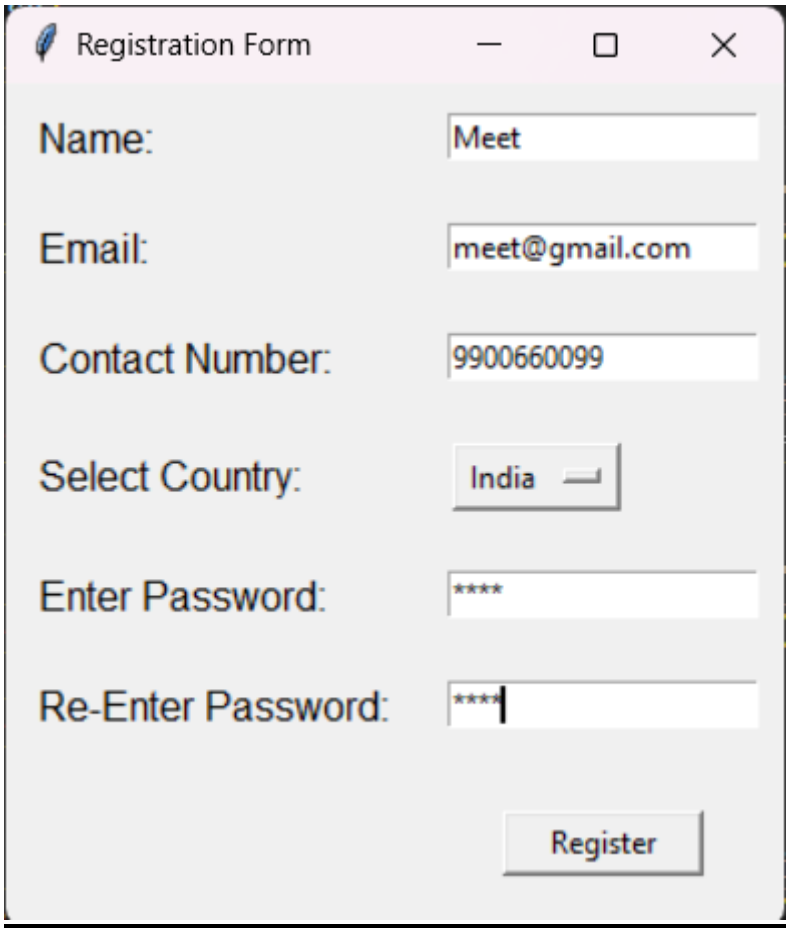
- **Frame**: serves as a holding area for other widgets and serves as a container.
- **Text**: It enables us to display and alter text in a variety of styles and offers a prepared text display.
- **Label:** Used to display text and images, but we are unable to interact with it.
- **Button:** Often used add buttons and we may add functions and methods to it.
- **Entry:** One-line string text can be entered into this widget.
- **Labelframe:** For intricate window layouts, this widget serves as a separator or container.
- **Listbox:** It just has text elements, all of which are the same colour and font.
- **Scrollbar:** This gives a sliding controller.
- **Canvas:** Custom widgets can be implemented using the canvas widget.
- **Scale**: This widget offers graphical slider items that let us choose different scale values.
- **Radiabutton**: Use a radio button to carry out one of several choices.
- **Checkbox**: Use a checkbox to implement on-off choices.
- **Listbox**: It just has text elements, all of which are the same colour and font.

- **PROGRAM:**

```
from tkinter import *

base = Tk()
base.title("Registration Form")

# Name
lb1 = Label(base, text="Name:", font=("arial", 12))
lb1.grid(row=0, column=0, padx=10, pady=10, sticky="w")
en1 = Entry(base)
en1.grid(row=0, column=1, padx=10, pady=10)

# Email
lb3 = Label(base, text="Email:", font=("arial", 12))
lb3.grid(row=1, column=0, padx=10, pady=10, sticky="w")
en3 = Entry(base)
en3.grid(row=1, column=1, padx=10, pady=10)

# Contact Number
lb4 = Label(base, text="Contact Number:", font=("arial", 12))
lb4.grid(row=2, column=0, padx=10, pady=10, sticky="w")
en4 = Entry(base)
en4.grid(row=2, column=1, padx=10, pady=10)

# Select Country
lb2 = Label(base, text="Select Country:", font=("arial", 12))
lb2.grid(row=4, column=0, padx=10, pady=10, sticky="w")
cv = StringVar()
cv.set("Select")
drplist = OptionMenu(base, cv, *("United States", "India", "Nepal",
"Germany"))
drplist.grid(row=4, column=1, padx=10, pady=10, sticky="w")

# Enter Password
lb6 = Label(base, text="Enter Password:", font=("arial", 12))
lb6.grid(row=5, column=0, padx=10, pady=10, sticky="w")
en6 = Entry(base, show='*')
en6.grid(row=5, column=1, padx=10, pady=10)

# Re-Enter Password
lb7 = Label(base, text="Re-Enter Password:", font=("arial", 12))
lb7.grid(row=6, column=0, padx=10, pady=10, sticky="w")
```

```
en7 = Entry(base, show='*')
en7.grid(row=6, column=1, padx=10, pady=10)

# Register Button
Button(base, text="Register", width=10).grid(row=7, column=1,
pady=20)

base.mainloop()
```

- **OUTPUT:**



- ***CONCLUSION:*** Hence, we have successfully implemented program on creating GUI with python containing widgets such as labels, textbox, radio, checkboxes and custom dialog boxes; LO 1.

NAME: **Meet Raut**

DIV: **S2-1**

ROLL NO: **2201084**

EXPERIMENT – 7:

- ***AIM:*** **To append data to existing file and then display the entire file.**
- ***THEORY:***

Append Data to Existing File and Display the Entire File:

In Python, you can use the built-in open() function to open a file in different modes, such as 'r' for reading, 'w' for writing, and 'a' for appending. To append data to an existing file, you can open the file in append mode and then use the write() method to add new content. After appending, you can read and display the entire file.

- **PROGRAM 1:**

```
file=open('hello.txt','r')
print(file.read())

file=open('hello.txt','w')
file.write("Writing PYTHON LAB\n")

file=open('hello.txt','a')
file.write("Appending an text TSEC\n")
file.close()
```

- **OUTPUT:**

```
Writing PYTHON LAB
Appending an text TSEC
This is extra test IndiaThis is extra test India
```

- **PROGRAM 2:**

```
file1 = open("myfile.txt", "w")
L = ["This is Delhi \n", "This is Paris \n", "This is London"]
file1.writelines(L)
file1.close()

# Append-adds at last
# append mode
file1 = open("myfile.txt", "a")

# writing newline character
file1.write("\n")
file1.write("Today")

# without newline character
file1.write("Tomorrow")


file1 = open("myfile.txt", "r")
print("Output of Readlines after appending")
print(file1.read())
print()
file1.close()
```

- **<u>OUTPUT:</u>**

```
Output of Readlines after appending
This is Delhi
This is Paris
This is London
TodayTomorrow
```

- ***<u>CONCLUSION:</u>*** **Hence, we have successfully implemented program on exploring files and directories; LO 2.**

NAME: **Meet Raut**

DIV: **S21**

ROLL NO: **2201084**

EXPERIMENT – 8:

- ***AIM:*** **To count number of lines, words and characters in a file.** .
- ***THEORY:***

Count Number of Lines, Words, and Characters in a File:

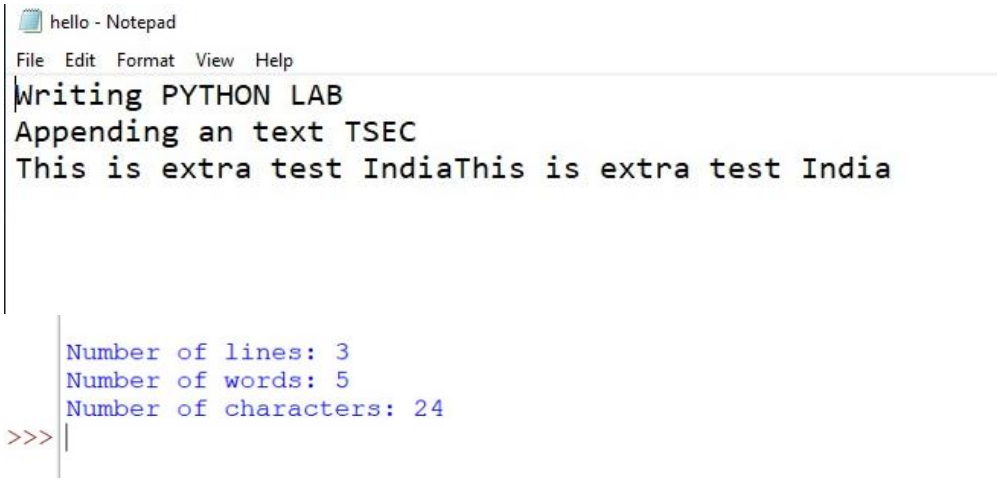
To count the number of lines, words, and characters in a file, you can read the file and then use Python string and list methods to perform the required counts. For example, to count lines, you can split the content based on newline characters (\n). To count words, you can use the split() method, and to count characters, you can use the len() function.

- **PROGRAM 1:**

```
filename= r"C:\Users\Lenovo\Desktop\S21_2201072\hello.txt"
num_lines = 0
num_words = 0
num_chars = 0

with open(filename, 'r') as file:
    for line in file:
        num_lines += 1

words = line.split()
num_words += len(words)
num_chars += len(line)

file =  open(filename, 'a')
extra_text = "This is extra test India"

file.write(extra_text)
file.close()
```

```
print(f"Number of lines: {num_lines}")
print(f"Number of words: {num_words}")
print(f"Number of characters: {num_chars}")
```

- **OUTPUT:**



- ***CONCLUSION:*** **Hence, we have successfully implemented program on exploring files and directories; LO 2.**

NAME: **Meet Raut**

DIV: **S2-1**

ROLL NO: **2201084**

EXPERIMENT – 9:

- ***AIM:*** **To study and implement program to demonstrate use of NumPy: Array objects.**

- ***THEORY:***

Python Numpy:

NumPy stands for numeric python which is a python package for the computation and processing of the multidimensional and single dimensional array elements.
Travis Oliphant created NumPy package in 2005 by injecting the features of the ancestor module Numeric into another module Numarray.

Numpy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays. It is the fundamental package for scientific computing with Python. Besides its obvious scientific uses, Numpy can also be used as an efficient multi-dimensional container of generic data.

Arrays in Numpy:

Array in Numpy is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers. In Numpy, number of dimensions of the array is called rank of the array.A tuple of integers giving the size of the array along each dimension is known as shape of the array. An array class in Numpy is called as ndarray. Elements in Numpy arrays are accessed by using square brackets and can be initialized by using nested Python Lists. Creating a Numpy Array Arrays in Numpy can be created by multiple ways, with various number of Ranks, defining the size of the Array. Arrays can also be created with the use of various data types such as lists, tuples, etc. The type of the resultant array is deduced from the type of the elements in the sequences. Note: Type of array can be explicitly defined while creating the array.

Accessing the array Index: In a numpy array, indexing or accessing the array index can be done in multiple ways. To print a range of an array, slicing is done. Slicing of an array is defining a range in a new array which is used to print a range of elements from the original array. Since, sliced array holds a range of elements of the original array, modifying content with the help of sliced array modifies the original array content.

Basic Array Operations: In numpy, arrays allow a wide range of operations which can be performed on a particular array or a combination of Arrays. These operation include some basic Mathematical operation as well as Unary and Binary operations.

Data Types in Numpy:

Every Numpy array is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers. Every ndarray has an associated data type (dtype) object. This data type object (dtype) provides information about the layout of the array. The values of an ndarray are stored in a buffer which can be thought of as a contiguous block of memory bytes which can be interpreted by the dtype object. Numpy provides a large set of numeric datatypes that can be used to construct arrays. At the time of Array creation, Numpy tries to guess a datatype, but functions that construct arrays usually also include an optional argument to explicitly specify the datatype.

Constructing a Datatype Object: In Numpy, datatypes of Arrays need not to be defined unless a specific datatype is required. Numpy tries to guess the datatype for Arrays which are not predefined in the constructor function.

Math Operations on DataType array: In Numpy arrays, basic mathematical operations are performed element-wise on the array. These operations are applied both as operator overloads and as functions. Many useful functions are provided in Numpy for performing computations on Arrays such as **sum**: for addition of Array elements, **T**: for Transpose of elements, etc.

The need of NumPy:

With the revolution of data science, data analysis libraries like NumPy, SciPy, Pandas, etc. have seen a lot of growth. With a much easier syntax than other programming languages, python is the first choice language for the data scientist.

NumPy provides a convenient and efficient way to handle the vast amount of data. NumPy is also very convenient with Matrix multiplication and data reshaping. NumPy is fast which makes it reasonable to work with a large set of data.

There are the following advantages of using NumPy for data analysis.

1. NumPy performs array-oriented computing.
2. It efficiently implements the multidimensional arrays.
3. It performs scientific computations.
4. It is capable of performing Fourier Transform and reshaping the data stored in multidimensional arrays.
5. NumPy provides the in-built functions for linear algebra and random number generation.

Nowadays, NumPy in combination with SciPy and Mat-plotlib is used as the replacement to MATLAB as Python is more complete and easier programming language than MATLAB.

NumPy Ndarray:

Ndarray is the n-dimensional array object defined in the numpy which stores the collection of the similar type of elements. In other words, we can define a ndarray as the collection of the data type (dtype) objects.

The ndarray object can be accessed by using the 0 based indexing. Each element of the Array object contains the same size in the memory.

Creating a ndarray object

The ndarray object can be created by using the array routine of the numpy module. For this purpose, we need to import the numpy.

The parameters are described in the following table.

SN	Parameter	Description
1	object	It represents the collection object. It can be a list, tuple, dictionary, set, etc.
2	dtype	We can change the data type of the array elements by changing this option to the specified type. The default is none.
3	copy	It is optional. By default, it is true which means the object is copied.
4	order	There can be 3 possible values assigned to this option. It can be C (column order), R (row order), or A (any)
5	subok	The returned array will be base class array by default. We can change this to make the subclasses passes through by setting this option to true.
6	ndmin	It represents the minimum dimensions of the resultant array.

Finding the dimensions of the Array

The ndim function can be used to find the dimensions of the array.

Finding the size of each array element

The itemsize function is used to get the size of each array item. It returns the number of bytes taken by each array element.

Finding the data type of each array item

To check the data type of each array item, the dtype function is used. Consider the following example to check the data type of the array items.

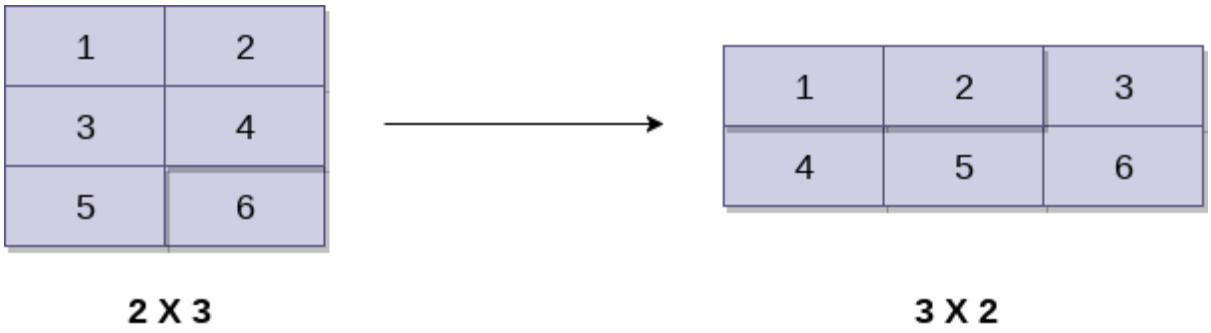
Finding the shape and size of the array

To get the shape and size of the array, the size and shape function associated with the numpy array is used.

Reshaping the array objects

By the shape of the array, we mean the number of rows and columns of a multi-dimensional array. However, the numpy module provides us the way to reshape the array by changing the number of rows and columns of the multi-dimensional array.

The reshape() function associated with the ndarray object is used to reshape the array. It accepts the two parameters indicating the row and columns of the new shape of the array.



Slicing in the Array

Slicing in the NumPy array is the way to extract a range of elements from an array. Slicing in the array is performed in the same way as it is performed in the python list.

Linspace

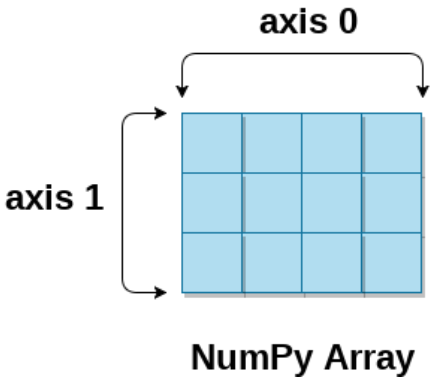
The linspace() function returns the evenly spaced values over the given interval.

Finding the maximum, minimum, and sum of the array elements

The NumPy provides the max(), min(), and sum() functions which are used to find the maximum, minimum, and sum of the array elements respectively.

NumPy Array Axis

A NumPy multi-dimensional array is represented by the axis where axis-0 represents the columns and axis-1 represents the rows. We can mention the axis to perform row-level or column-level calculations like the addition of row or column elements.



- **PROGRAM:**

```
import numpy as np

a = np.array([1, 22, 3, 10, 15, 4, 55, 23, 44, 70])
b = np.zeros((2, 3, 4, 5))
d = np.full((3, 3, 3), 2)

print("1D array:", a)

print("The maximum element:", a.max())
print("The minimum element:", a.min())
print("The Length of array is:", len(a))
print("The sum of the elements:", a.sum())

print("Numbers after the 2nd index:", a[2:])
print("Last 2 numbers:", a[-2:])
```

```
print("-----------------------------------------------")



print("\n3D array with all 2s:")
print(d)

print("-----------------------------------------------")

print("4D array with all 0s:")
print(b)

print("-----------------------------------------------")

A = np.array([[11, 12, 13], [14, 15, 16], [17, 18, 19]])
C = np.array([[9, 8, 7], [6, 5, 4], [3, 2, 1]])

D = np.add(A, C)


print("Matrix A:")
print(A)
print("\nMatrix C:")
print(C)

print("\nAddition of matrix A & C:")
print(D)

print("-----------------------------------------------")

x = np.random.randint(100)
print("Random Number:", x)

print("-----------------------------------------------")

matrices = [np.random.randint(1, 4, size=(3, 3)) for _ in range(4)]
```

```
for i, matrix in enumerate(matrices):
    print(f"\nMatrix {i+1}:")
    print("Random matrix with no b/w 1 & 3:")
    print(matrix)

print("-----------------------------------------------")

array1 = np.arange(0, 15, 3)
print("ARANGE ARRAY:")
print(array1)

print("-----------------------------------------------")

array2 = np.linspace(0, 40, 5)
print("LINESPACE FUNCTION:")
print(array2)

print("-----------------------------------------------")

a1 = np.random.randint(1, 10, size=(3, 3))
b1 = np.random.randint(1, 10, size=(3, 3))
c1 = np.random.randint(1, 10, size=(3, 3))

print("Array a1:")
print(a1)
print("\nArray b1:")
print(b1)
print("\nArray c1:")
print(c1)

min_a1 = np.min(a1)
max_a1 = np.max(a1)
std_a1 = np.std(a1)
avg_a1 = np.average(a1)
var_a1 = np.var(a1)
med_a1 = np.median(a1)
mean_a1 = np.mean(a1)
```

```
min_b1 = np.min(b1)
max_b1 = np.max(b1)
std_b1 = np.std(b1)
avg_b1 = np.average(b1)
var_b1 = np.var(b1)
med_b1 = np.median(b1)
mean_b1 = np.mean(b1)

min_c1 = np.min(c1)
max_c1 = np.max(c1)
std_c1 = np.std(c1)
avg_c1 = np.average(c1)
var_c1 = np.var(c1)
med_c1 = np.median(c1)
mean_c1 = np.mean(c1)

print("\n---FOR MATRIX a1---")
print("Min of a1:", min_a1)
print("Max of a1:", max_a1)
print("Standard deviation of a1:", std_a1)
print("Average of a1:", avg_a1)
print("Variance of a1:", var_a1)
print("Median of a1:", med_a1)
print("Mean of a1:", mean_a1)

print("\n---FOR MATRIX b1---")
print("Min of b1:", min_b1)
print("Max of b1:", max_b1)
print("Standard deviation of b1:", std_b1)
print("Average of b1:", avg_b1)
print("Variance of b1:", var_b1)
print("Median of b1:", med_b1)
print("Mean of b1:", mean_b1)

print("\n---FOR MATRIX c1---")
print("Min of c1:", min_c1)
```

```
print("Max of c1:", max_c1)
print("Standard deviation of c1:", std_c1)
print("Average of c1:", avg_c1)
print("Variance of c1:", var_c1)
print("Median of c1:", med_c1)
print("Mean of c1:", mean_c1)
```

- **OUTPUT:**

```
1D array: [ 1 22  3 10 15  4 55 23 44 70]
The maximum element: 70
The minimum element: 1
The Length of array is: 10
The sum of the elements: 247
Numbers after the 2nd index: [ 3 10 15  4 55 23 44 70]
Last 2 numbers: [44 70]
-------------------------------------------------

3D array with all 2s:
[[[2 2 2]
  [2 2 2]
  [2 2 2]]

 [[2 2 2]
  [2 2 2]
  [2 2 2]]

 [[2 2 2]
  [2 2 2]
  [2 2 2]]]
-------------------------------------------------
```

```
------------------------------------------------
4D array with all 0s:
[[[[0. 0. 0. 0. 0.]
   [0. 0. 0. 0. 0.]
   [0. 0. 0. 0. 0.]
   [0. 0. 0. 0. 0.]]

  [[0. 0. 0. 0. 0.]
   [0. 0. 0. 0. 0.]
   [0. 0. 0. 0. 0.]
   [0. 0. 0. 0. 0.]]

  [[0. 0. 0. 0. 0.]
   [0. 0. 0. 0. 0.]
   [0. 0. 0. 0. 0.]
   [0. 0. 0. 0. 0.]]]


 [[[0. 0. 0. 0. 0.]
   [0. 0. 0. 0. 0.]
   [0. 0. 0. 0. 0.]
   [0. 0. 0. 0. 0.]]

  [[0. 0. 0. 0. 0.]
   [0. 0. 0. 0. 0.]
   [0. 0. 0. 0. 0.]
   [0. 0. 0. 0. 0.]]

  [[0. 0. 0. 0. 0.]
   [0. 0. 0. 0. 0.]
   [0. 0. 0. 0. 0.]
   [0. 0. 0. 0. 0.]]]]
```

```
------------------------------------------------
Matrix A:
[[11 12 13]
 [14 15 16]
 [17 18 19]]

Matrix C:
[[9 8 7]
 [6 5 4]
 [3 2 1]]

Addition of matrix A & C:
[[20 20 20]
 [20 20 20]
 [20 20 20]]
------------------------------------------------
Random Number: 9
------------------------------------------------

Matrix 1:
Random matrix with no b/w 1 & 3:
[[2 2 1]
 [2 2 1]
 [3 3 3]]

Matrix 2:
Random matrix with no b/w 1 & 3:
[[1 1 1]
 [1 2 2]
 [1 2 3]]
```

```
Matrix 3:
Random matrix with no b/w 1 & 3:
[[2 1 2]
 [3 2 3]
 [3 1 1]]

Matrix 4:
Random matrix with no b/w 1 & 3:
[[1 1 2]
 [1 1 2]
 [3 1 1]]
--------------------------------------------------
ARANGE ARRAY:
[ 0  3  6  9 12]
--------------------------------------------------
LINESPACE FUNCTION:
[ 0. 10. 20. 30. 40.]
--------------------------------------------------
Array a1:
[[9 7 7]
 [9 9 5]
 [8 8 2]]

Array b1:
[[3 6 3]
 [2 5 5]
 [8 5 6]]
```

```
Array c1:
[[9 6 1]
 [6 6 1]
 [8 1 8]]

---FOR MATRIX a1---
Min of a1: 2
Max of a1: 9
Standard deviation of a1: 2.1829869671542776
Average of a1: 7.111111111111111
Variance of a1: 4.765432098765432
Median of a1: 8.0
Mean of a1: 7.111111111111111

---FOR MATRIX b1---
Min of b1: 2
Max of b1: 8
Standard deviation of b1: 1.7497795275581802
Average of b1: 4.777777777777778
Variance of b1: 3.0617283950617282
Median of b1: 5.0
Mean of b1: 4.777777777777778

---FOR MATRIX c1---
Min of c1: 1
Max of c1: 9
Standard deviation of c1: 3.0711722135745005
Average of c1: 5.111111111111111
Variance of c1: 9.432098765432098
Median of c1: 6.0
Mean of c1: 5.111111111111111
```

- ***CONCLUSION:*** **Hence, we have successfully implemented program to demonstrate use of NumPy: Array objects; LO 1.**

NAME: **Meet Raut**

DIV: **S2-1**

ROLL NO: **2201084**

EXPERIMENT – 10:

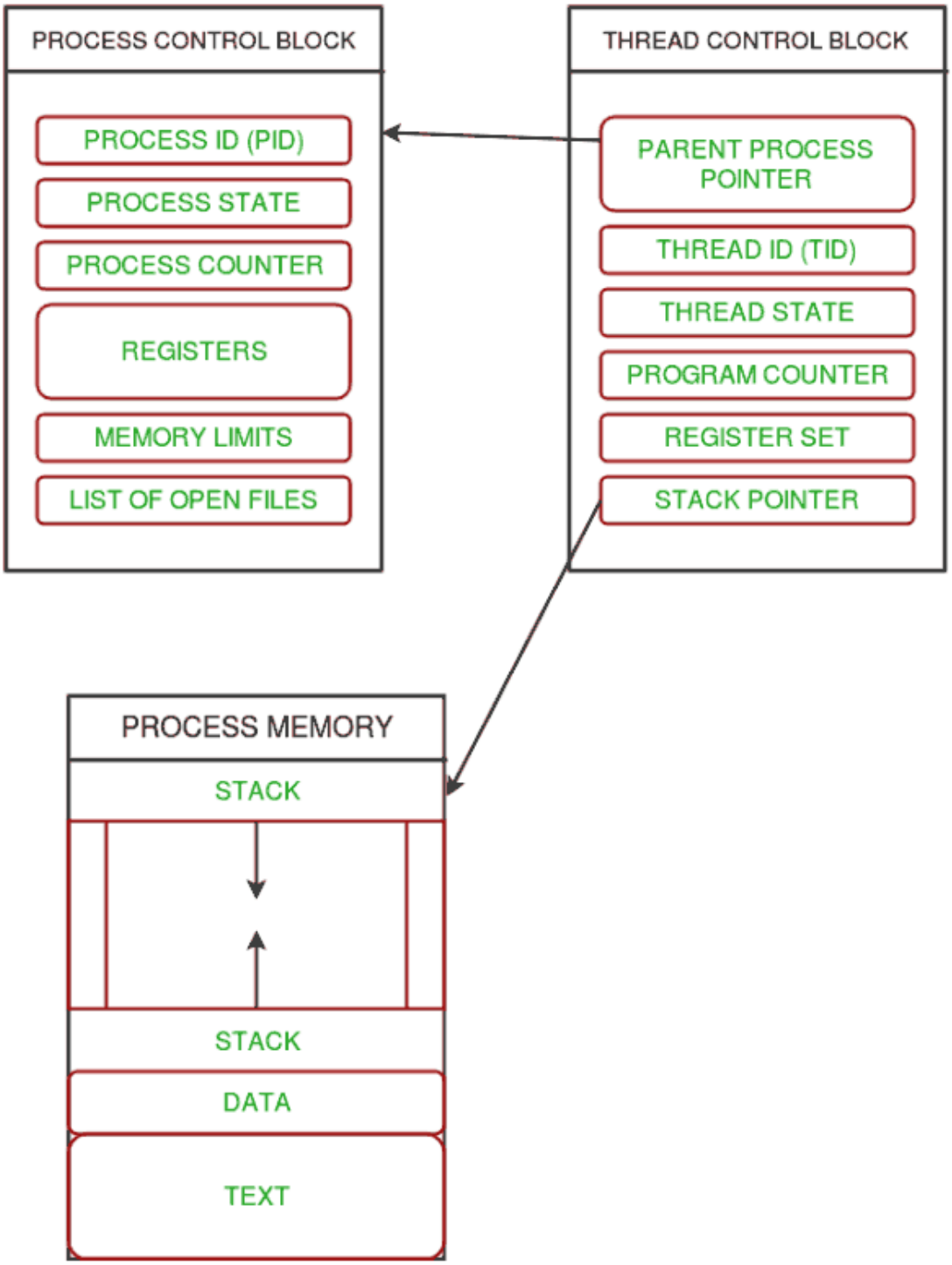
- ***AIM:*** **To study and implement programs on Threading using python.**

- ***THEORY:***

An Intro to Python Threading:

A **thread** is an entity within a process that can be scheduled for execution. Also, it is the smallest unit of processing that can be performed in an OS (Operating System). In simple words, a thread is a sequence of such instructions within a program that can be executed independently of other code. For simplicity, you can assume that a thread is simply a subset of a process! A thread contains all this information in a **Thread Control Block (TCB)**:

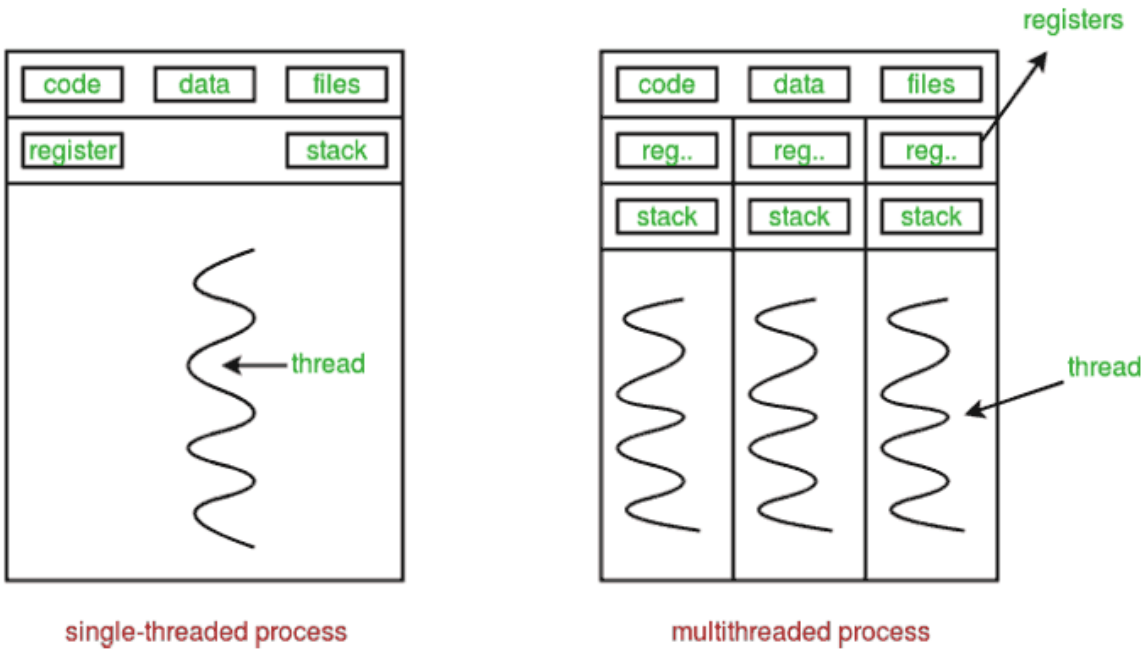
- **Thread Identifier:** Unique id (TID) is assigned to every new thread
- **Stack pointer:** Points to the thread's stack in the process. The stack contains the local variables under the thread's scope.
- **Program counter:** a register that stores the address of the instruction currently being executed by a thread.
- **Thread state:** can be running, ready, waiting, starting, or done.
- **Thread's register set:** registers assigned to thread for computations.
- **Parent process Pointer:** A pointer to the Process control block (PCB) of the process that the thread lives on.



Multiple threads can exist within one process where:

- Each thread contains its own **register set** and **local variables (stored in the stack)**.
- All threads of a process share **global variables (stored in heap)** and the **program code**.

Consider the diagram below to understand how multiple threads exist in memory:

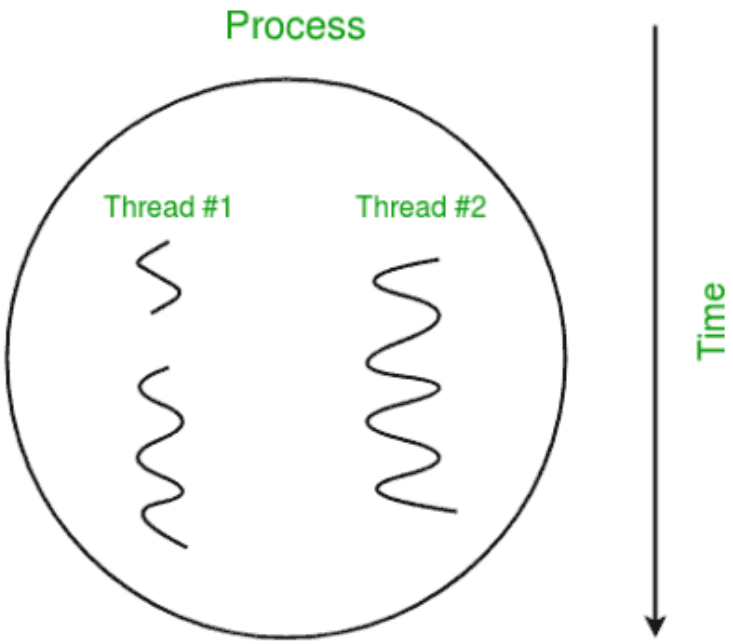


single-threaded process

multithreaded process

Python Multithreading:

Multithreading is a stringing procedure in Python programming to run various strings simultaneously by quickly exchanging between strings with a central processor help (called setting exchanging). In addition, it makes it possible to share its data space with the primary threads of a process, making it easier than with individual processes to share information and communicate with other threads. The goal of multithreading is to complete multiple tasks at the same time, which improves application rendering and performance.



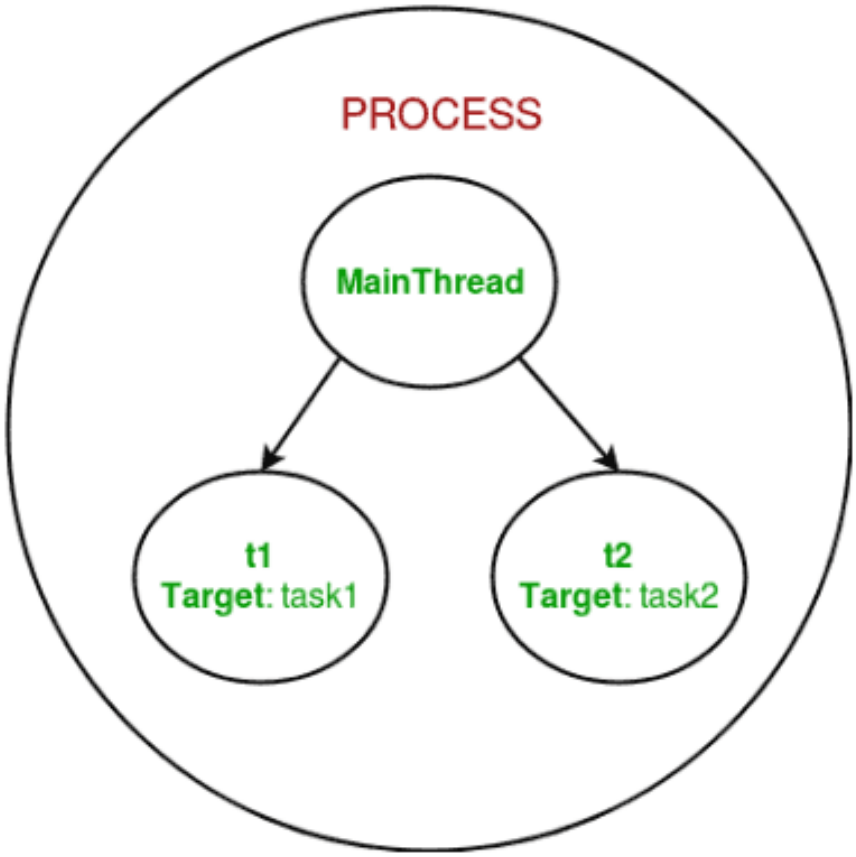
Benefits of Using Python for Multithreading:

The following are the advantages of using Python for multithreading:

1. It guarantees powerful usage of PC framework assets.
2. Applications with multiple threads respond faster.
3. It is more cost-effective because it shares resources and its state with sub-threads (child).
4. It makes the multiprocessor engineering more viable because of closeness.
5. By running multiple threads simultaneously, it cuts down on time.
6. To store multiple threads, the system does not require a lot of memory.

When to use Multithreading in Python?

It is an exceptionally valuable strategy for efficient and working on the presentation of an application. Programmers can run multiple subtasks of an application at the same time by using multithreading. It lets threads talk to the same processor and share resources like files, data, and memory. In addition, it makes it easier for the user to continue running a program even when a portion of it is blocked or too long.



How to achieve multithreading in Python?

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

1. The thread module
2. 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:

```
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:

```
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.

For example:

```
import threading
def print_hello(n):
print("Hello, how old are you ", n)
t1 = threading.Thread( target = print_hello, args =(18, ))
```

In the above code, we invoked the **print_hello()** function as the target parameter. The **print_hello()** contains one parameter **n**, which passed to the **args** parameter.

3. 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.

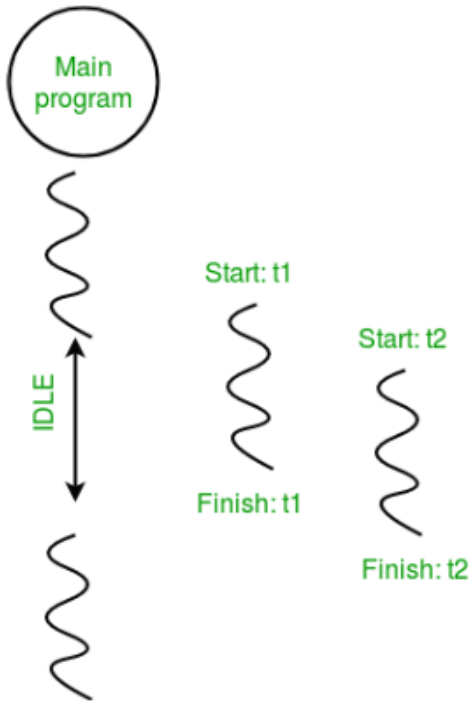
Syntax:

```
t1.start()
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. The join() method stops the main thread from running when the above program is run and waits until the thread t1 has finished running. The main thread begins its execution once the t1 has completed successfully.

5. Synchronizing Threads in Python

It is a thread synchronization mechanism that makes sure that no two threads can run the same part of the program at the same time to access shared resources. Critical sections could be used to describe the situation. To avoid the critical section condition, in which two threads cannot simultaneously access resources, we employ a race condition.



Python ThreadPool:

A thread pool is a collection of threads that are created in advance and can be reused to execute multiple tasks. The concurrent.futures module in Python provides a ThreadPoolExecutor class that makes it easy to create and manage a thread pool.
In this example, we define a function worker that will run in a thread. We create a ThreadPoolExecutor with a maximum of 2 worker threads. We then submit two tasks to the pool using the submit method. The pool manages the execution of the tasks in its worker threads. We use the shutdown method to wait for all tasks to complete before the main thread continues.
Multithreading can help you make your programs more efficient and responsive. However, it’s important to be careful when working with threads to avoid issues such as race conditions and deadlocks.
This code uses a thread pool created with **concurrent.futures.ThreadPoolExecutor** to run two worker tasks concurrently. The main thread waits for the worker threads to finish using **pool.shutdown(wait=True)**. This allows for efficient parallel processing of tasks in a multi-threaded environment.

- **PROGRAM 1:**

```
import thread # import the thread module
import time # import time module

def cal_sqre(num): # define the cal_sqre function
    print(" Calculate the square root of the given number")
    for n in num:
        time.sleep(0.3) # at each iteration it waits for 0.3 time
        print(' Square is : ', n * n)

def cal_cube(num): # define the cal_cube() function
    print(" Calculate the cube of  the given number")
    for n in num:
        time.sleep(0.3) # at each iteration it waits for 0.3 time
        print(" Cube is : ", n * n *n)

arr = [4, 5, 6, 7, 2] # given array

t1 = time.time() # get total time to execute the functions
cal_sqre(arr) # call cal_sqre() function
cal_cube(arr) # call cal_cube() function

print(" Total time taken by threads is :", time.time() - t1) # print the total time
```

- **OUTPUT:**

```
Calculate the square root of the given number
 Square is:  16
 Square is:  25
 Square is:  36
 Square is:  49
 Square is:  4
 Calculate the cube of the given number
 Cube is:  64
 Cube is:  125
 Cube is:  216
 Cube is:  343
 Cube is:  8
 Total time taken by threads is: 3.005793809890747
```

- **PROGRAM 2:**

```
import time # import time module
import threading
from threading import *
def cal_sqre(num): # define a square calculating function
    print(" Calculate the square root of the given number")
    for n in num: # Use for loop
        time.sleep(0.3) # at each iteration it waits for 0.3 time
        print(' Square is : ', n * n)

def cal_cube(num): # define a cube calculating function
    print(" Calculate the cube of  the given number")
    for n in num: # for loop
        time.sleep(0.3) # at each iteration it waits for 0.3 time
        print(" Cube is : ", n * n *n)
```

```
ar = [4, 5, 6, 7, 2] # given array

t = time.time() # get total time to execute the functions
#cal_cube(ar)
#cal_sqre(ar)
th1 = threading.Thread(target=cal_sqre, args=(ar, ))
th2 = threading.Thread(target=cal_cube, args=(ar, ))
th1.start()
th2.start()
th1.join()
th2.join()
print(" Total time taking by threads is :", time.time() - t) # print the total time
print(" Again executing the main thread")
print(" Thread 1 and Thread 2 have finished their execution.")
```

- **OUTPUT:**

```
Calculate the square root of the given number
 Calculate the cube of the given number
 Square is:  16
 Cube is:  64
 Square is:  25
 Cube is:  125
 Square is:  36
 Cube is:  216
 Square is:  49
 Cube is:  343
 Square is:  4
 Cube is:  8
 Total time taken by threads is: 1.5140972137451172
 Again executing the main thread
 Thread 1 and Thread 2 have finished their execution.
```

- **PROGRAM 3:**

```
import threading

def print_cube(num):
        print("Cube: {}" .format(num * num * num))

def print_square(num):
        print("Square: {}" .format(num * num))

if __name__ =="__main__":
        t1 = threading.Thread(target=print_square, args=(10,))
        t2 = threading.Thread(target=print_cube, args=(10,))

        t1.start()
        t2.start()

        t1.join()
        t2.join()
```

```
    print("Done!")
```

- **OUTPUT:**

```
Square: 100
Cube: 1000
Done!
```

- **PROGRAM 4:**

```
import threading
import os

def task1():
    print("Task 1 assigned to thread:
{}".format(threading.current_thread().name))
    print("ID of process running task 1: {}".format(os.getpid()))

def task2():
    print("Task 2 assigned to thread:
{}".format(threading.current_thread().name))
    print("ID of process running task 2: {}".format(os.getpid()))

if __name__ == "__main__":

    print("ID of process running main program: {}".format(os.getpid()))

    print("Main thread name: {}".format(threading.current_thread().name))

    t1 = threading.Thread(target=task1, name='t1')
    t2 = threading.Thread(target=task2, name='t2')

    t1.start()
```

```
    t2.start()

    t1.join()
    t2.join()
```

- **OUTPUT:**

```
ID of process running main program: 1141
Main thread name: MainThread
Task 1 assigned to thread: t1
ID of process running task 1: 1141
Task 2 assigned to thread: t2
ID of process running task 2: 1141
```

- **PROGRAM 5:**

```
import concurrent.futures

def worker():
    print("Worker thread running")

pool = concurrent.futures.ThreadPoolExecutor(max_workers=2)

pool.submit(worker)
pool.submit(worker)

pool.shutdown(wait=True)

print("Main thread continuing to run")
```

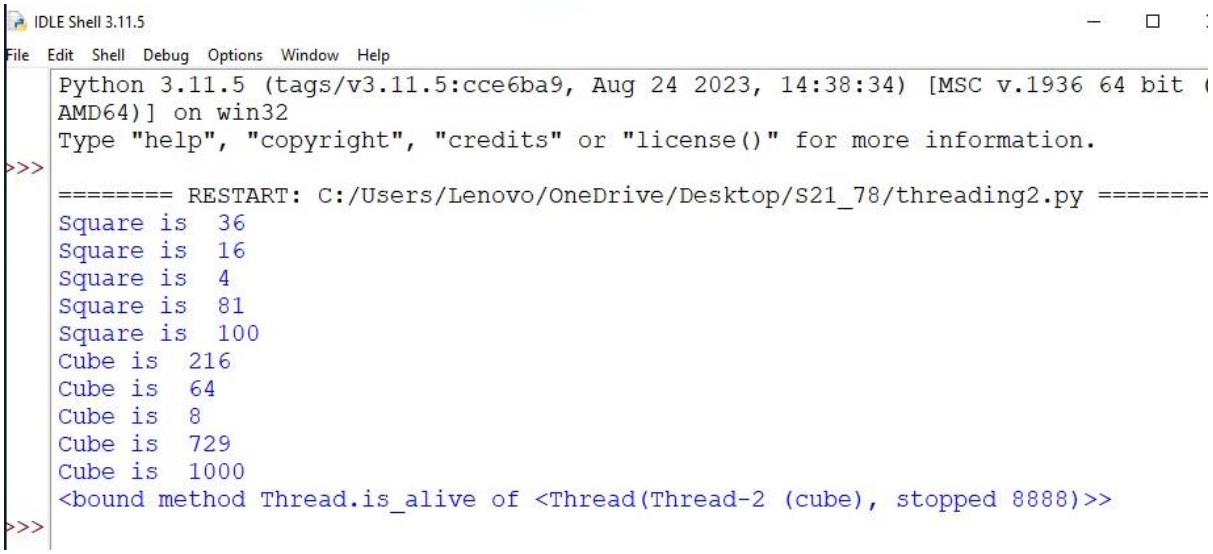
- **OUTPUT:**

```
Worker thread running
Worker thread running
Main thread continuing to run
```

- **PROGRAM 6:**

```
import threading
import thread
import time

def squares(num):
    for n in num:
        time.sleep(0.3)
        print("Square is ",n*n)

def cube(num):
    time.sleep(0.3)
    print ("Cube is ",n*n*n)

arr=[6,4,2,9,10]

t1=threading.Thread(target=sqares, args=(arr, ))
t2=threading.Thread(target=cube, args=(arr, ))
t1.start()
t1.join()
t2.join()
print(t2.is_alive)
t1._stop()
```

- **OUTPUT:**



- ***CONCLUSION:*** **Hence, we have successfully implemented program on Threading using python; LO 5.**

NAME: **Meet Raut**

DIV: **S2-1**

ROLL NO: **2201084**

EXPERIMENT – 11:

- ***AIM:*** **To study and implement program on Exception Handling.**
- ***THEORY:***

What is an Exception?

An exception in Python is an incident that happens while executing a program that causes the regular course of the program's commands to be disrupted. When a Python code comes across a condition it can't handle, it raises an exception. An object in Python that describes an error is called an exception.

When a Python code throws an exception, it has two options: handle the exception immediately or stop and quit.

Try and Except Statement - Catching Exceptions

In Python, we catch exceptions and handle them using try and except code blocks. The try clause contains the code that can raise an exception, while the except clause contains the code lines that handle the exception.

How to Raise an Exception

If a condition does not meet our criteria but is correct according to the Python interpreter, we can intentionally raise an exception using the raise keyword. We can use a customized exception in conjunction with the statement.

Assertions in Python

When we're finished verifying the program, an assertion is a consistency test that we can switch on or off.

The simplest way to understand an assertion is to compare it with an if-then condition. An exception is thrown if the outcome is false when an expression is evaluated.

Assertions are made via the assert statement, which was added in Python 1.5 as the latest keyword.

Assertions are commonly used at the beginning of a function to inspect for valid input and at the end of calling the function to inspect for valid output.

The assert Statement

Python examines the adjacent expression, preferably true when it finds an assert statement. Python throws an AssertionError exception if the result of the expression is false.

The syntax for the assert clause is –

```
assert Expressions[, Argument]
```

Python uses ArgumentException, if the assertion fails, as the argument for the AssertionError. We can use the try-except clause to catch and handle AssertionError exceptions, but if they aren't, the program will stop, and the Python interpreter will generate a traceback.

Try with Else Clause

Python also supports the else clause, which should come after every except clause, in the try, and except blocks. Only when the try clause fails to throw an exception the Python interpreter goes on to the else block.

Finally Keyword in Python

The finally keyword is available in Python, and it is always used after the try-except block. The finally code block is always executed after the try block has terminated normally or after the try block has terminated for some other reason.

User-Defined Exceptions

By inheriting classes from the typical built-in exceptions, Python also lets us design our customized exceptions.

Here is an illustration of a RuntimeError. In this case, a class that derives from RuntimeError is produced. Once an exception is detected, we can use this to display additional detailed information.

We raise a user-defined exception in the try block and then handle the exception in the except block.

Python Exceptions List

Here is the complete list of Python in-built exceptions.

Sr.No.	Name of the Exception	Description of the Exception
1	**Exception**	All exceptions of Python have a base class.
2	**StopIteration**	If the next() method returns null for an iterator, this exception is raised.
3	**SystemExit**	The sys.exit() procedure raises this value.
4	**StandardError**	Excluding the StopIteration and SystemExit, this is the base class for all Python built-in exceptions.
5	**ArithmeticError**	All mathematical computation errors belong to this base class.
6	**OverflowError**	This exception is raised when a computation surpasses the numeric data type's maximum limit.
7	**FloatingPointError**	If a floating-point operation fails, this exception is raised.
8	**ZeroDivisionError**	For all numeric data types, its value is raised whenever a number is attempted to be divided by zero.
9	**AssertionError**	If the Assert statement fails, this exception is raised.
10	**AttributeError**	This exception is raised if a variable reference or assigning a value fails.
11	**EOFError**	When the endpoint of the file is approached, and the interpreter didn't get any input value by raw_input() or input() functions, this exception is raised.
12	**ImportError**	This exception is raised if using the import keyword to import a module fails.

13	**KeyboardInterrupt**	If the user interrupts the execution of a program, generally by hitting Ctrl+C, this exception is raised.
14	**LookupError**	LookupErrorBase is the base class for all search errors.
15	**IndexError**	This exception is raised when the index attempted to be accessed is not found.
16	**KeyError**	When the given key is not found in the dictionary to be found in, this exception is raised.
17	**NameError**	This exception is raised when a variable isn't located in either local or global namespace.
18	**UnboundLocalError**	This exception is raised when we try to access a local variable inside a function, and the variable has not been assigned any value.
19	**EnvironmentError**	All exceptions that arise beyond the Python environment have this base class.
20	**IOError**	If an input or output action fails, like when using the print command or the open() function to access a file that does not exist, this exception is raised.
22	**SyntaxError**	This exception is raised whenever a syntax error occurs in our program.
23	**IndentationError**	This exception was raised when we made an improper indentation.
24	**SystemExit**	This exception is raised when the sys.exit() method is used to terminate the Python interpreter. The parser exits if the situation is not addressed within the code.
25	**TypeError**	This exception is raised whenever a data type-incompatible action or function is tried to be executed.
26	**ValueError**	This exception is raised if the parameters for a built-in method for a particular data type are of the correct type but have been given the wrong values.

27	**RuntimeError**	This exception is raised when an error that occurred during the program's execution cannot be classified.
28	**NotImplementedError**	If an abstract function that the user must define in an inherited class is not defined, this exception is raised.

- **PROGRAM:**

```
for i in range (1,10):
    print("\n1. Zero Division Error\n 2. Value Error\n 3. Type Error\n 4. Index Error\n 5. Assertion Error")


    ch=int(input("Enter Your Choice: "))

    if(ch==1):
        try:
            a=int(input("Enter a: "))
            b=int(input("Enter b: "))
            d=a/b
            print("Division: ",d)
        except ZeroDivisionError:
            print("Zero division error occurs")
        print("Test")

    elif(ch==2):
        try:
            a=int(input("Enter a: "))
            b=int(input("Enter b: "))
            d=a/b
```

```
        print("Division: ",d)
    except ValueError:
        print("Value error occurs")
    print("Test")

elif(ch==3):
    try:
        a=int(input("Enter a: "))
        b=int(input("Enter b: "))
        d=a/'b'
        print("Division: ",d)
    except TypeError:
        print("Type error occurs")
    print("Test")

elif(ch==4):
    try:
        a=[1,2,3,4]
        print("Index 5= ",a[5])
    except IndexError:
        print("Index error occurs")
    print("Test")

elif(ch==5):
    a=int(input("Enter number b/w 2 to 6: "))
    try:
        assert a>=2 and a<=6
        print("Entered number is: ",a)
    except AssertionError:
```

```
        print("Assertion error occurs")
    print("Test")

else:
    print("Invalid Choice")

print("Thank You")
```

- **OUTPUT:**

```
1. Zero Division Error
 2. Value Error
 3. Type Error
 4. Index Error
 5. Assertion Error
Enter Your Choice: 1
Enter a: 12
Enter b: 6
Division:  2.0
Test
Thank You

1. Zero Division Error
 2. Value Error
 3. Type Error
 4. Index Error
 5. Assertion Error
Enter Your Choice: 1
Enter a: 12
Enter b: 0
Zero division error occurs
Test
Thank You
```

```
1. Zero Division Error
 2. Value Error
 3. Type Error
 4. Index Error
 5. Assertion Error
Enter Your Choice: 2
Enter a: 2
Enter b: a
Value error occurs
Test
Thank You
```

```
1. Zero Division Error
 2. Value Error
 3. Type Error
 4. Index Error
 5. Assertion Error
Enter Your Choice: 2
Enter a: 2
Enter b: 2
Division:  1.0
Test
Thank You
```

```
1. Zero Division Error
 2. Value Error
 3. Type Error
 4. Index Error
 5. Assertion Error
Enter Your Choice: 3
Enter a: 12
Enter b: 2
Type error occurs
Test
Thank You
```

```
1. Zero Division Error
 2. Value Error
 3. Type Error
 4. Index Error
 5. Assertion Error
Enter Your Choice: 4
Index error occurs
Test
Thank You
```

```
1. Zero Division Error
 2. Value Error
 3. Type Error
 4. Index Error
 5. Assertion Error
Enter Your Choice: 5
Enter number b/w 2 to 6: 3
Entered number is:  3
Test
Thank You

1. Zero Division Error
 2. Value Error
 3. Type Error
 4. Index Error
 5. Assertion Error
Enter Your Choice: 5
Enter number b/w 2 to 6: 7
Assertion error occurs
Test
Thank You
```

```
1. Zero Division Error
 2. Value Error
 3. Type Error
 4. Index Error
 5. Assertion Error
Enter Your Choice: 6
Invalid Choice
Thank You

1. Zero Division Error
 2. Value Error
 3. Type Error
 4. Index Error
 5. Assertion Error
Enter Your Choice:
```

- ***CONCLUSION:*** **Hence, we have successfully implemented program on Exception Handling; LO 1.**

NAME: **Meet Anand Raut**

DIV: **S2-1**

ROLL NO: **2201084**

EXPERIMENT – 12:

- ***AIM:*** **To study and implement program on demonstrating CRUD (create, red, update and delete) operations on database (SQLite/MySQL) using python**

- ***THEORY:***

Python SQLite – CRUD Operations:

CRUD Operations: The abbreviation CRUD expands to Create, Read, Update and Delete. These four are fundamental operations in a database. In the sample database, we will create it, and do some operations. Let's discuss these operations one by one with the help of examples.

CREATE

The create command is used to create the table in database.

Syntax: *CREATE TABLE table_name ($Attr_1$ $Type_1$, $Attr_2$ $Type_2$, ... , $Attr_n$ $Type_n$) ;*

INSERT

This refers to the insertion of new data into the table. Data is inserted in the form of a tuple. The number of attributes in the tuple must be equal to that defined in the relation schema while creating the table.

1. To insert attributes in the order specified in the relation schema:

***Syntax: INSERT INTO** tableName **VALUES** (value1, value2, ... valuen)*

2.To insert attributes in the order specified in the relation schema or in a different order:

***INSERT INTO** tableName (Attribute1, Attribute3, Attribute2 . . .) **VALUES** (value1, value3, value2 . . .)*

READ

This refers to reading data from a database. A read statement has three clauses:

1. **SELECT:** Takes as the predicate the attributes to be queried, use ***** for all attributes.
2. **FROM:** Takes as the predicate a relation.
3. **WHERE:** Takes as the predicate a condition, this is not compulsory.

***Example: SELECT** NAME, POINTS, ACCURACY **FROM** gfg **WHERE** ACCURACY>85;*

UPDATE

This refers to the updating of tuple values already present in the table.

***Syntax: UPDATE** tableName **SET** $Attribute_1 = Value_1$, Attribute2 = Value2 , . . .*
***WHERE** condition;*
The WHERE clause must be included, else all records in the table will be updated.

DELETE

This refers to the deletion of the tuple present in the table.

***SYNTAX: DELETE FROM** tableName **WHERE** condition*

If WHERE clause is not used then all the records will be deleted.

- **PROGRAM:**

```
import sqlite3
from tkinter import *

# Create an SQLite database connection
conn = sqlite3.connect('registration.db')
cursor = conn.cursor()

# Create the users table if it doesn't exist
cursor.execute('''
    CREATE TABLE IF NOT EXISTS users (
        id INTEGER PRIMARY KEY,
        name TEXT,
        email TEXT,
        contact_number TEXT,
        gender TEXT,
        country TEXT,
        password TEXT
    )
''')

def register_user():
    # Get form input values
    name = en1.get()
    email = en2.get()
    contact_number = en3.get()
    gender = var.get()  # Assumes var is an IntVar from radio buttons
```

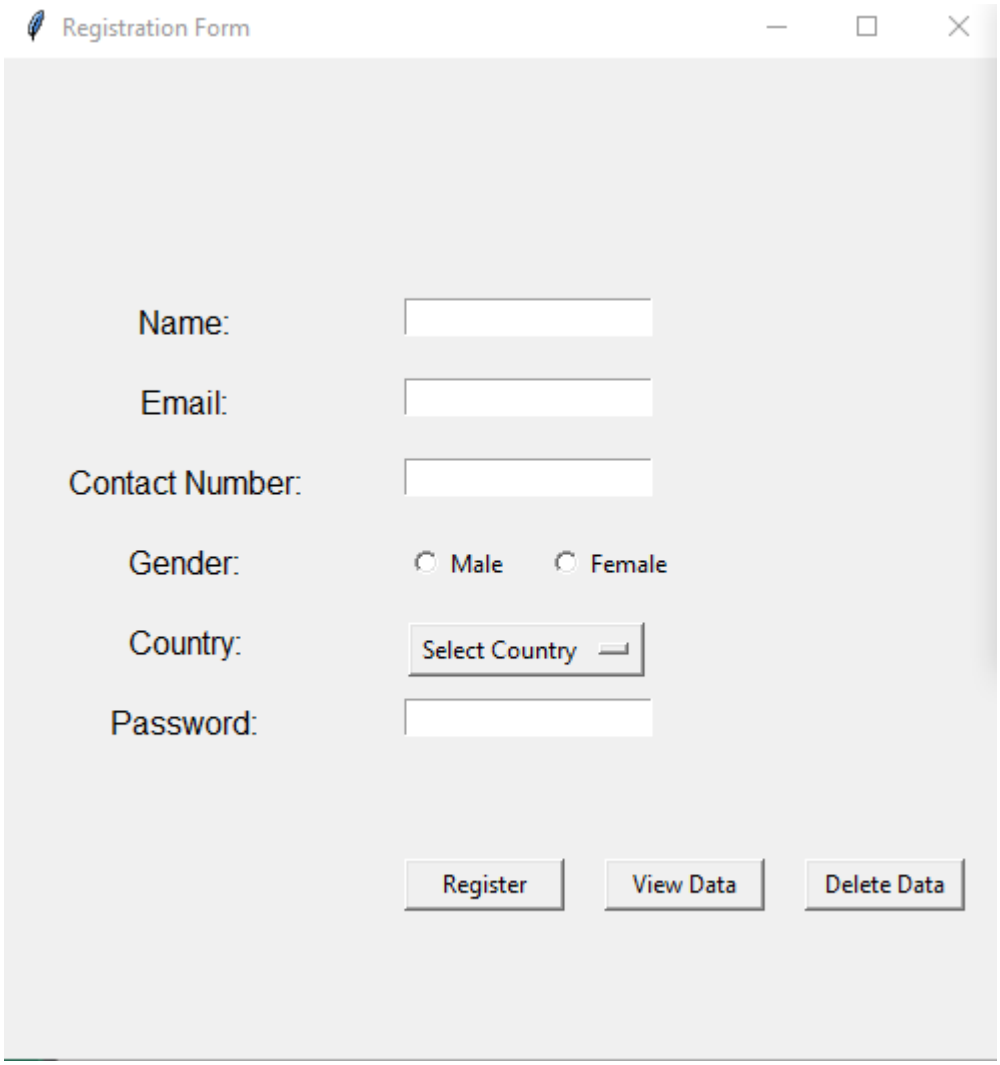
```
    country = var_country.get()  # Assumes var_country is a StringVar from the dropdown
    password = en6.get()

    # Insert user data into the database
    cursor.execute('''
        INSERT INTO users (name, email, contact_number, gender, country, password)
        VALUES (?, ?, ?, ?, ?, ?)
    ''', (name, email, contact_number, gender, country, password))
    conn.commit()
    clear_entries()

def clear_entries():
    en1.delete(0, 'end')
    en2.delete(0, 'end')
    en3.delete(0, 'end')
    en6.delete(0, 'end')
    var.set(0)
    var_country.set("Select Country")

def view_data():
    view_window = Toplevel()
    view_window.title("View Registered Data")
    view_window.geometry("600x300")  # Adjust window size here

    scrollbar = Scrollbar(view_window)
    scrollbar.pack(side=RIGHT, fill=Y)
```

```
    listbox = Listbox(view_window, yscrollcommand=scrollbar.set, width=80)  # Adjust width here
    listbox.pack(side=LEFT, fill=BOTH)

    scrollbar.config(command=listbox.yview)

    cursor.execute("SELECT * FROM users")
    data = cursor.fetchall()
    for row in data:
        listbox.insert(END, row)

def delete_data():
    delete_window = Toplevel()
    delete_window.title("Delete Data")

    Label(delete_window, text="Enter Name to Delete:", width=20, font=("arial", 12)).pack()
    entry_name = Entry(delete_window)
    entry_name.pack()

    def delete():
        name = entry_name.get()
        cursor.execute("DELETE FROM users WHERE name=?", (name,))
        conn.commit()
        delete_window.destroy()

    Button(delete_window, text="Delete", command=delete).pack()
```

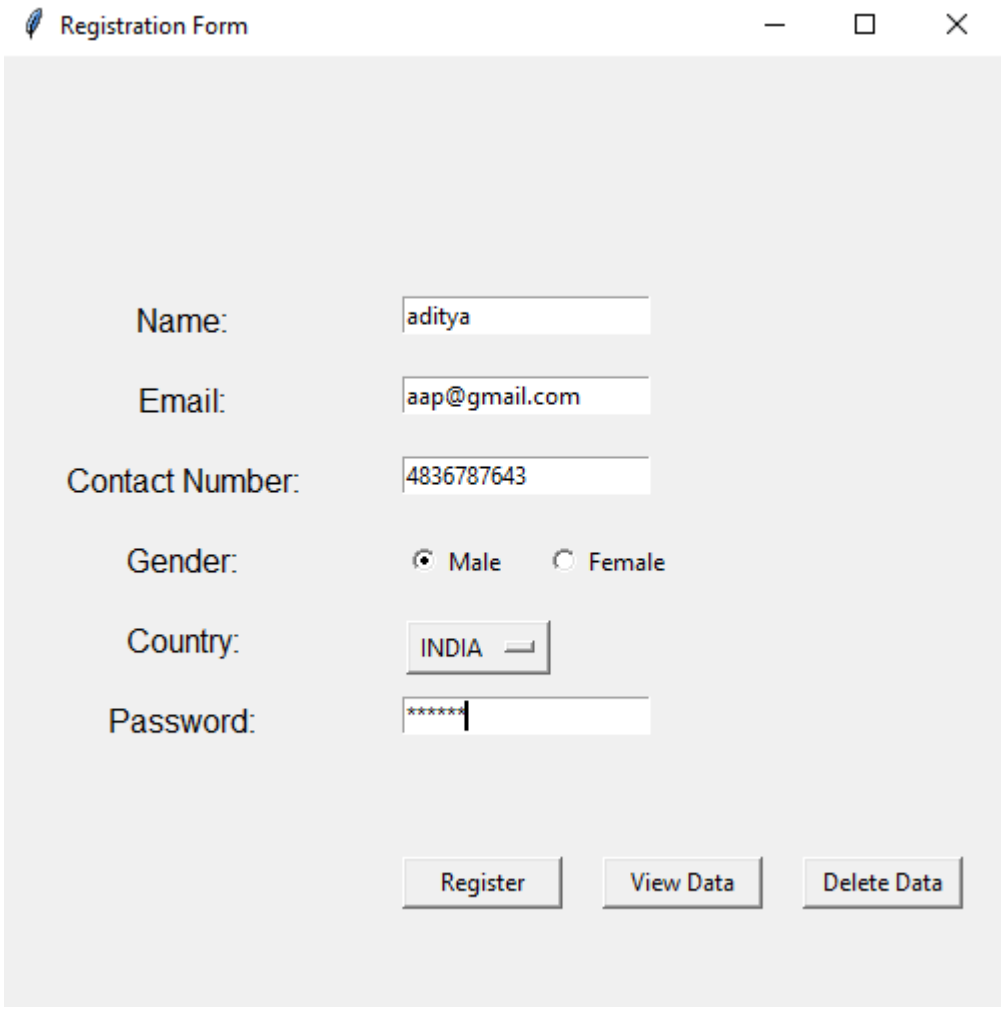
```
# Create the registration form
base = Tk()
base.geometry("500x500")
base.title("Registration Form")

Label(base, text="Name:", width=15, font=("arial", 12)).place(x=20, y=120)
en1 = Entry(base)
en1.place(x=200, y=120)

Label(base, text="Email:", width=15, font=("arial", 12)).place(x=20, y=160)
en2 = Entry(base)
en2.place(x=200, y=160)

Label(base, text="Contact Number:", width=15, font=("arial", 12)).place(x=20,
y=200)
en3 = Entry(base)
en3.place(x=200, y=200)

Label(base, text="Gender:", width=15, font=("arial", 12)).place(x=20, y=240)
var = IntVar()
Radiobutton(base, text="Male", variable=var, value=1).place(x=200, y=240)
Radiobutton(base, text="Female", variable=var, value=2).place(x=270, y=240)

Label(base, text="Country:", width=15, font=("arial", 12)).place(x=20, y=280)
var_country = StringVar(base)
var_country.set("Select Country")
```

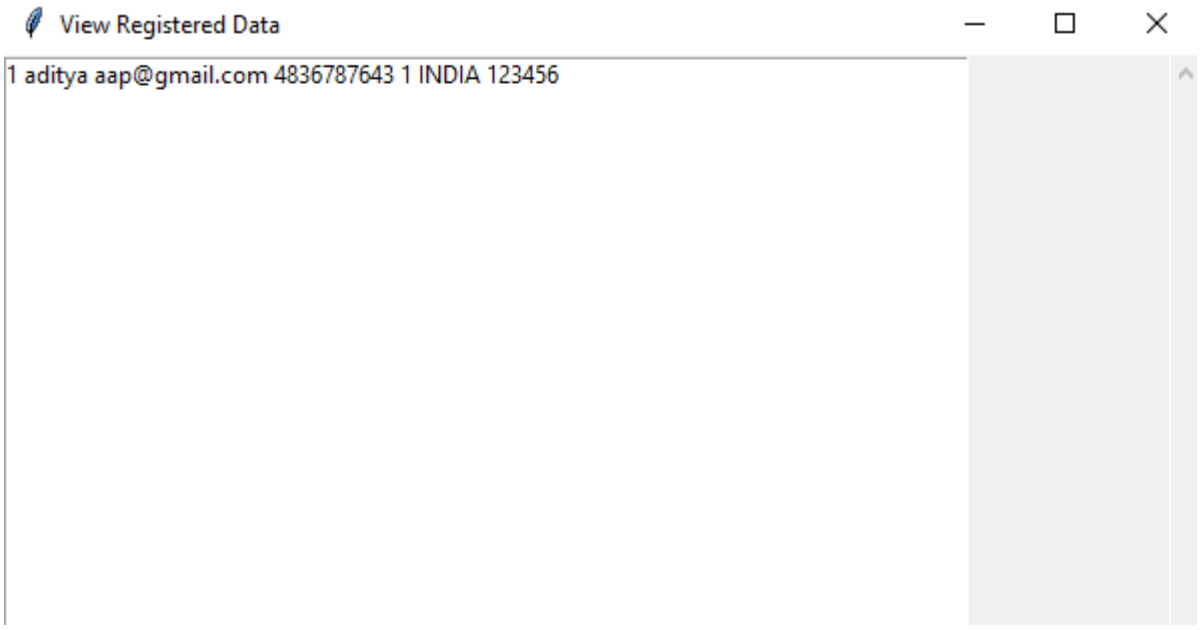
```
dropdown = OptionMenu(base, var_country, "INDIA", "Canada", "UK",
"Australia")

dropdown.place(x=200, y=280)


Label(base, text="Password:", width=15, font=("arial", 12)).place(x=20, y=320)

en6 = Entry(base, show='*')

en6.place(x=200, y=320)


Button(base, text="Register", width=10, command=register_user).place(x=200,
y=400)

Button(base, text="View Data", width=10, command=view_data).place(x=300,
y=400)

Button(base, text="Delete Data", width=10,
command=delete_data).place(x=400, y=400)


base.mainloop()


# Close the cursor and connection

cursor.close()

conn.close()
```

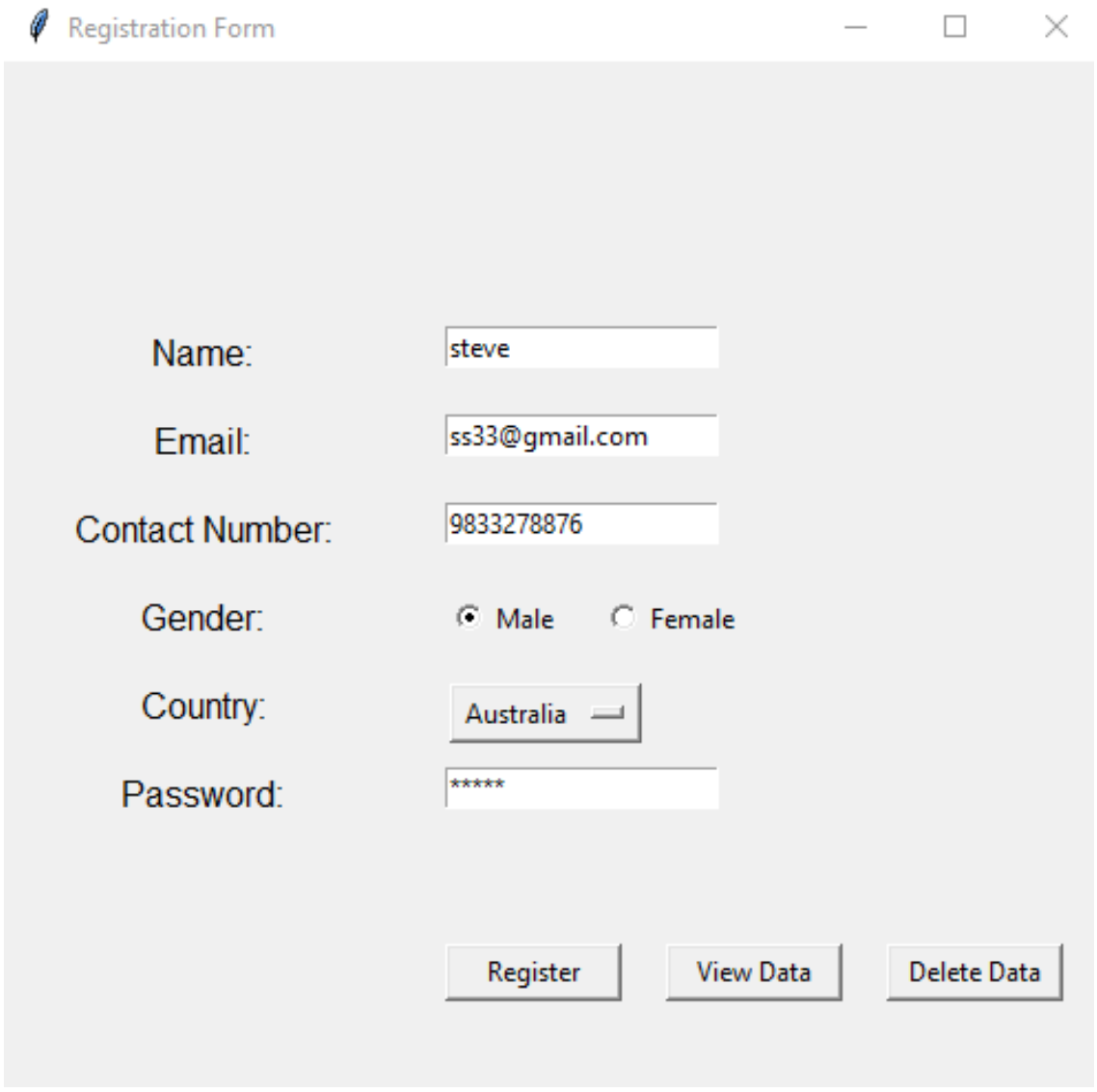
- **OUTPUT:**


Registration Form
Name:
Email:
Contact Number:
Gender:
Male
Female
Country:
Select Country
Password:
Register
View Data
Delete Data

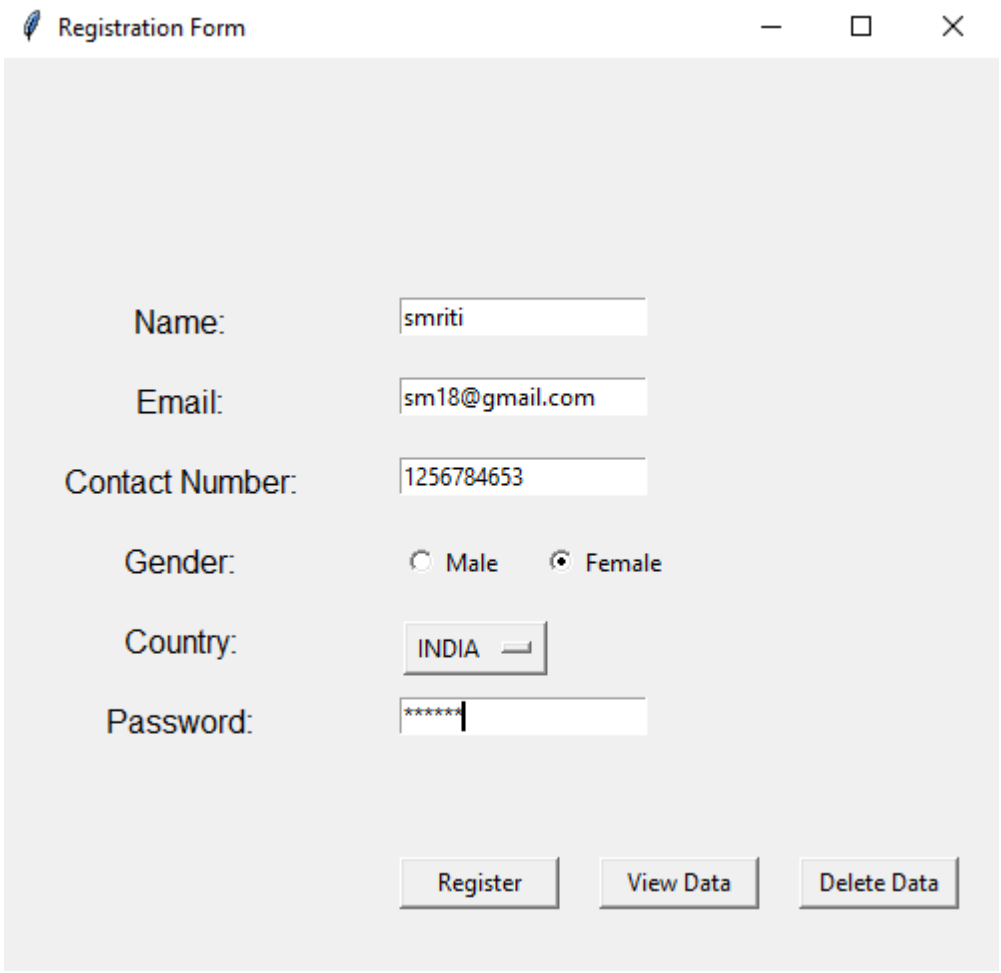

View Registered Data

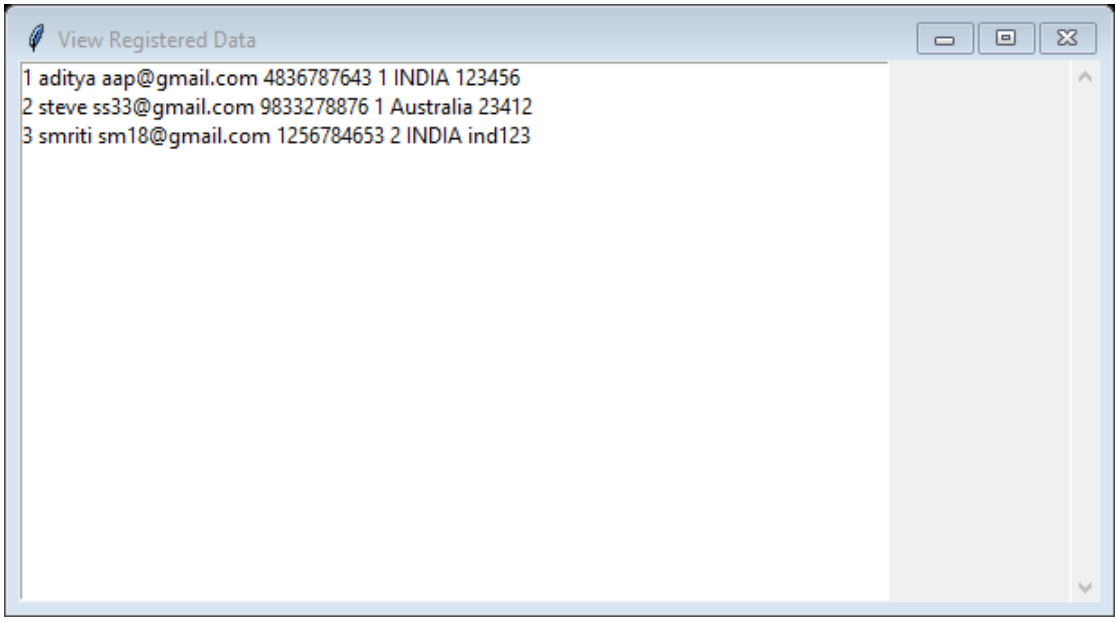

Registration Form
Name:
aditya
Email:
aap@gmail.com
Contact Number:
4836787643
Gender:
Male
Female
Country:
INDIA
Password:
******
Register
View Data
Delete Data

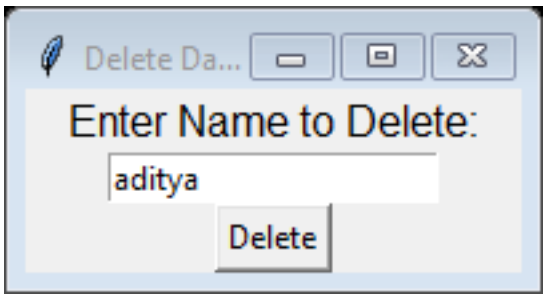

View Registered Data
1 aditya aap@gmail.com 4836787643 1 INDIA 123456

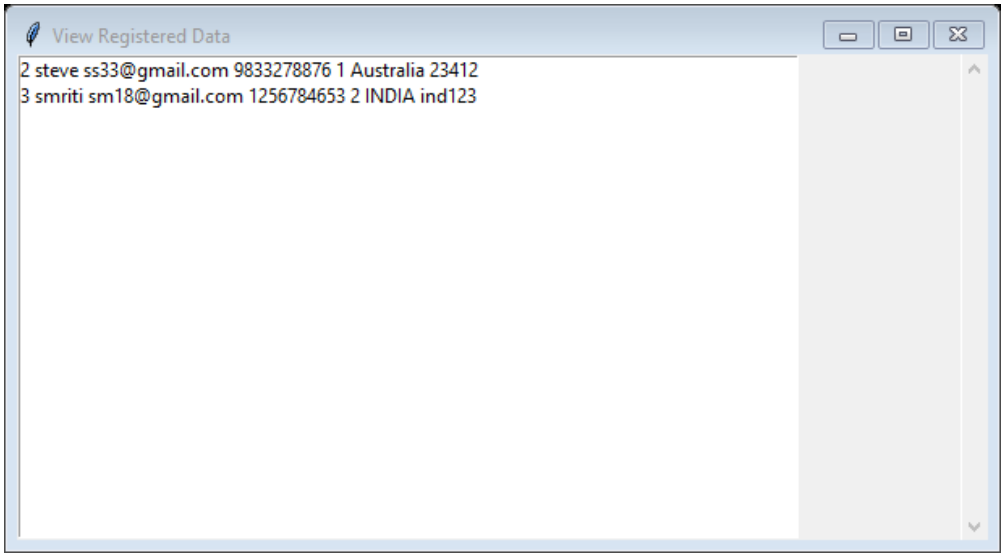

Registration Form
Name:
steve
Email:
ss33@gmail.com
Contact Number:
9833278876
Gender:
Male
Female
Country:
Australia
Password:
*****
Register
View Data
Delete Data

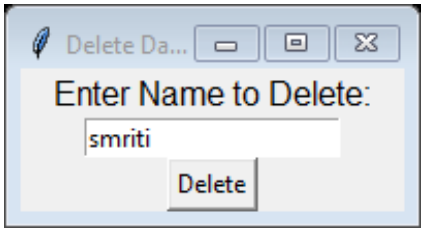

View Registered Data
1 aditya aap@gmail.com 4836787643 1 INDIA 123456
2 steve ss33@gmail.com 9833278876 1 Australia 23412

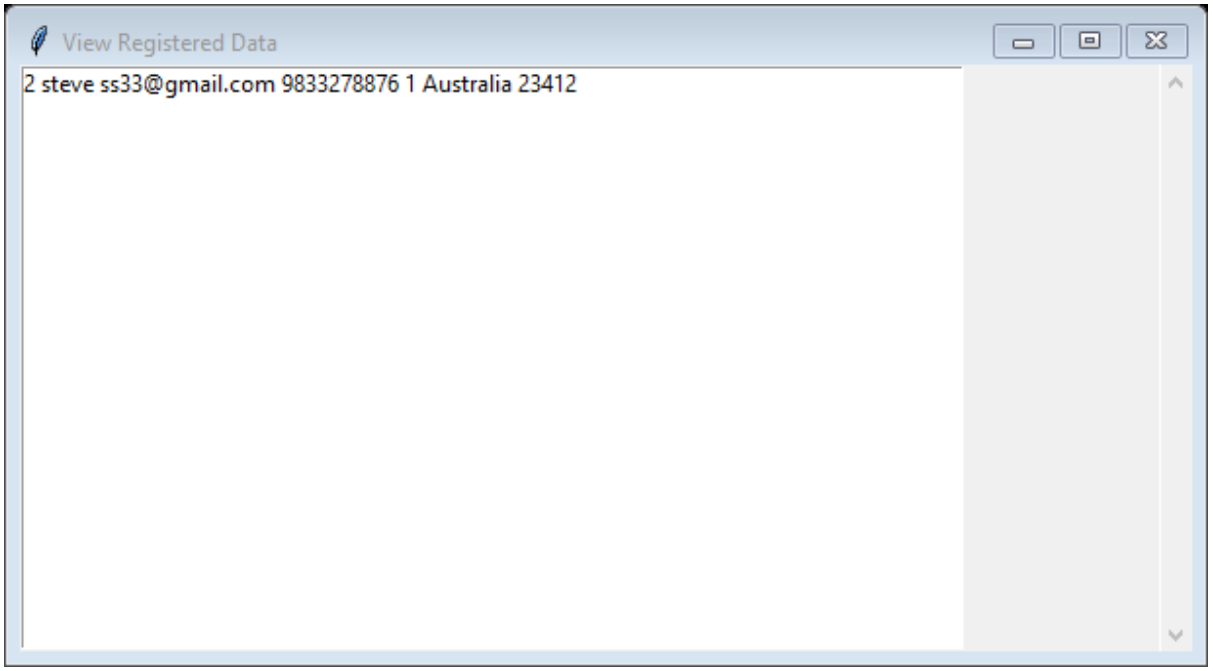

Registration Form
Name:
smriti
Email:
sm18@gmail.com
Contact Number:
1256784653
Gender:
Male
Female
Country:
INDIA
Password:
Register
View Data
Delete Data

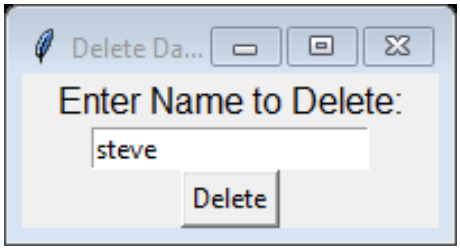

View Registered Data
1 aditya aap@gmail.com 4836787643 1 INDIA 123456
2 steve ss33@gmail.com 9833278876 1 Australia 23412
3 smriti sm18@gmail.com 1256784653 2 INDIA ind123

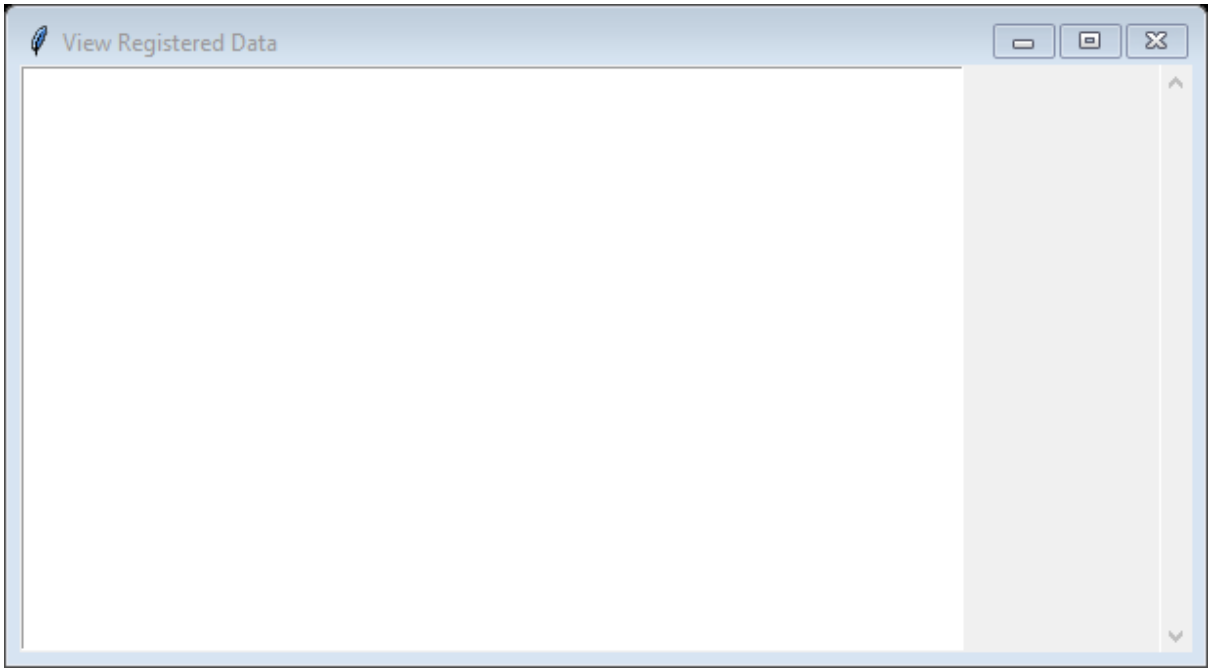

Delete Da...
Enter Name to Delete:
aditya
Delete


View Registered Data
2 steve ss33@gmail.com 9833278876 1 Australia 23412
3 smriti sm18@gmail.com 1256784653 2 INDIA ind123


Delete Da...
Enter Name to Delete:
smriti
Delete


View Registered Data
2 steve ss33@gmail.com 9833278876 1 Australia 23412





- ***CONCLUSION:*** **Hence, we have successfully implemented program on demonstrating CRUD operations on Registration form using SQLite python; LO 1.**

NAME: **Meet Raut**

DIV: **S2-1**

ROLL NO: **2201084**

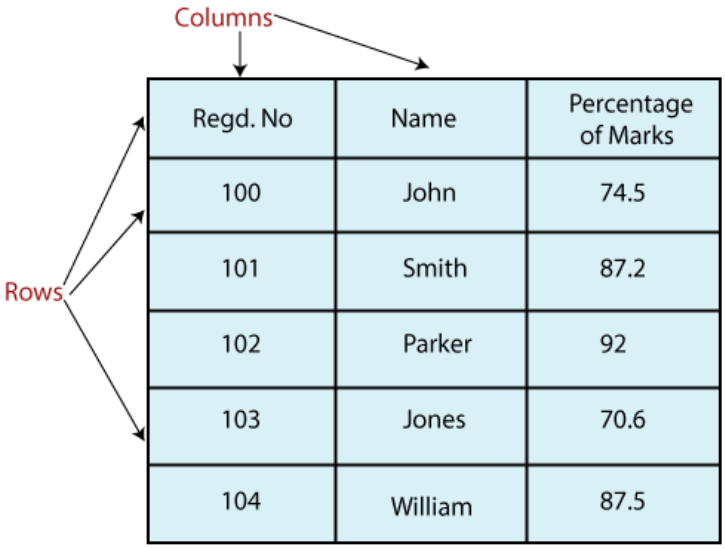
EXPERIMENT – 13:

- ***AIM:*** **To study and implement programs to demonstrate Data Series and Data Frames using Pandas.**

- ***THEORY:***

Pandas DataFrame is a widely used data structure which works with a two-dimensional array with labeled axes (rows and columns). DataFrame is defined as a standard way to store data that has two different indexes, i.e., **row index** and **column index**. It consists of the following properties:

- The columns can be heterogeneous types like int, bool, and so on.
- It can be seen as a dictionary of Series structure where both the rows and columns are indexed. It is denoted as "columns" in case of columns and "index" in case of rows.
- **data:** It consists of different forms like ndarray, series, map, constants, lists, array.
- **index:** The Default np.arrange(n) index is used for the row labels if no index is passed.
- **columns:** The default syntax is np.arrange(n) for the column labels. It shows only true if no index is passed.
- **dtype:** It refers to the data type of each column.
- **copy():** It is used for copying the data



Regd. No	Name	Percentage of Marks
100	John	74.5
101	Smith	87.2
102	Parker	92
103	Jones	70.6
104	William	87.5

Create a DataFrame:

We can create a DataFrame using following ways:

- dict
- Lists
- Numpy ndarrrays
- Series

Column Selection:

We can select any column from the DataFrame.

Column Addition:

We can also add any new column to an existing DataFrame.

Column Deletion:

We can also delete any column from the existing DataFrame.

Row Selection:

We can select any row by passing the row label to a **loc** function. The rows can also be selected by passing the integer location to an **iloc** function. For selecting a row, we have passed the integer location to an **iloc** function. It is another method to select multiple rows using '**:**' operator.

Addition of rows:

We can easily add new rows to the DataFrame using **append** function. It adds the new rows at the end.

Deletion of rows:

We can delete or drop any rows from a DataFrame using the **index** label. If in case, the label is duplicate then multiple rows will be deleted.

DataFrame Functions

There are lots of functions used in DataFrame which are as follows:

Functions	Description
Pandas DataFrame.append()	Add the rows of other dataframe to the end of the given dataframe.
Pandas DataFrame.apply()	Allows the user to pass a function and apply it to every single value of the Pandas series.
Pandas DataFrame.assign()	Add new column into a dataframe.
Pandas DataFrame.astype()	Cast the Pandas object to a specified dtype.astype() function.
Pandas DataFrame.concat()	Perform concatenation operation along an axis in the DataFrame.
Pandas DataFrame.count()	Count the number of non-NA cells for each column or row.
Pandas DataFrame.describe()	Calculate some statistical data like percentile, mean and std of the numerical values of the Series or DataFrame.
Pandas DataFrame.drop_duplicates()	Remove duplicate values from the DataFrame.
Pandas DataFrame.groupby()	Split the data into various groups.
Pandas DataFrame.head()	Returns the first n rows for the object based on position.
Pandas DataFrame.hist()	Divide the values within a numerical variable into "bins".

Pandas DataFrame.iterrows()	Iterate over the rows as (index, series) pairs.
Pandas DataFrame.mean()	Return the mean of the values for the requested axis.
Pandas DataFrame.melt()	Unpivots the DataFrame from a wide format to a long format.
Pandas DataFrame.merge()	Merge the two datasets together into one.
Pandas DataFrame.pivot_table()	Aggregate data with calculations such as Sum, Count, Average, Max, and Min.
Pandas DataFrame.query()	Filter the dataframe.
Pandas DataFrame.sample()	Select the rows and columns from the dataframe randomly.
Pandas DataFrame.shift()	Shift column or subtract the column value with the previous row value from the dataframe.
Pandas DataFrame.sort()	Sort the dataframe.
Pandas DataFrame.sum()	Return the sum of the values for the requested axis by the user.
Pandas DataFrame.to_excel()	Export the dataframe to the excel file.
Pandas DataFrame.transpose()	Transpose the index and columns of the dataframe.
Pandas DataFrame.where()	Check the dataframe for one or more conditions.

- **OUTPUT:**

```
import pandas as pd
data=pd.read_csv("nbaNew.csv")
data.tail(5)
```

[19] ✓ 0.1s Python

	#	SeasonStart	PlayerName	PlayerSalary	Pos	Age	Tm	G	GS	MP	...	FT%	ORB	DRB	TRB	AST
24625	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
24626	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
24627	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
24628	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
24629	NaN	NaN	https://www.kaggle.com/drgilermo/nba-players-s...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN

5 rows × 54 columns

```
data.columns
```

[2] ✓ 0.0s Python

```
Index(['#', 'SeasonStart', 'PlayerName', 'PlayerSalary ', 'Pos', 'Age', 'Tm',
       'G', 'GS', 'MP', 'PER', 'TS%', '3PAr', 'FTr', 'ORB%', 'DRB%', 'TRB%',
       'AST%', 'STL%', 'BLK%', 'TOV%', 'USG%', 'blanl', 'OWS', 'DWS', 'WS',
       'WS/48', 'blank2', 'OBPM', 'DBPM', 'BPM', 'VORP', 'FG', 'FGA', 'FG%',
       '3P', '3PA', '3P%', '2P', '2PA', '2P%', 'eFG%', 'FT', 'FTA', 'FT%',
       'ORB', 'DRB', 'TRB', 'AST', 'STL', 'BLK', 'TOV', 'PF', 'PTS'],
      dtype='object')
```

```
import pandas as pd
data=pd.read_csv("nbaNew.csv")
data.head(5)
```

[9] ✓ 0.1s Python

	#	SeasonStart	PlayerName	PlayerSalary	Pos	Age	Tm	G	GS	MP	...	FT%	ORB	DRB	TRB	AST	STL	BLK	TOV	
0	8035.0	1986.0	A.C. Green	NaN	PF	22.0	LAL	82.0	1.0	1542.0	...	61.10%	160.0	221.0	381.0	54.0	49.0	49.0	99.0	2
1	8420.0	1987.0	A.C. Green	NaN	PF	23.0	LAL	79.0	72.0	2240.0	...	78.00%	210.0	405.0	615.0	84.0	70.0	80.0	102.0	1
2	8807.0	1988.0	A.C. Green	NaN	PF	24.0	LAL	82.0	64.0	2636.0	...	77.30%	245.0	465.0	710.0	93.0	87.0	45.0	120.0	2
3	9242.0	1989.0	A.C. Green	NaN	PF	25.0	LAL	82.0	82.0	2510.0	...	78.60%	258.0	481.0	739.0	103.0	94.0	55.0	119.0	1
4	9688.0	1990.0	A.C. Green	$1,750,000.00	PF	26.0	LAL	82.0	82.0	2709.0	...	75.10%	262.0	450.0	712.0	90.0	66.0	50.0	116.0	2

5 rows × 54 columns

```
data[["Age","Pos"]]
```

[15] ✓ 0.0s — Python

	Age	Pos
0	22.0	PF
1	23.0	PF
2	24.0	PF
3	25.0	PF
4	26.0	PF
...	...	...
24625	NaN	NaN
24626	NaN	NaN
24627	NaN	NaN
24628	NaN	NaN
24629	NaN	NaN

24630 rows × 2 columns

```
data.sum()
```

[11] ✓ 0.2s — Python

```
C:\Users\Lenovo\AppData\Local\Temp\ipykernel_14284\1263598667.py:1: FutureWarning: Dropping of nuisance columns in DataFrame reductions (wi
  data.sum()
```

```
#             3.042072e+08
SeasonStart   4.906566e+07
Age           6.563710e+05
G             1.251813e+06
GS            4.301780e+05
MP            2.920023e+07
PER           3.007581e+05
3PAr          2.987947e+03
ORB%          1.285271e+05
DRB%          2.850304e+05
TRB%          2.146144e+05
AST%          2.934397e+05
STL%          3.427080e+04
BLK%          2.932970e+04
TOV%          2.953964e+05
USG%          3.713235e+05
blanl         0.000000e+00
OWS           3.091090e+04
DWS           3.017550e+04
WS            6.111330e+04
WS/48         1.566613e+03
blank2        0.000000e+00
OBPM         -3.698510e+04
DBPM         -1.140860e+04
BPM          -4.838880e+04
...
BLK           5.089080e+05
TOV           1.452548e+06
PF            2.864737e+06
PTS           1.256110e+07
```

```
data.info()
```

[3] ✓ 0.0s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 24630 entries, 0 to 24629
Data columns (total 54 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   #              24624 non-null  float64
 1   SeasonStart    24624 non-null  float64
 2   PlayerName     24625 non-null  object
 3   PlayerSalary   10978 non-null  object
 4   Pos            24624 non-null  object
 5   Age            24616 non-null  float64
 6   Tm             24624 non-null  object
 7   G              24624 non-null  float64
 8   GS             18233 non-null  float64
 9   MP             24138 non-null  float64
 10  PER            24101 non-null  float64
 11  TS%            24538 non-null  object
 12  3PAr           18839 non-null  float64
 13  FTr            24525 non-null  object
 14  ORB%           20792 non-null  float64
 15  DRB%           20792 non-null  float64
 16  TRB%           21571 non-null  float64
 17  AST%           22555 non-null  float64
 18  STL%           20792 non-null  float64
 19  BLK%           20792 non-null  float64
...
 52  PF             24624 non-null  float64
 53  PTS            24624 non-null  float64
dtypes: float64(44), object(10)
memory usage: 10.1+ MB
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```
data.isnull()
```

[4] ✓ 0.0s Python

	#	SeasonStart	PlayerName	PlayerSalary	Pos	Age	Tm	G	GS	MP	...	FT%	ORB	DRB	TRB	AST	STL	BLK	TOV
0	False	False	False	True	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False
1	False	False	False	True	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False
2	False	False	False	True	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False
3	False	False	False	True	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
24625	True	True	True	True	True	True	True	True	True	True	...	True	True	True	True	True	True	True	True
24626	True	True	True	True	True	True	True	True	True	True	...	True	True	True	True	True	True	True	True
24627	True	True	True	True	True	True	True	True	True	True	...	True	True	True	True	True	True	True	True
24628	True	True	True	True	True	True	True	True	True	True	...	True	True	True	True	True	True	True	True
24629	True	True	False	True	True	True	True	True	True	True	...	True	True	True	True	True	True	True	True

24630 rows × 54 columns

```
data.dropna()
data
```

[20] ✓ 0.0s Python

	#	SeasonStart	PlayerName	PlayerSalary	Pos	Age	Tm	G	GS	MP	...	FT%	ORB	DRB	TR
0	8035.0	1986.0	A.C. Green	NaN	PF	22.0	LAL	82.0	1.0	1542.0	...	61.10%	160.0	221.0	381
1	8420.0	1987.0	A.C. Green	NaN	PF	23.0	LAL	79.0	72.0	2240.0	...	78.00%	210.0	405.0	615
2	8807.0	1988.0	A.C. Green	NaN	PF	24.0	LAL	82.0	64.0	2636.0	...	77.30%	245.0	465.0	710
3	9242.0	1989.0	A.C. Green	NaN	PF	25.0	LAL	82.0	82.0	2510.0	...	78.60%	258.0	481.0	739
4	9688.0	1990.0	A.C. Green	$1,750,000.00	PF	26.0	LAL	82.0	82.0	2709.0	...	75.10%	262.0	450.0	712
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
24625	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	Na
24626	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	Na
24627	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	Na
24628	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	Na
24629	NaN	NaN	https://www.kaggle.com/drgilermo/nba-players-s...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	Na

24630 rows × 54 columns

```
data.isna()
```

[17] ✓ 0.0s Python

	#	SeasonStart	PlayerName	PlayerSalary	Pos	Age	Tm	G	GS	MP	...	FT%	ORB	DRB	TRB	AST	STL	BLK	TOV
0	False	False	False	True	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False
1	False	False	False	True	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False
2	False	False	False	True	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False
3	False	False	False	True	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
24625	True	True	True	True	True	True	True	True	True	True	...	True	True	True	True	True	True	True	True
24626	True	True	True	True	True	True	True	True	True	True	...	True	True	True	True	True	True	True	True
24627	True	True	True	True	True	True	True	True	True	True	...	True	True	True	True	True	True	True	True
24628	True	True	True	True	True	True	True	True	True	True	...	True	True	True	True	True	True	True	True
24629	True	True	False	True	True	True	True	True	True	True	...	True	True	True	True	True	True	True	True

24630 rows × 54 columns

```
data.sort_values("Age")
```

[21] ✓ 0.0s Python

	#	SeasonStart	PlayerName	PlayerSalary	Pos	Age	Tm	G	GS	MP	...	FT%	ORB	DRB	TR
5467	16836.0	2004.0	Darko Milicic	$3,865,440.00	C	18.0	DET	34.0	0.0	159.0	...	58.30%	11.0	32.0	43
271	14148.0	1999.0	Al Harrington	NaN	PF	18.0	IND	21.0	0.0	160.0	...	60.00%	20.0	19.0	39
14322	12900.0	1997.0	Kobe Bryant	$1,167,240.00	SG	18.0	LAL	71.0	6.0	1103.0	...	81.90%	47.0	85.0	132
923	17127.0	2005.0	Andris Biedrins	$1,857,360.00	C	18.0	GSW	30.0	1.0	384.0	...	47.50%	47.0	71.0	118
11447	13219.0	1997.0	Jermaine O'Neal	NaN	PF	18.0	POR	45.0	0.0	458.0	...	60.30%	39.0	85.0	124
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
24625	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	Na
24626	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	Na
24627	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	Na
24628	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	Na
24629	NaN	NaN	https://www.kaggle.com/drgilermo/nba-players-s...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	Na

24630 rows × 54 columns

```
data.insert(loc=2, column='Association', value="NBA")
data.insert(loc=3, column='PlayerStatus', value="Avail")
```

[25] ✓ 0.0s Python

```
data
```

[26] ✓ 0.0s Python

	#	SeasonStart	Association	PlayerStatus	PlayerName	PlayerSalary	Pos	Age	Tm	G	...	FT%	ORB
0	8035.0	1986.0	NBA	Avail	A.C. Green	NaN	PF	22.0	LAL	82.0	...	61.10%	160.0
1	8420.0	1987.0	NBA	Avail	A.C. Green	NaN	PF	23.0	LAL	79.0	...	78.00%	210.0
2	8807.0	1988.0	NBA	Avail	A.C. Green	NaN	PF	24.0	LAL	82.0	...	77.30%	245.0
3	9242.0	1989.0	NBA	Avail	A.C. Green	NaN	PF	25.0	LAL	82.0	...	78.60%	258.0
4	9688.0	1990.0	NBA	Avail	A.C. Green	$1,750,000.00	PF	26.0	LAL	82.0	...	75.10%	262.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
24625	NaN	NaN	NBA	Avail	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
24626	NaN	NaN	NBA	Avail	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
24627	NaN	NaN	NBA	Avail	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
24628	NaN	NaN	NBA	Avail	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
24629	NaN	NaN	NBA	Avail	https://www.kaggle.com/drgilermo/nba-players-s...	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN

24630 rows × 56 columns

```
data['Age'].sum()
```

[27] ✓ 0.0s Python

```
656371.0
```

+ Code + Markdown

```
data['Age'].min()
```

[29] ✓ 0.0s Python

```
18.0
```

```
data['Age'].max()
```

[30] ✓ 0.0s Python

```
44.0
```

```
data.loc[2]
```

[31] ✓ 0.0s

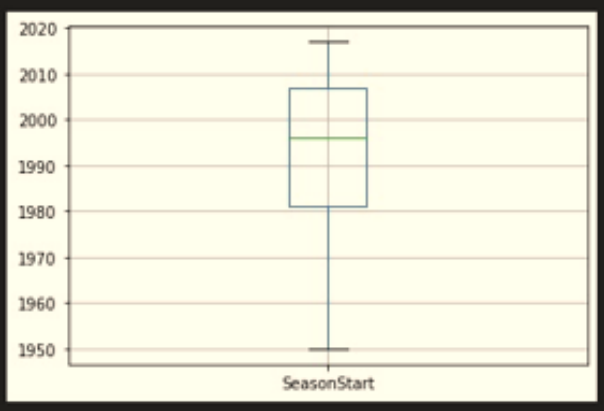
```
#                   8807.0
SeasonStart         1988.0
Association            NBA
PlayerStatus         Avail
PlayerName      A.C. Green
PlayerSalary           NaN
Pos                     PF
Age                   24.0
Tm                     LAL
G                     82.0
GS                    64.0
MP                  2636.0
PER                   14.5
TS%                 58.10%
3PAr                 0.003
FTr                 59.20%
ORB%                  11.1
DRB%                  19.1
TRB%                  15.3
AST%                   4.5
STL%                   1.6
BLK%                   1.0
TOV%                  12.9
USG%                  14.7
blanl                  NaN
...
BLK                   45.0
TOV                  120.0
PF                   204.0
PTS                  937.0
Name: 2, dtype: object
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```
data[['SeasonStart']].boxplot()
```

[35] ✓ 3.4s

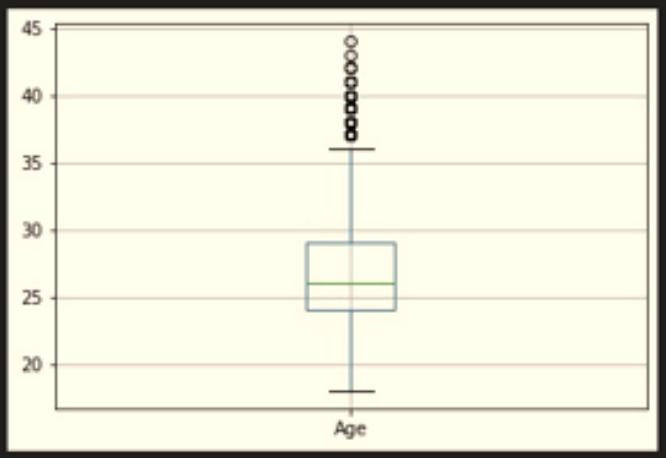
```
<AxesSubplot:>
```



```
data[['Age']].boxplot()
```

[36] ✓ 0.1s

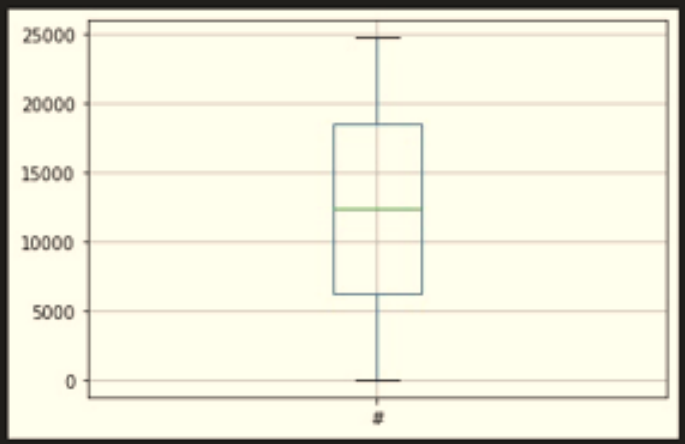
```
<AxesSubplot:>
```




```
data[['#']].boxplot()
```

[38] ✓ 0.0s

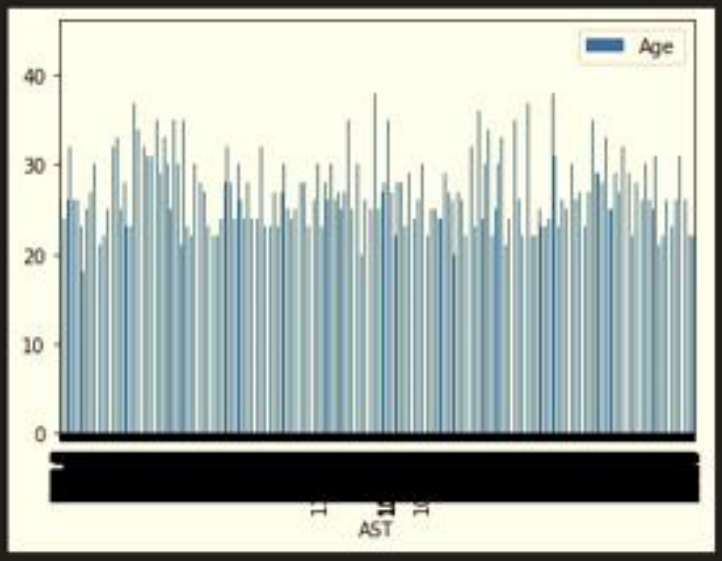
```
<AxesSubplot:>
```



```
import matplotlib.pyplot as plt

data.plot(x='AST', y='Age', kind="bar")
```

[39] ✓ 2m 51.9s

```
<AxesSubplot:xlabel='AST'>
```



- ***CONCLUSION:*** **Hence, we have successfully implemented program on demonstrating Data Series and Data Frames using Pandas; LO 1.**

NAME: **Meet Raut**

DIV: **S2-1**

ROLL NO: **2201084**

EXPERIMENT – 14:

- ***AIM:*** **To study and implement Menu Driven program for data structure using built in function for Link List, Stack and Queue.**

- ***THEORY:***

Data structure organizes the storage in computers so that we can easily access and change data. Stacks and Queues are the earliest data structure defined in computer science. A simple Python list can act as a queue and stack as well. A queue follows FIFO rule (First In First Out) and used in programming for sorting. It is common for stacks and queues to be implemented with an array or linked list.

Python offers various ways to implement the stack. Such as

- **List**
- **Collection.deque**
- **LifeQueue**

Here, **Python List** can be used to implement the stack. For those following, in-build methods are used in the below program.

- **append() method -** To PUSH or insert elements into the stack.
- **pop() method -** To POP out or remove the element from the stack in LIFO order.
- **len(stack) -** To find out the size of the stack.

LINK LIST:

In Python, a linked list can be implemented using a class, with each instance of the class representing a node in the list. Each node has a reference (pointer) to the next node in the list, as well as a value. The last node in the list has a reference to None, indicating the end of the list. Linked lists have advantages over arrays when it comes to inserting and deleting elements, as the elements do not have to be shifted in memory. However, they have slower random-access times.

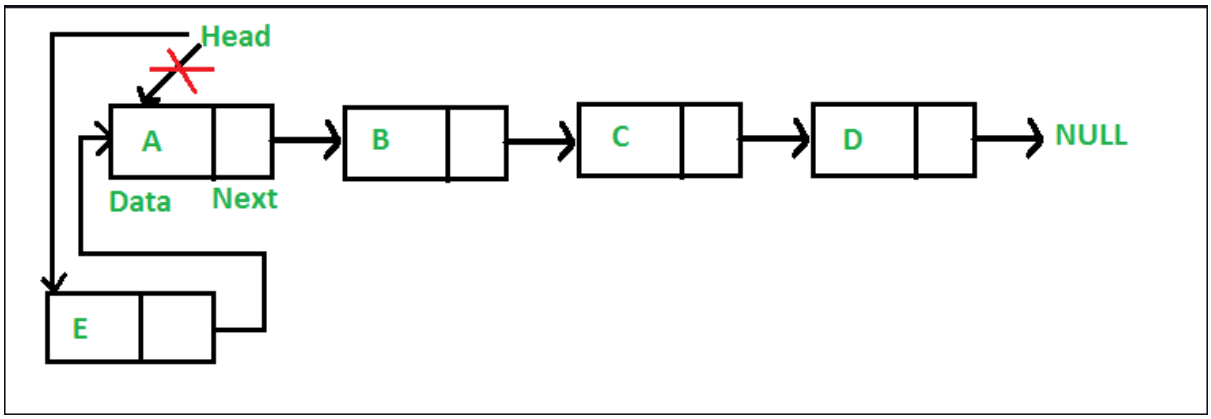
Nodes

These nodes generally have two attributes:

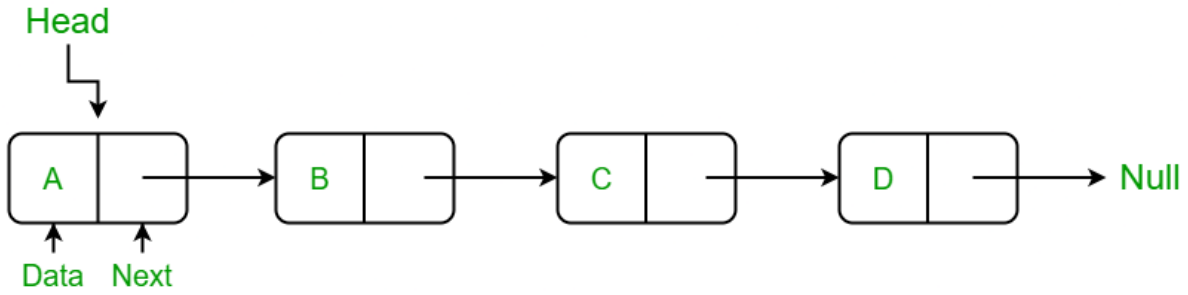
1. One that holds data. This data can be a character, string, stack, queue, another object, or another linked list. It can be anything.
2. A reference to another node, the "next" node, often in the form of the actual address in memory.
3. In doubly linked list, they also have a reference to the "previous" node.

The linked list

These nodes make up a Linked List which can be added to, or removed from



A linked list is a type of linear data structure similar to arrays. It is a collection of nodes that are linked with each other. A node contains two things first is data and second is a link that connects it with another node. Below is an example of a linked list with four nodes and each node contains character data and a link to another node. Our first node is where **head** points and we can access all the elements of the linked list using the **head.**



Creating a linked list in Python

In this LinkedList class, we will use the Node class to create a linked list. In this class, we have an **__init__** method that initializes the linked list with an empty head. Next, we have created an **insertAtBegin()** method to insert a node at the beginning of the linked list, an **insertAtIndex()** method to insert a node at the given index of the linked list,

and **insertAtEnd()** method inserts a node at the end of the linked list. After that, we have the **remove_node()** method which takes the data as an argument to delete that node. In the **remove_node()** method we traverse the linked list if a node is present equal to data then we delete that node from the linked list. Then we have the **sizeOfLL()** method to get the current size of the linked list and the last method of the LinkedList class is **printLL()** which traverses the linked list and prints the data of each node.

Creating a Node Class

We have created a Node class in which we have defined a **__init__** function to initialize the node with the data passed as an argument and a reference with None because if we have only one node then there is nothing in its reference.

Insertion in Linked List:

Insertion at Beginning in Linked List

This method inserts the node at the beginning of the linked list. In this method, we create a **new_node** with the given **data** and check if the head is an empty node or not if the head is empty then we make the **new_node** as **head** and **return** else we insert the head at the next **new_node** and make the **head** equal to **new_node.**

Insert a Node at a Specific Position in a Linked List

This method inserts the node at the given index in the linked list. In this method, we create a **new_node** with given data , a current_node that equals to the head, and a counter **'position'** initializes with **0.** Now, if the index is equal to zero it means the node is to be inserted at begin so we called **insertAtBegin() method** else we run a while loop until the **current_node** is not equal to **None** or **(position+1)** is not equal to the index we have to at the one position back to insert at a given position to make the linking of nodes and in each iteration, we increment the position by 1 and make the **current_node** next of it. When the loop breaks and if **current_node** is not equal to **None** we insert new_node at after to the **current_node.** If **current_node** is equal to **None** it means that the index is not present in the list and we print **"Index not present".**

Insertion in Linked List at End

This method inserts the node at the end of the linked list. In this method, we create a **new_node** with the given data and check if the **head** is an empty node or not if the **head** is empty then we make the **new_node** as **head** and return **else** we make a **current_node equal** to **the head** traverse to the last **node** of the linked list and when we get **None** after the current_node the while loop breaks and insert the **new_node** in the next of **current_node** which is the last node of linked list.

Delete Node in a Linked List:

Remove First Node from Linked List

This method removes the first node of the linked list simply by making the second node **head** of the linked list.

Remove Last Node from Linked List

In this method, we will delete the last node. First, we traverse to the second last node using the while loop, and then we make the next of that node **None** and last node will be removed.

Delete a Linked List Node at a given Position

In this method, we will remove the node at the given index, this method is similar to the **insert_at_inded()** method. In this method, if the **head** is **None** we simply **return** else we initialize a **current_node** with **self.head** and **position** with **0.** If the position is equal to the index we called the **remove_first_node()** method else we traverse to the one node before that we want to remove using the while loop. After that when we out of the while loop we check **that current_node** is equal to **None** if not then we make the next of current_node equal to the next of node that we want to remove else we print the message **"Index not present"** because **current_node** is equal to **None.**

Delete a Linked List Node of a given Data

This method removes the node with the given data from the linked list. In this method, firstly we made a **current_node** equal to the **head** and run a **while loop** to traverse the linked list. This while loop breaks when **current_node** becomes **None** or the data next to the current node is equal to the data given in the argument. Now, After coming out of the loop if the **current_node** is equal to **None** it means that the node is not present in the data and we just return, and if the data next to the **current_node** is equal to the data given then we remove that node by making next of that removed_node to the next of current_node. And this is implemented using the if else condition.

Linked List Traversal in Python:

This method traverses the linked list and prints the data of each node. In this method, we made a **current_node** equal to the **head** and iterate through the linked list using a **while loop** until the **current_node** become None and print the data of **current_node** in each iteration and make the **current_node** next to it.

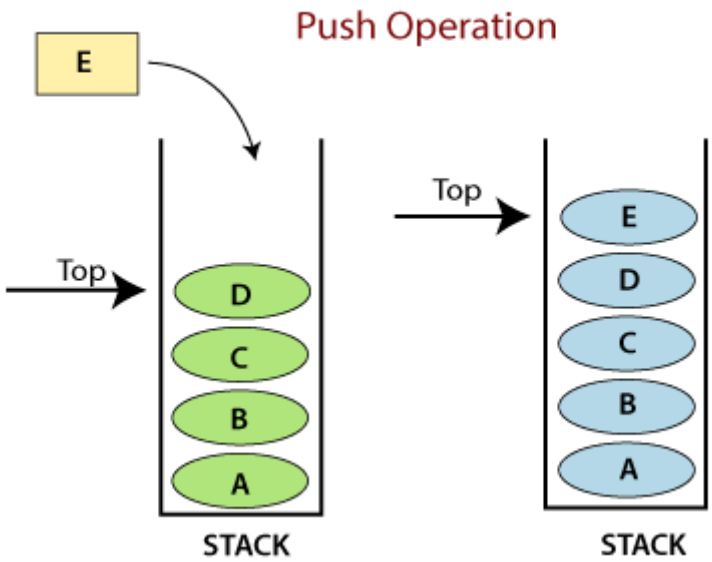
Get Length of a Linked List in Python:

This method returns the size of the linked list. In this method, we have initialized a counter **'size'** with 0, and then if the head is not equal to None we traverse the linked list using a **while loop** and increment the **size** with 1 in each iteration and return the size when **current_node** becomes **None else** we return 0.

Stack:

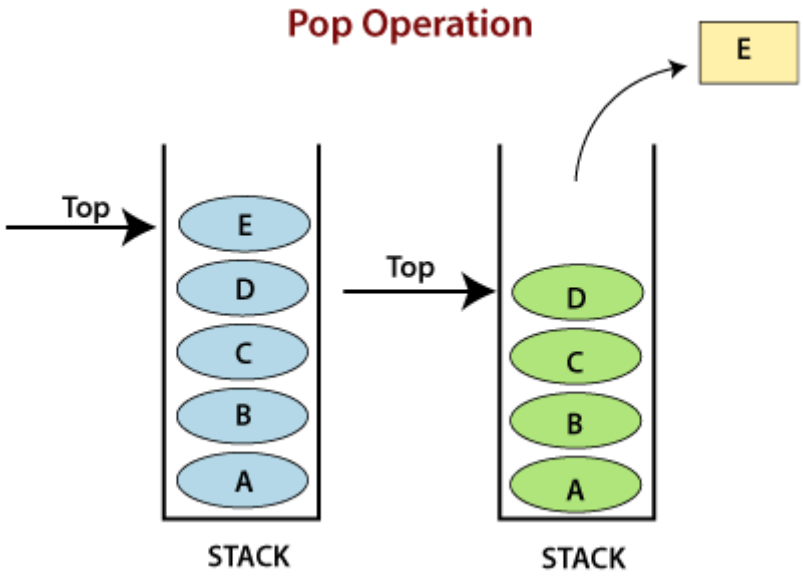
A Stack is a data structure that follows the LIFO(Last In First Out) principle. To implement a stack, we need two simple operations:

- **push -** It adds an element to the top of the stack.
- **pop -** It removes an element from the top of the stack.



Operations:

- **Adding -** It adds the items in the stack and increases the stack size. The addition takes place at the top of the stack.
- **Deletion -** It consists of two conditions, first, if no element is present in the stack, then underflow occurs in the stack, and second, if a stack contains some elements, then the topmost element gets removed. It reduces the stack size.
- **Traversing -** It involves visiting each element of the stack.



Characteristics:

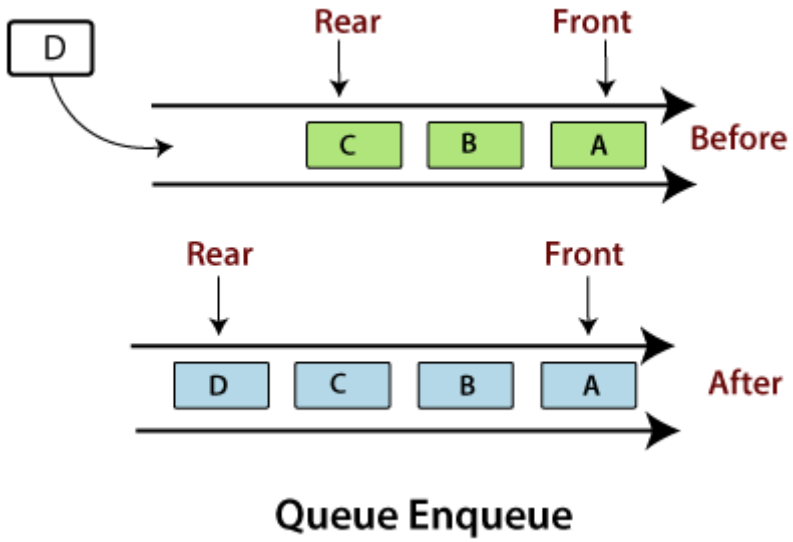
- Insertion order of the stack is preserved.
- Useful for parsing the operations.
- Duplicacy is allowed.

QUEUE:

A Queue follows the First-in-First-Out (FIFO) principle. It is opened from both the ends hence we can easily add elements to the back and can remove elements from the front.

To implement a queue, we need two simple operations:

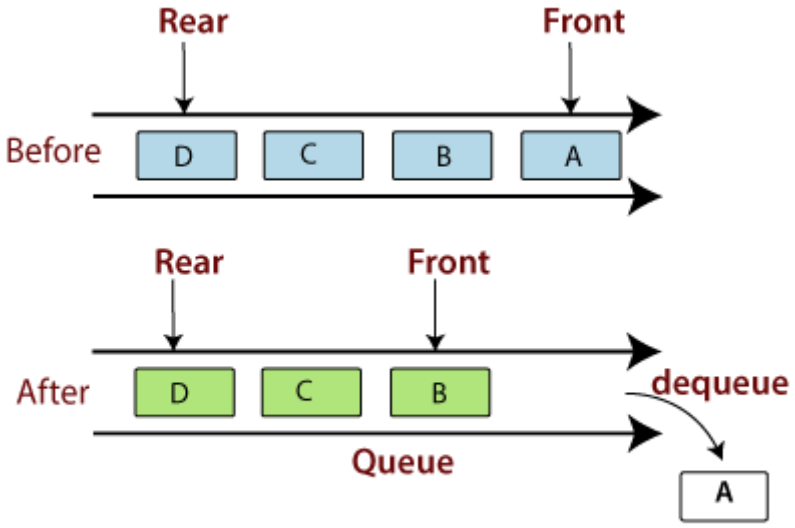
- **enqueue -** It adds an element to the end of the queue.
- **dequeue -** It removes the element from the beginning of the queue.



Queue Enqueue

Operations on Queue

- **Addition -** It adds the element in a queue and takes place at the rear end, i.e., at the back of the queue.
- **Deletion -** It consists of two conditions - If no element is present in the queue, Underflow occurs in the queue, or if a stack contains some elements then element present at the front gets deleted.
- **Traversing -** It involves to visit each element of the queue.



Queue Dequeue

Characteristics

- Insertion order of the queue is preserved.
- Duplicacy is allowed.
- Useful for parsing CPU task operations.

Note: **The implementation of a queue is a little bit different. A queue follows the "First-In-First-Out". Time plays an important factor here. The Stack is fast because we insert and pop the elements from the end of the list, whereas in the queue, the insertion and pops are made from the beginning of the list, so it becomes slow. The cause of this time difference is due to the properties of the list, which is fast in the end operation but slow at the beginning operations because all other elements have to be shifted one by one.**

- **PROGRAM:**

```
from collections import deque

class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class LinkedList:
    def __init__(self):
        self.head = None

    def insert_at_beginning(self, data):
        new_node = Node(data)
        new_node.next = self.head
```

```
        self.head = new_node

    def insert_at_end(self, data):
        if not self.head:
            self.head = Node(data)
            return
        temp = self.head
        while temp.next:
            temp = temp.next
        temp.next = Node(data)

    def insert_at_position(self, data, position):
        if position == 0:
            self.insert_at_beginning(data)
            return
        new_node = Node(data)
        temp = self.head
        for _ in range(position - 1):
            if temp is None:
                print("Position out of range")
                return
            temp = temp.next
        if temp is None:
            print("Position out of range")
            return
        new_node.next = temp.next
        temp.next = new_node

    def delete_at_position(self, position):
```

```
    if self.head is None:
        print("Linked list is empty")
        return
    temp = self.head
    if position == 0:
        self.head = temp.next
        temp = None
        return
    for _ in range(position - 1):
        temp = temp.next
        if temp is None:
            break
    if temp is None or temp.next is None:
        print("Position out of range")
        return
    next_node = temp.next.next
    temp.next = None
    temp.next = next_node

def delete_at_beginning(self):
    if not self.head:
        print("Linked list is empty")
        return
    self.head = self.head.next

def delete_at_end(self):
    if not self.head:
        print("Linked list is empty")
        return
```

```
        if not self.head.next:
            self.head = None
            return
        temp = self.head
        while temp.next.next:
            temp = temp.next
        temp.next = None

    def display(self):
        temp = self.head
        while temp:
            print(temp.data, end=" ")
            temp = temp.next
        print()

class Stack:
    def __init__(self):
        self.stack = []

    def push(self, data):
        self.stack.append(data)

    def pop(self):
        if not self.is_empty():
            return self.stack.pop()
        else:
            print("Stack is empty")

    def peek(self):
```

```
        if not self.is_empty():
            return self.stack[-1]
        else:
            print("Stack is empty")

    def is_empty(self):
        return len(self.stack) == 0

class Queue:
    def __init__(self):
        self.queue = deque()

    def enqueue(self, data):
        self.queue.append(data)

    def dequeue(self):
        if not self.is_empty():
            return self.queue.popleft()
        else:
            print("Queue is empty")

    def view(self):
        return list(self.queue)

    def is_empty(self):
        return len(self.queue) == 0

def linked_list_menu(ll):
    while True:
```

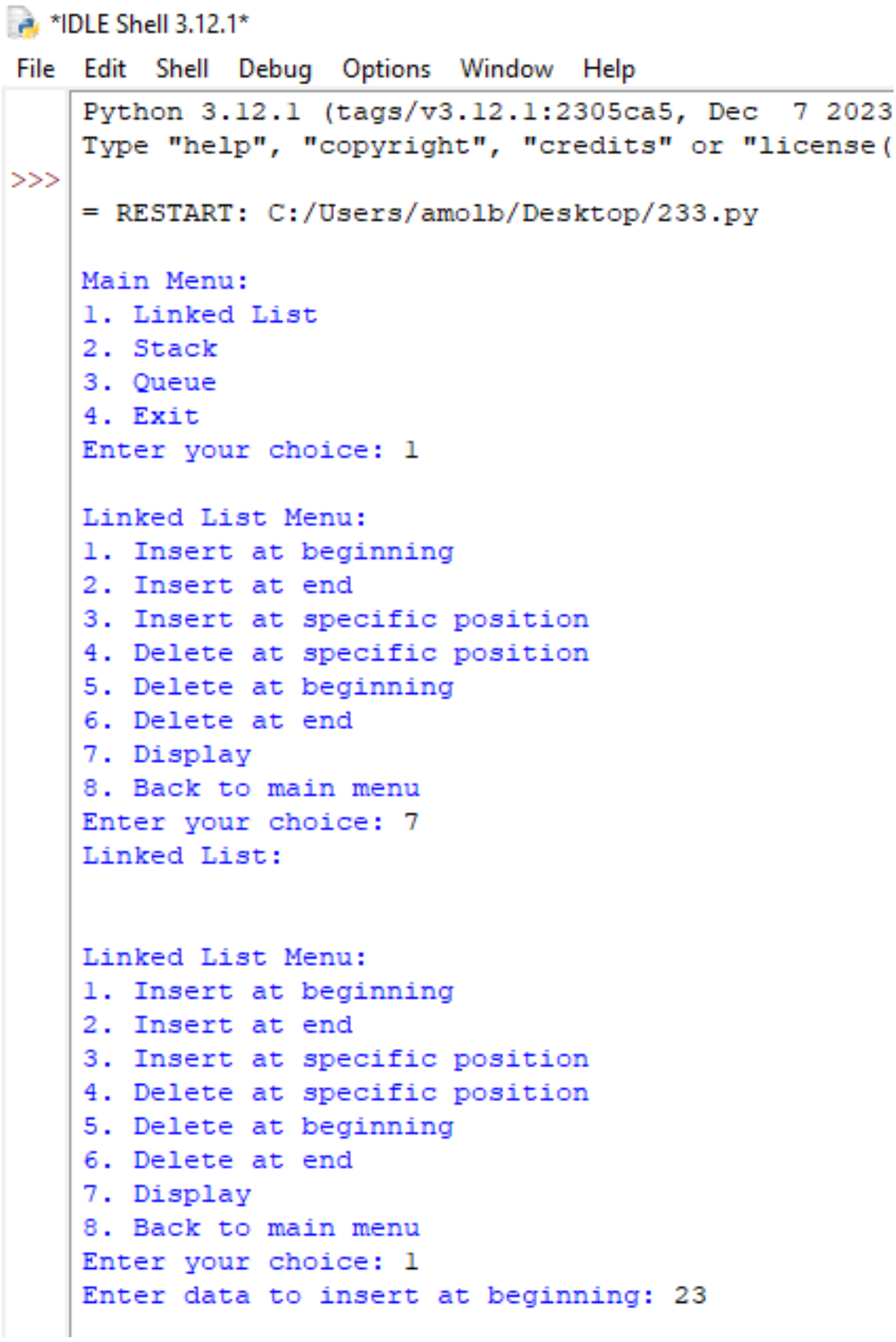
```
print("\nLinked List Menu:")
print("1. Insert at beginning")
print("2. Insert at end")
print("3. Insert at specific position")
print("4. Delete at specific position")
print("5. Delete at beginning")
print("6. Delete at end")
print("7. Display")
print("8. Back to main menu")
choice = int(input("Enter your choice: "))
if choice == 1:
    data = int(input("Enter data to insert at beginning: "))
    ll.insert_at_beginning(data)
elif choice == 2:
    data = int(input("Enter data to insert at end: "))
    ll.insert_at_end(data)
elif choice == 3:
    data = int(input("Enter data to insert: "))
    position = int(input("Enter position to insert at: "))
    ll.insert_at_position(data, position)
elif choice == 4:
    position = int(input("Enter position to delete: "))
    ll.delete_at_position(position)
elif choice == 5:
    ll.delete_at_beginning()
elif choice == 6:
    ll.delete_at_end()
elif choice == 7:
    print("Linked List:")
```

```
            ll.display()
        elif choice == 8:
            break
        else:
            print("Invalid choice")

def stack_menu(stack):
    while True:
        print("\nStack Menu:")
        print("1. Push")
        print("2. Pop")
        print("3. Peek")
        print("4. Back to main menu")
        choice = int(input("Enter your choice: "))
        if choice == 1:
            data = int(input("Enter data to push: "))
            stack.push(data)
        elif choice == 2:
            print("Popped element:", stack.pop())
        elif choice == 3:
            print("Top element:", stack.peek())
        elif choice == 4:
            break
        else:
            print("Invalid choice")

def queue_menu(queue):
    while True:
        print("\nQueue Menu:")
```

```
        print("1. Enqueue")
        print("2. Dequeue")
        print("3. View Queue")
        print("4. Back to main menu")
        choice = int(input("Enter your choice: "))
        if choice == 1:
            data = int(input("Enter data to enqueue: "))
            queue.enqueue(data)
        elif choice == 2:
            print("Dequeued element:", queue.dequeue())
        elif choice == 3:
            print("Queue:", queue.view())
        elif choice == 4:
            break
        else:
            print("Invalid choice")

if __name__ == "__main__":
    ll = LinkedList()
    stack = Stack()
    queue = Queue()

    while True:
        print("\nMain Menu:")
        print("1. Linked List")
        print("2. Stack")
        print("3. Queue")
        print("4. Exit")
        choice = int(input("Enter your choice: "))
```

```
if choice == 1:
    linked_list_menu(ll)
elif choice == 2:
    stack_menu(stack)
elif choice == 3:
    queue_menu(queue)
elif choice == 4:
    print("Exiting program")
    break
else:
    print("Invalid choice")
```

- **OUTPUT:**


IDLE Shell 3.12.1
File Edit Shell Debug Options Window Help
Python 3.12.1 (tags/v3.12.1:2305ca5, Dec 7 2023
Type "help", "copyright", "credits" or "license(
>>>
= RESTART: C:/Users/amolb/Desktop/233.py
Main Menu:
1. Linked List
2. Stack
3. Queue
4. Exit
Enter your choice: 1
Linked List Menu:
1. Insert at beginning
2. Insert at end
3. Insert at specific position
4. Delete at specific position
5. Delete at beginning
6. Delete at end
7. Display
8. Back to main menu
Enter your choice: 7
Linked List:
Linked List Menu:
1. Insert at beginning
2. Insert at end
3. Insert at specific position
4. Delete at specific position
5. Delete at beginning
6. Delete at end
7. Display
8. Back to main menu
Enter your choice: 1
Enter data to insert at beginning: 23

```
Linked List Menu:
1. Insert at beginning
2. Insert at end
3. Insert at specific position
4. Delete at specific position
5. Delete at beginning
6. Delete at end
7. Display
8. Back to main menu
Enter your choice: 1
Enter data to insert at beginning: 12

Linked List Menu:
1. Insert at beginning
2. Insert at end
3. Insert at specific position
4. Delete at specific position
5. Delete at beginning
6. Delete at end
7. Display
8. Back to main menu
Enter your choice: 2
Enter data to insert at end: 34

Linked List Menu:
1. Insert at beginning
2. Insert at end
3. Insert at specific position
4. Delete at specific position
5. Delete at beginning
6. Delete at end
7. Display
8. Back to main menu
Enter your choice: 3
Enter data to insert: 18
Enter position to insert at: 1
```

```
Linked List Menu:
1. Insert at beginning
2. Insert at end
3. Insert at specific position
4. Delete at specific position
5. Delete at beginning
6. Delete at end
7. Display
8. Back to main menu
Enter your choice: 7
Linked List:
12 18 23 34

Linked List Menu:
1. Insert at beginning
2. Insert at end
3. Insert at specific position
4. Delete at specific position
5. Delete at beginning
6. Delete at end
7. Display
8. Back to main menu
Enter your choice: 4
Enter position to delete: 1

Linked List Menu:
1. Insert at beginning
2. Insert at end
3. Insert at specific position
4. Delete at specific position
5. Delete at beginning
6. Delete at end
7. Display
8. Back to main menu
Enter your choice: 7
Linked List:
12 23 34
```

```
Linked List Menu:
1. Insert at beginning
2. Insert at end
3. Insert at specific position
4. Delete at specific position
5. Delete at beginning
6. Delete at end
7. Display
8. Back to main menu
Enter your choice: 5

Linked List Menu:
1. Insert at beginning
2. Insert at end
3. Insert at specific position
4. Delete at specific position
5. Delete at beginning
6. Delete at end
7. Display
8. Back to main menu
Enter your choice: 6

Linked List Menu:
1. Insert at beginning
2. Insert at end
3. Insert at specific position
4. Delete at specific position
5. Delete at beginning
6. Delete at end
7. Display
8. Back to main menu
Enter your choice: 7
Linked List:
23
```

```
Linked List Menu:
1. Insert at beginning
2. Insert at end
3. Insert at specific position
4. Delete at specific position
5. Delete at beginning
6. Delete at end
7. Display
8. Back to main menu
Enter your choice: 8

Main Menu:
1. Linked List
2. Stack
3. Queue
4. Exit
Enter your choice: 2

Stack Menu:
1. Push
2. Pop
3. Peek
4. Back to main menu
Enter your choice: 1
Enter data to push: 34

Stack Menu:
1. Push
2. Pop
3. Peek
4. Back to main menu
Enter your choice: 1
Enter data to push: 44
```

```
Stack Menu:
1. Push
2. Pop
3. Peek
4. Back to main menu
Enter your choice: 1
Enter data to push: 55

Stack Menu:
1. Push
2. Pop
3. Peek
4. Back to main menu
Enter your choice: 3
Top element: 55

Stack Menu:
1. Push
2. Pop
3. Peek
4. Back to main menu
Enter your choice: 2
Popped element: 55

Stack Menu:
1. Push
2. Pop
3. Peek
4. Back to main menu
Enter your choice: 2
Popped element: 44

Stack Menu:
1. Push
2. Pop
3. Peek
4. Back to main menu
Enter your choice: 2
Popped element: 34
```

```
Stack Menu:
1. Push
2. Pop
3. Peek
4. Back to main menu
Enter your choice: 2
Stack is empty
Popped element: None

Stack Menu:
1. Push
2. Pop
3. Peek
4. Back to main menu
Enter your choice: 3
Stack is empty
Top element: None

Stack Menu:
1. Push
2. Pop
3. Peek
4. Back to main menu
Enter your choice: 4
```

```
Main Menu:
1. Linked List
2. Stack
3. Queue
4. Exit
Enter your choice: 3

Queue Menu:
1. Enqueue
2. Dequeue
3. View Queue
4. Back to main menu
Enter your choice: 3
Queue: []

Queue Menu:
1. Enqueue
2. Dequeue
3. View Queue
4. Back to main menu
Enter your choice: 1
Enter data to enqueue: 23

Queue Menu:
1. Enqueue
2. Dequeue
3. View Queue
4. Back to main menu
Enter your choice: 1
Enter data to enqueue: 34

Queue Menu:
1. Enqueue
2. Dequeue
3. View Queue
4. Back to main menu
Enter your choice: 1
Enter data to enqueue: 45
```

```
Queue Menu:
1. Enqueue
2. Dequeue
3. View Queue
4. Back to main menu
Enter your choice: 3
Queue: [23, 34, 45]

Queue Menu:
1. Enqueue
2. Dequeue
3. View Queue
4. Back to main menu
Enter your choice: 2
Dequeued element: 23

Queue Menu:
1. Enqueue
2. Dequeue
3. View Queue
4. Back to main menu
Enter your choice: 2
Dequeued element: 34

Queue Menu:
1. Enqueue
2. Dequeue
3. View Queue
4. Back to main menu
Enter your choice: 2
Dequeued element: 45
```

```
Queue Menu:
1. Enqueue
2. Dequeue
3. View Queue
4. Back to main menu
Enter your choice: 2
Queue is empty
Dequeued element: None

Queue Menu:
1. Enqueue
2. Dequeue
3. View Queue
4. Back to main menu
Enter your choice: 4

Main Menu:
1. Linked List
2. Stack
3. Queue
4. Exit
Enter your choice: 4
Exiting program
```

- ***CONCLUSION:*** **Hence, we have successfully implemented Menu driven program for data structure using built in function for Link List, Stack and Queue; LO 3.**

PYTHON MINI-PROJECT REPORT

Group members:

Meet Raut - 2201084
Vaishnavi Poti - 2201078
Prasad Satpute - 2201089

HandWave: A Django Website for Indian Sign Language to English Conversion

Introduction

HandWave is a Django-based web application designed to bridge communication barriers by converting Indian Sign Language (ISL) into English text. Developed with the aim of enhancing accessibility for the hearing-impaired community, HandWave integrates various Python libraries such as OpenCV, MediaPipe, and TensorFlow to achieve accurate hand gesture recognition and translation.

Features

1. Real-time Hand Gesture Detection: HandWave utilizes the OpenCV and MediaPipe libraries to capture video from the user's webcam in real-time. The application employs advanced computer vision techniques to detect and track hand gestures accurately, allowing users to communicate seamlessly using ISL.

2. Machine Learning-based Translation: HandWave leverages machine learning techniques, specifically Keras and TensorFlow, to recognize hand gestures and translate them into English text. Trained on a dataset of ISL gestures, the model accurately predicts the corresponding English text for each detected gesture, enabling effective communication between ISL users and individuals proficient in English.

3. User-friendly Interface: The website features an intuitive user interface that makes it easy for users to interact with the application. With a simple design and clear instructions, HandWave ensures a seamless user experience for both ISL users and those unfamiliar with sign language.

How it Works

Upon accessing the HandWave website, users are prompted to enable their webcam to begin communication. The application then captures live video feed and employs computer vision algorithms to detect hand gestures in real-time. Once a gesture is recognized, the machine learning model processes the data and generates the corresponding English text, which is displayed to the user on the interface.

Benefits

1. Enhanced Accessibility: HandWave eliminates communication barriers for the hearing-impaired community by providing a platform for easy and effective communication in real-time.

2. Educational Tool: The application serves as an educational resource for individuals interested in learning ISL or understanding the basics of sign language communication.

3. Community Engagement: HandWave fosters community engagement and inclusivity by promoting awareness of ISL and facilitating communication between individuals with diverse linguistic backgrounds.

TECH STACK/TOOLS USED

The project utilized a combination of cutting-edge technologies to achieve its objectives. OpenCV and MediaPipe, renowned for their robust computer vision capabilities, formed the foundation of the gesture recognition system, enabling real-time capture and analysis of hand movements. These tools were instrumental in accurately detecting and tracking hand gestures, crucial for effective communication in sign language. Additionally, machine learning techniques powered by Keras and TensorFlow were integrated to enhance the system's intelligence. Leveraging these frameworks, the project implemented a sophisticated machine learning model capable of translating Indian Sign Language gestures into English text. Moreover, the website's frontend was developed using HTML and CSS, providing a user-friendly interface for seamless interaction with the application. Lastly, the backend was built on the Django framework, facilitating smooth integration of the various components and ensuring the website's robustness and scalability. Overall, the project employed a diverse array of tools and technologies to create an innovative solution for converting Indian Sign Language to English, thereby enhancing accessibility and inclusivity for the hearing-impaired community.

OUTPUT

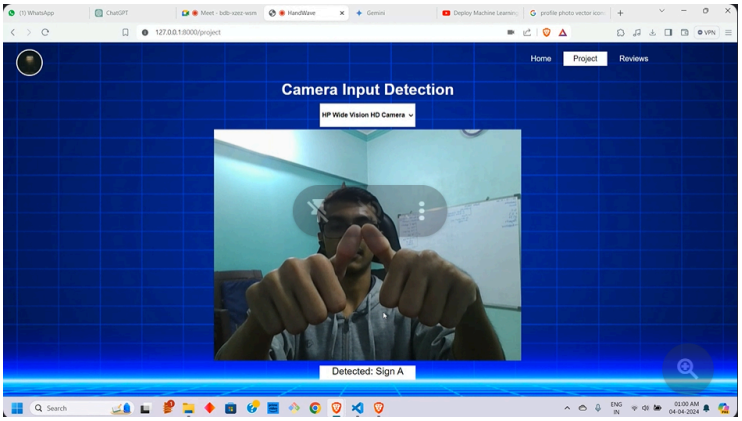


Fig 1.1- Django Website

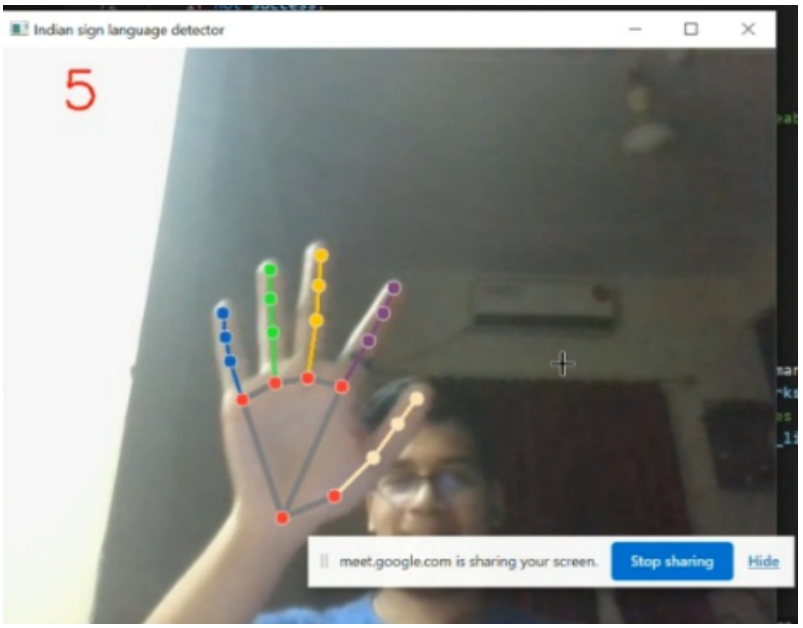


Fig 2.1- One hand gesture detection

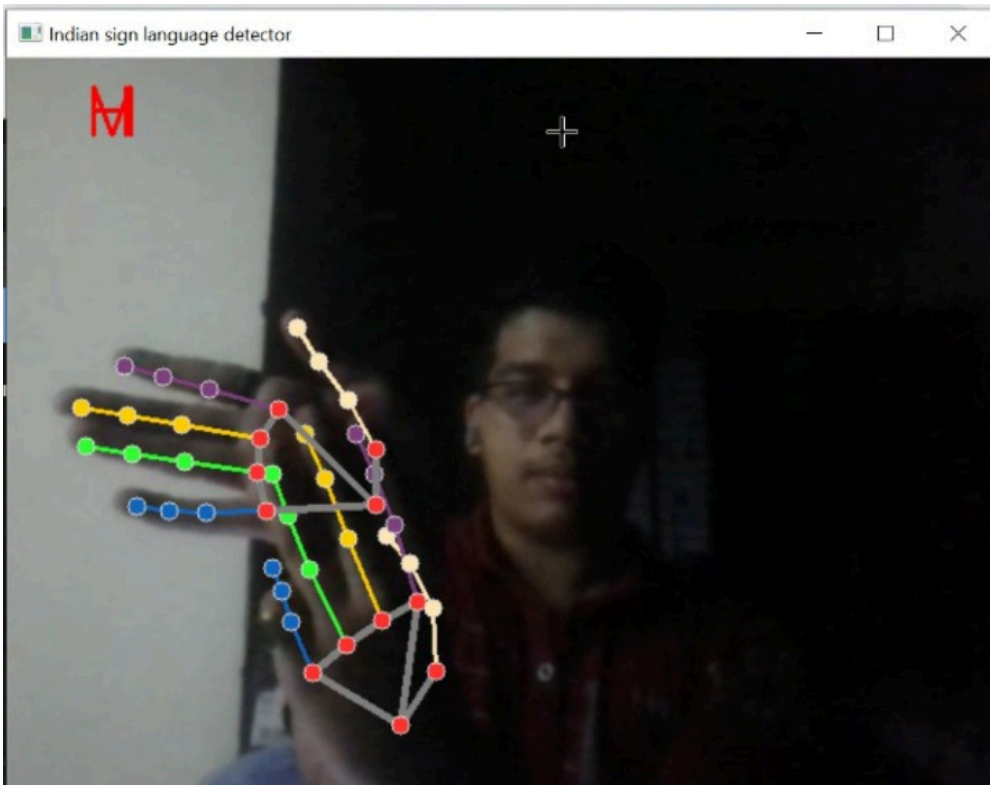
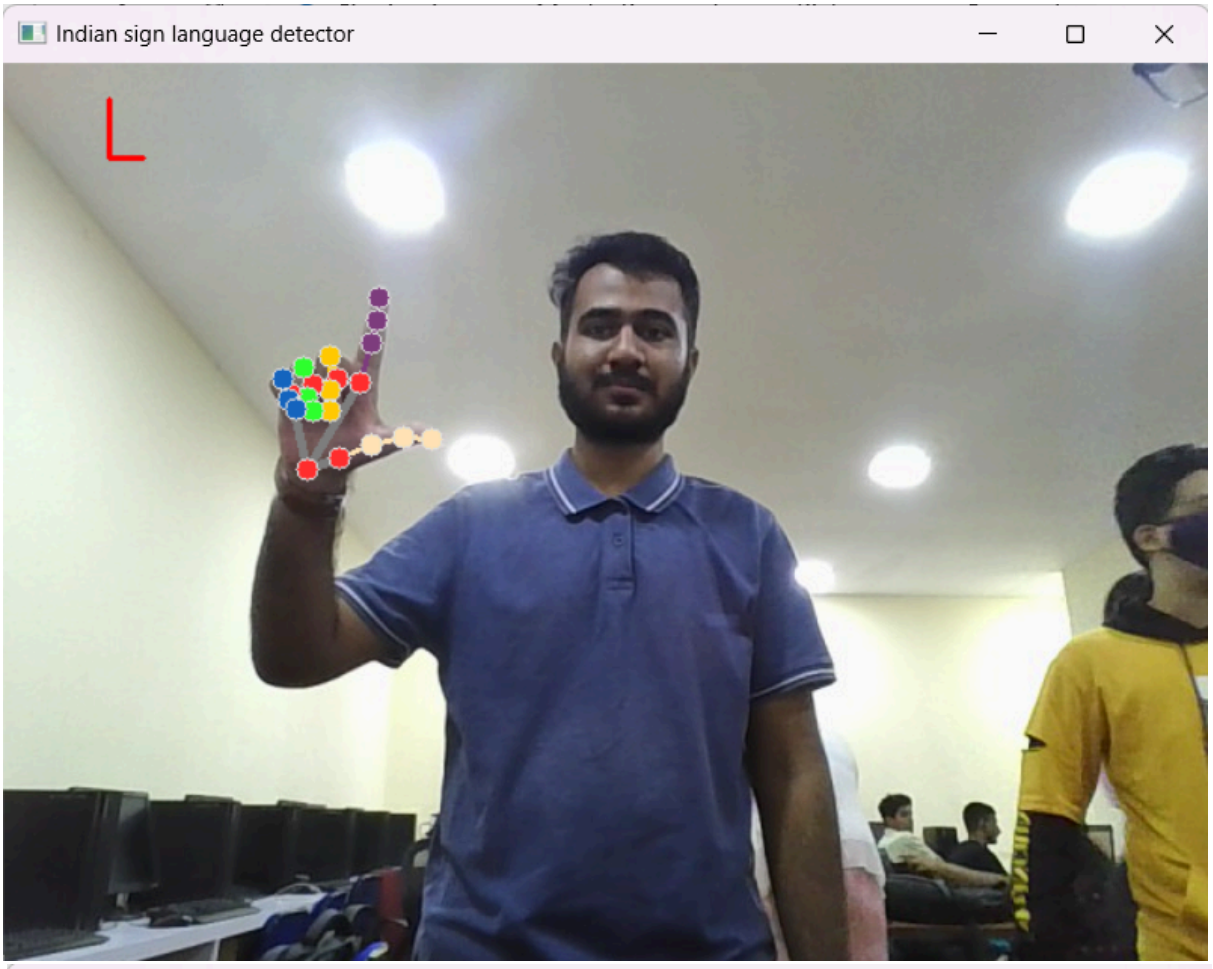


Fig 2.2- Two hand gesture detection


Indian sign language detector
L


Indian sign language detector
B

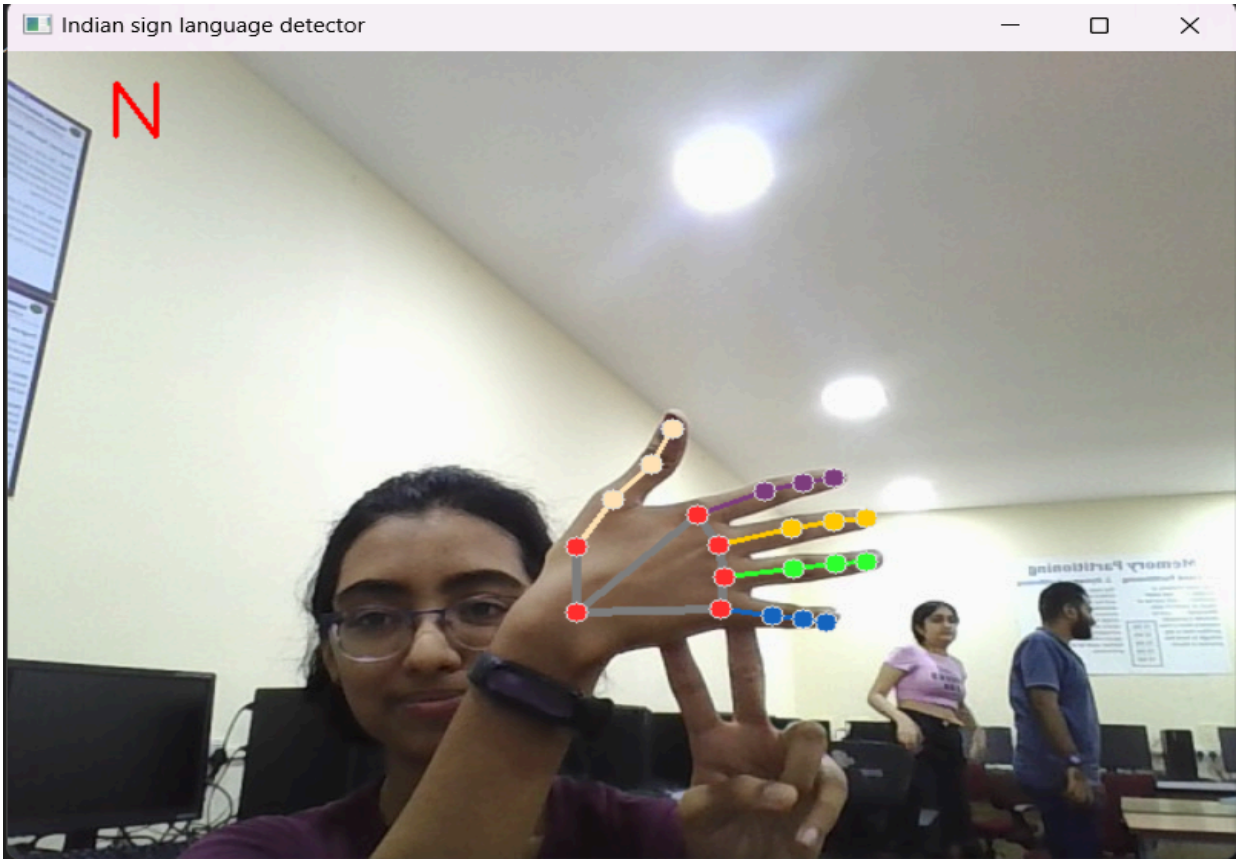


Fig 3.1,3.2,3.3,3.4- Other outputs showcasing gesture detection

Future Developments

Moving forward, the HandWave development team aims to enhance the application's functionality and accuracy further. Future updates may include:

- Integration of additional sign languages to cater to a broader user base.
- Improvements in gesture recognition algorithms to achieve higher accuracy and reliability.
- Implementation of user feedback mechanisms to continually refine and optimize the translation process.

Conclusion

HandWave represents a significant advancement in leveraging technology to promote inclusivity and accessibility for individuals with hearing impairments. By combining computer vision techniques with machine learning algorithms, the application enables seamless communication between ISL users and English speakers, thereby fostering greater connectivity and understanding in society. As the development of HandWave continues, it holds the potential to make a meaningful impact on the lives of individuals with hearing

disabilities, empowering them to express themselves freely and participate fully in a world of communication.

Meet A. Raut
S21 84

A++

Reviewed 14/05/24

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

Assignment 1

Q.1) Explain the manipulation functions of List, Tupple, dictionary and set in detail.

Ans. • Lists:

1. Append: list.append(x) adds an item x to the list at end.

2. Extending: list.extend(iterable) extends the list by appending elements from the iterable

3. Inserting: list.insert(i,x) inserts an item x at a specified index

4. Removing: list.remove(x) removes the first occurence of item x.

5. Popping: list.pop([i]) removes and returns the item at index i. If no index is specified, it removes and returns the last item.

6. Clearing: list.clear() removes all items from the list

7. Counting: list.count(x) returns the number of occurences of item x.

8. Sorting: list.sort() sorts the items of the lists in place.

• Tuples:

Tupes are the same as lists with the exception that the data once entered cannot be changed no matter what.

User can create, access and append to the tupple.

- ~~Tuples~~ Dictionaries:

1) Accessing: dict[key] retrieves the value associated with the specified key

2) Adding: dict[key] = value adds a new key-value pair

3) Removing: ~~dict~~ del dict[key] removes the item with the specified key

4) Getting keys: dict.keys() returns a view of all keys in the dictionary.

5) Clearing: dict.clear() removes all items from the dictionary

- Sets:

1) Adding: set.add(x) adds an element x to the set.

2) Clearing: set.clear() removes all elements from the set

3) Updating: set.update(iterable) updates the set by adding elements from the iterable.

4) Discarding: set.discard(x) removes element x from the set if it is present

5) Operations: Union, intersection, difference, Symmetric difference, subset, superset, disjointness are also common set operations

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

Q. Explain the constructs (polymorphism, encapsulation, inheritance) of OOP in detail.

Ans.

* Encapsulation:

Encapsulation is the bunding of data and methods that operates on that data into a single unit, called a class. In python, this is achieved through the use of classes. A class serves as a blueprint for creating objects, and it encapsulates both data attributes and methods that operate on those attributes. By encapsulating data and methods within a class we can control access to the data, making it possible to enforce constraints and ensure data integrity.

```
Class Car:
    def __init__(self, make, model):
        self.make = make          #public
        self._model = model       # protected
        self.__year = 2022        # private

    def get_year(self):
        return self.__year

car = Car("Toyota", "Camry")
print(car.make)      # public
print(car._model)    # protected
print(car.get_year()) # Accessing private attribute through method.
```

* Inheritance :

Inheritance is a mechanism in OOP that allows a new class to inherit properties and behavior from an existing class. In python, inheritance is achieved by specifying the superclass in parantheses after the subclass name when defining the subclass.

The subclass inherits all attributes and methods of its superclass and can also define its own additional attributes and methods.

```
class Vehicle :
    def __init__(self, make, model):
        self.make = make
        self.model = model

    def display_info(self):
        return f"{self.make} {self.model}"

Class Car(Vehicle):
    def __init__(self, make, model, year):
        super().__init__(make, model)
        self.year = year

    def display_info(self):
        return f"{self.year} {self.make} {self.model}"

car = car("Toyota", "camry", 2022)
print(car.display_info())
```

* Polymorphism:

polymorphism is the ability of different objects to respond to the same message in different ways. In python, polymorphism is achieved through method ~~overloading~~ overriding and method overloading.

~~Method overloading~~

Method overring allows a subclass to provide a specific implementation of a method that is already defined in its superclass.

Method overloading, on the other hand is not directly supported in Python as it is in some other languages like JAVA, but we can achieve a similar effect using default parameter values or variable length argument lists.

```
class Animal:
    def speak(self):
        pass

Class Dog(Animal):
    def speak(self):
        return "Woof!"

def make_sound(animal):
    return animal.speak()

dog = Dog()

print(make_sound(dog))
```

Q.3) Explain different anonymous function with examples.

Ans.

Anonymous functions, also known as lambda functions, are small inline functions that can be defined without a proper name or a def keyword. They are typically used for short, one-time operations where defining a full function using def is unnecessary.

Here are some different types of annonymous functions with examples:

1. Basic lambda function:

It takes one or more arguments and returns a value.

```
add = lambda x, y: x+y
print(add(3, 5))
```

2. Lambda function with conditional expression:

A lambda function with a conditional expression

```
check_even = lambda x: True if x % 2 == 0 else False
print(check_even(4))
```

3. Lambda function as a key function:

Using labda function as a key function for sorting.

```
names = ['Alice', 'Bob', 'Charlie', 'David']
sorted_names = sorted(names, key = lambda x: len(x))
print(sorted_names)
```

4. Lambda function with default arguments:
It is a function with default arguments.

```
power = lambda x, y=2: x ** y
print(power(3))
print(power(3,3))
```

5. Lambda function for mapping:
Using lambda functions with map() to apply a function to every element in an iterable.

```
numbers = [1, 2, 3, 4, 5]
squared_numbers = list(map(lambda x: x ** 2, numbers))
print(squared_numbers)
```

6. Lambda function for filtering:
Using lambda functions with filter() to filter elements from an iterable.

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
even_numbers = list(filter(lambda x: x % 2 == 0, numbers))
print(even_numbers)
```

Lambda functions provide a concise way to define small function without the need for a formal function definition. They are especially useful in scenarios where a simple operations needs to be performed inline, such as in sorting, filtering or mapping operation.

Meet Raut
S21 84

A++

26/03/24

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

Assignment 2

Q.1)

Ans. 1. Reading and writing data :

a) pd.read_csv():
Reads data from a CSV file into a Dataframe

b) pd.read-excel():
Reads data from an Excel file into a Dataframe

c) pd.to_csv():
Writes dataframe to a CSV file

d) pd.to-excel():
Writes Dataframe to an Excel file.

2. Viewing data :

a) df.head(n):
Returns the first n rows of the dataframe

b) df.tail(n):
Returns the last n rows of the dataframe

3. Filtering data :

a) df[df['column'] condition]:
Filters rows based on a condition

b) df.query('condition'):
Filters rows using a query expression.

4. Sorting data :

a) df.sort_values(by='column'):
Sorts the Dataframe by the values in a column.

b) df.sort_index():
Sorts the dataframe by index

5. Statistical Functions:

a) df.describe():

Generates descriptive statistics of the Dataframe.

b) df.mean()
df.median()
df.std()
:

Calculates various statistical measures.

Q.2)

Ans. Python supports various file operations, including:

1. Opening a file
2. Reading from a file
3. Writing to a file
4. Appending to a file
5. Closing a file

Code:

```
# Opening a file
file_path = "example.txt"
file = open(file_path, "r")

# Reading from a file
content = file.read()
print("File content:")
print(content)

# Closing a file
file.close()

# Writing to a file
file = open(file_path, "w")
file.write("This is new line")
file.close()
```

```
# Appending to a file
file = open(file_path, "a")
file.write("This line is appending"
file.close()
```

In this example:

1) We first open a file in read mode and print it's content
2) Then, we open the some file in write mode and write some content to it.
3) After that, we open file again in append mode and append some more content to it.

Q.3)

Ans. Networking in python refers to the ability to communicate and exchange data between computers over a network using Python. Python offers several built-in modules like socket, http, urllib & third-party libraries like requests for handling networking tasks.

1. Socket Programming:
It provides low-level networking interface for creating client-server applications, supporting both TCP and UDP protocols. It allows creating sockets, binding them to specific addresses and ports, and establishing connections for data transmission.

2. HTTP Requests:
It provides functionalities for making HTTP requests and handling responses. It allows fetching web pages, sending POST requests, handling cookies, and more.

3. URL Handling:
It provides utilities for handling URL's including parsing fetching and opening resources across the network.

4. Third-party libraries:
Libraries like requests simplify HTTP requests by providing a higher-level interface compared to the built-in module, making it easier to work with APIs and web services