python-programming-lab-1

March 11, 2025

Python Programming - 2301CS404

Lab - 1

Deep Vamja

Roll no :236

0.0.1 01) WAP to print “Hello World”

[1]:
```
print('hello world')
```

```
hello world
```

0.0.2 02) WAP to print addition of two numbers with and without using input().

a = 10 b = 5 print(a+b)

[21]:
```
a = int(input('enter first number'))
b = int(input('enter second number'))
print(a+b)
```

```
enter first number 3
enter second number 3
6
```

0.0.3 03) WAP to check the type of the variable.

[2]:
```
a = 10.0
print(type(a))
```

```
<class 'float'>
```

0.0.4 04) WAP to calculate simple interest.

[7]:
```
p = 100
r = 5
t = 3

SI = (p*r*t)/100
```

```
print(SI)
```

```
15.0
```

0.0.5 05) WAP to calculate area and perimeter of a circle.

[21]:
```
R = 5
AOC = 3.14*R*R

print(AOC)
```

```
78.5
```

perimeter of circle

[22]:
```
r = 5
c = 2*3.14*r
print(c)
```

```
31.400000000000002
```

0.0.6 06) WAP to calculate area of a triangle.

[23]:
```
b = 3
h = 4
a = (b*h)/2
print(a)
```

```
6.0
```

0.0.7 07) WAP to compute quotient and remainder.

[24]:
```
def compute(n, m):

        # for quotient
        q = n//m
        print("Quotient: ", q)

        # for remainder
        r = n%m
        print("Remainder", r)

compute(10, 3)
compute(99, 5)
```

```
Quotient:  3
Remainder 1
```

```
Quotient:  19
Remainder 4
```

0.0.8 08) WAP to convert degree into Fahrenheit and vice versa.

[25]:
```
F = 56
c = ((F) - 32) * 5/9
print(c)

C = 4
f = ((C) * 9/5) + 32
print(f)
```

```
13.333333333333334
39.2
```

0.0.9 09) WAP to find the distance between two points in 2-D space.

[26]:
```
import math

def distance(x1, y1, x2, y2):

  return math.sqrt((x2 - x1)**2 + (y2 - y1)**2)


x1, y1 = 9, 8
x2, y2 = 8, -5
print("The distance between given  points is:", distance(x1, y1, x2, y2))
```

```
The distance between given  points is: 13.038404810405298
```

0.0.10 10) WAP to print sum of n natural numbers.

[27]:
```
n = int(input('enter the number'))
sum = n*(n+1)/2
print(sum)
```

```
10.0
```

0.0.11 11) WAP to print sum of square of n natural numbers.

[28]:
```
n = 4
Sumofsquare = (n * (n + 1) * (2 * n + 1)) / 6
print (Sumofsquare)
```

```
30.0
```

0.0.12 12) WAP to concate the first and last name of the student.

[29]:
```
firstName = "deep"
lastName = "vamja"
res = firstName + " " + lastName
print(res)
```

```
deep vamja
```

0.0.13 13) WAP to swap two numbers.

[31]:
```
a = int(input('enter first num'))
b = int(input('enter second num'))

print ("Before swapping: ")
print("Value of a : ", a, " and b : ", b)


a, b = b, a

print ("After swapping: ")
print("Value of a : ", a, " and b : ", b)
```

```
Before swapping:
Value of a :  4  and b :  5
After swapping:
Value of a :  5  and b :  4
```

0.0.14 14) WAP to get the distance from user into kilometer, and convert it into meter, feet, inches and centimeter.

[33]:
```
kilometer=float(input("Enter The Kilometer : "))

m=kilometer*1000
cm=m*100
i=cm/2.54
ft=i/12
print("Kilometers : ",kilometer)
print("Meters  : ",m)
print("Centimeters : ",cm)
print("Inches : ",i)
print("Feet : ",ft)
```

```
Kilometers :  4.0
Meters  :  4000.0
Centimeters :  400000.0
Inches :  157480.3149606299
Feet :  13123.359580052493
```

0.0.15 15) WAP to get day, month and year from the user and print the date in the given format: 23-11-2024.

[34]:
```
day = int(input('enter the day'))
month = int(input('enter the month'))
year = int(input('enter the year'))

print((day),(month),(year),sep = "-")
```

```
12-2-2024
```

[ ]:
```
fp = open("")
```

ykluqrlxj

March 12, 2025

Python Programming - 2301CS404

Lab - 2

Deep Vamja

Roll no :236

1 if..else..

1.0.1 01) WAP to check whether the given number is positive or negative.

[1]:
```
a = int(input("Enter Number : "))
if a > 0:
    print("Positive")
else:
    print("Negative")
```

```
Positive
```

1.0.2 02) WAP to check whether the given number is odd or even.

[3]:
```
a = int(input("Enter Number : "))
if a % 2 == 0 :
    print("Even")
else :
    print("Odd")
```

```
Even
```

1.0.3 03) WAP to find out largest number from given two numbers using simple if and ternary operator.

[4]:
```
a = 4
b = 7

if a > b :
    print(a , "is Largest")
else :
```

```
    print(b , "is Largest")
```

```
7 is Largest
```

[5]:
```
a = 4
b = 7

print(a , "is Largest") if a > b else print(b , "is Largest")
```

```
7 is Largest
```

1.0.4 04) WAP to find out largest number from given three numbers.

[6]:
```
a = 14
b = 2
c = 8

if a > b :
    if a > c :
        print(a , "is Largest")
    else :
        print(c , "is Largest")
elif b > c :
    print(b , "is Largest")
else :
    print(c , "is Largest")
```

```
14 is Largest
```

1.0.5 05) WAP to check whether the given year is leap year or not.

[If a year can be divisible by 4 but not divisible by 100 then it is leap year but if it is divisible by 400 then it is leap year]

[7]:
```
year = 2023

if (year % 400 == 0) or (year % 4 == 0 and year % 100 != 0) :
    print("Leap Year")
else :
    print("Not a Leap Year")
```

```
Not a Leap Year
```

1.0.6 06) WAP in python to display the name of the day according to the number given by the user.

[2]:
```
day = int(input("Enter Number: "))

day = day % 7

match day:
    case 1:
        print("Monday")
    case 2:
        print("Tuesday")
    case 3:
        print("Wednesday")
    case 4:
        print("Thursday")
    case 5:
        print("Friday")
    case 6:
        print("Saturday")
    case 7:
        print("Sunday")
    case _:
        print("Invalid Input")
```

```
Wednesday
```

1.0.7 07) WAP to implement simple calculator which performs (add,sub,mul,div) of two no. based on user input.

[3]:
```
a = int(input("Enter First Number : "))
b = int(input("Enter Second Number : "))

ch = input("Enter Operator : ")

if ch == '+' :
    print("Sum : " , (a + b))
elif ch == '-' :
    print("Subtraction : " , (a - b))
elif ch == '*' :
    print("Multiplication : " , (a * b))
elif ch == '/' :
    print("Division : " , (a / b))
elif ch == '%' :
    print("Remainder : " , (a % b))
```

1.0.8 08) WAP to read marks of five subjects. Calculate percentage and print class accordingly.

Fail below 35 Pass Class between 35 to 45 Second Class between 45 to 60 First Class between 60 to 70 Distinction if more than 70

[6]:
```
a = int(input("Enter First Subject Mark : "))
b = int(input("Enter Second Subject Mark : "))
c = int(input("Enter Third Subject Mark : "))
d = int(input("Enter Fourth Subject Mark : "))
e = int(input("Enter Fifth Subject Mark : "))

percentage = (a + b + c + d + e) / 5

if percentage > 70 :
    print("Distinct Class")
elif percentage > 60 and percentage < 70 :
    print("First Class")
elif percentage > 45 and percentage < 60 :
    print("Second Class")
elif percentage > 35 and percentage < 45 :
    print("Pass Class")
elif percentage < 35 :
    print("Fail")
```

```
Pass Class
```

1.0.9 09) Three sides of a triangle are entered through the keyboard, WAP to check whether the triangle is isosceles, equilateral, scalene or right-angled triangle.

[8]:
```
a = int(input("Enter First Side Measure : "))
b = int(input("Enter Second Side Measure : "))
c = int(input("Enter Third Side Measure : "))

if a == b == c :
    print("Equilateral Triangle")
elif a == b or b == c or a == c :
    print("Isosceles Triangle")
elif (a*a + b*b == c*c) or (a*a + c*c == b*b) or (b*b + c*c == a*a) :
    print("Right-Angled Triangle")
else :
    print("Scalene Triangle")
```

```
Scalene Triangle
```

1.0.10 10) WAP to find the second largest number among three user input numbers.

[10]:
```
a = int(input("Enter First Number : "))
b = int(input("Enter Second Number : "))
c = int(input("Enter Third Number : "))

if (a > b and a < c) or (a < b and a > c) :
    print(a, "is Second Largest")
elif (b > a and b < c) or (b < a and b > c) :
    print(b, "is Second Largest")
else :
    print(c, "is Second Largest")
```

```
40 is Second Largest
```

1.0.11 11) WAP to calculate electricity bill based on following criteria. Which takes the unit from the user.

a. First 1 to 50 units – Rs. 2.60/unit
b. Next 50 to 100 units – Rs. 3.25/unit
c. Next 100 to 200 units – Rs. 5.26/unit
d. above 200 units – Rs. 8.45/unit

[1]:
```
unit = int(input("Enter Unit : "))
bill = 0

if unit <= 50:
    bill += unit * 2.6
elif unit <= 100:
    bill += 50 * 2.6
    bill += (unit - 50) * 3.25
elif unit <= 200:
    bill += 50 * 2.6
    bill += 50 * 3.25
    bill += (unit - 100) * 5.26
else:
    bill += 50 * 2.6
    bill += 50 * 3.25
    bill += 100 * 5.26
    bill += (unit - 200) * 8.45

print("Total Bill : " , bill)
```

```
Total Bill :  13.0
```

python-programming-lab-3

March 11, 2025

0.0.1 01) WAP to print 1 to 10.

Python Programming - 2301CS404

Lab - 3

Deep Vamja

Roll no :236

[ ]:
```
for i in range(1,10):
    print(i)
```

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

0.0.2 02) WAP to print 1 to n.

[ ]:
```
number = int(input("enter the number"))
for i in range(1,number+1):
    print(i)
```

```
enter the number 5
1
2
3
4
5
```

0.0.3 03) WAP to print odd numbers between 1 to n.

[ ]:
```
number = int(input("enter the number"))
for i in range(1,number+1):
        if(i%2==1):
            print(i)
```

```
enter the number 6
1
3
5
```

4) WAP to print numbers between two given numbers which is divisible by 2 but not divisible by 3.

[ ]:
```
start = int(input("Enter the start number: "))
end = int(input("Enter the end number: "))

def print_num(start, end):
    for num in range(start, end + 1):
        if num % 2 == 0 and num % 3 != 0:
            print(num, end=" ")

print_num(start, end)
```

0.0.4 05) WAP to print sum of 1 to n numbers.

[ ]:
```
def Sum_of_Numbers(n):
    sum= 0
    for i in range(1, n + 1):
        sum += i
    return sum


n = int(input("Enter a number n: "))


result = Sum_of_Numbers(n)
print(result)
```

```
Enter a number n:  3
{6}
```

0.0.5 06) WAP to print sum of series 1 + 4 + 9 + 16 + 25 + 36 + ...n.

[ ]:
```
n = int(input("Enter a number n: "))

def Sum_of_Series(n):
    sum = 0
    i = 1
    while i * i <= n:
        sum += i * i
        i += 1
    return sum

result = Sum_of_Series(n)
print(result)
```

0.0.6 07) WAP to print sum of series 1 − 2 + 3 − 4 + 5 − 6 + 7 ... n.

[ ]:
```
n = int(input("Enter a number n: "))

def sum_of_series(n):
    sum = 0
    for i in range(1, n + 1):

        if i % 2 != 0:
            sum = sum + i
        else:
            sum = sum - i
    return sum

result = sum_of_series(n)
print("The sum of the series 1 - 2 + 3 - 4 + 5 - 6 + ... up to {n} is:␣
 ↪",{result})
```

```
Enter a number n:  4
The sum of the series 1 - 2 + 3 - 4 + 5 - 6 + … up to {n} is:  {-2}
```

0.0.7 08) WAP to print multiplication table of given number.

[ ]:
```
n=int(input("enter a number"))
for i in range(1,11,1):
    print(n,'x',i,'=',n*i)
```

```
enter a number 6
6 x 1 = 6
6 x 2 = 12
6 x 3 = 18
6 x 4 = 24
```

```
6 x 5 = 30
6 x 6 = 36
6 x 7 = 42
6 x 8 = 48
6 x 9 = 54
6 x 10 = 60
```

0.0.8 09) WAP to find factorial of the given number.

[ ]:
```
def factorial(n):
    if n < 0:
        print("Factorial is not defined for negative numbers.")
    res = 1
    for i in range(1, n + 1):
        res *= i
    return res


number=int(input("enter the number to find its factorial"))
print(f"The factorial of {number} is {factorial(number)}.")
```

```
enter the number to find its factorial 5
The factorial of 5 is 120.
```

0.0.9 10) WAP to find factors of the given number.

[ ]:
```
def factors(x):
   print("The factors of",x,"are:")
   for i in range(1, x + 1):
       if x % i == 0:
           print(i)

num=int(input("enter a number"))

factors(num)
```

```
enter a number 3
The factors of 3 are:
1
3
```

0.0.10 11) WAP to find whether the given number is prime or not.

[ ]:
```
num=int(input("enter the number"))

if num > 1:

    for i in range(2, (num//2)+1):
```

```
        if ((num % i) == 0):
            print(num, "is not a prime number")
            break
    else:
        print(num, "is a prime number")
else:
    print(num, "is not a prime number")
```

```
enter the number 6
6 is not a prime number
```

0.0.11 12) WAP to print sum of digits of given number.

[ ]:
```
def Sum(n):

    sum = 0
    for digit in str(n):
      sum += int(digit)
    return sum

n = int(input("enter a number"))
print(Sum(n))
```

```
enter a number 56
11
```

0.0.12 13) WAP to check whether the given number is palindrome or not

[ ]:
```
num=int(input("enter the number"))
temp = num
reverse = 0
while temp > 0:
    remainder = temp % 10
    reverse = (reverse * 10) + remainder
    temp = temp // 10
if num == reverse:
  print('Palindrome')
else:
  print("Not Palindrome")
```

```
enter the number 121
Palindrome
```

0.0.13 14) WAP to print GCD of given two numbers.

[ ]:
```
num1 = int(input("enter a num1"))
num2 = int(input("enter a num2"))
gcd = 1

for i in range(1, min(num1, num2)):
    if num1 % i == 0 and num2 % i == 0:
        gcd = i
print(f" GCD of {num1} & {num2} is", gcd)
```

```
enter a num1 5
enter a num2 6
 GCD of 5 & 6 is 1
```

python-programming-lab-4

March 11, 2025

Python Programming - 2301CS404

Lab - 4

Deep Vamja

Roll no :236

1 String

1.0.1 01) WAP to check whether the given string is palindrome or not.

[15]:
```
def isPalindrome(s):
    return s == s[::-1]

s = input('enter the string')
ans = isPalindrome(s)

if ans:
    print("Yes it is a palindrme")
else:
    print("No it is not a palindrome")
```

```
Yes it is a palindrme
```

1.0.2 02) WAP to reverse the words in the given string.

[ ]:
```
s = input('enter the string')

s = s[::-1]
print(s)
```

1.0.3 03) WAP to remove ith character from given string.

[40]:
```
s = "deep"
s = s.removeprefix('d')
print(s)
```

```
s2 = "deep"
s2 = s2.removesuffix('p')
print(s2)



def remove_ith_character(s, i):

    if i < 0 or i >= len(s):
        return "Invalid index"

    return s[:i] + s[i+1:]


input_string = "hello world"
index = 4
result = remove_ith_character(input_string, index)
print("String after removing character:", result)
```

```
eep
dee
String after removing character: hell world
```

1.0.4 04) WAP to find length of string without using len function.

[41]:
```
def findLen(str):
    counter = 0
    while str[counter:]:
        counter += 1
    return counter

str = 'hello deep'
print(findLen(str))
```

```
10
```

1.0.5 05) WAP to print even length word in string.

[42]:
```
S="This is a python language"

s = S.split(" ")
for i in s:
    if len(i)%2==0:
        print(i)
```

```
This
is
```

```
python
language
```

1.0.6 06) WAP to count numbers of vowels in given string.

[ ]:
```
string = input('enter the string')
vowels = input('enter the vowel')

count = sum(string.count(vowel) for vowel in vowels)
print(count)
```

1.0.7 07) WAP to capitalize the first and last character of each word in a string.

[26]:
```
def capitalize_first_last(string):
    words = string.split()

    result = [
        word[0].upper() + word[1:-1] + word[-1].upper() if len(word) > 1 else␣
 ↪word.upper()
        for word in words
    ]
    return ' '.join(result)

input_string = input("Enter a string: ")
output_string = capitalize_first_last(input_string)
print("Output:", output_string)
```

```
Output: HellO FroM DeeP
```

1.0.8 08) WAP to convert given array to string.

[27]:
```
def array_to_string(arr):

    return ''.join(arr)

array = ['H', 'e', 'l', 'l', 'o']
result = array_to_string(array)
print("Array is converted to string:", result)
```

```
Array is converted to string: Hello
```

1.0.9 09) Check if the password and confirm password is same or not.

1.0.10 In case of only case's mistake, show the error message.

[22]:
```
def check_passwords(password, confirm_password):
    if password == confirm_password:
        print("Passwords match.")
    elif password.lower() == confirm_password.lower():
        print("Error: Passwords do not match due to case sensitivity in given␣
 ↪string.")
    else:
        print("Error: Passwords do not match.")


password = input("Enter password: ")
confirm_password = input("Enter confirm password: ")
check_passwords(password, confirm_password)
```

```
Passwords match.
```

1.0.11 10) : Display credit card number.

1.0.12 card no. : 1234 5678 9012 3456

1.0.13 display as : **** **** **** 3456

[4]:
```
def credit_card(card_number):

    parts = card_number.split()

    masked_parts = ["****"] * (len(parts) - 1) + [parts[-1]]

    return " ".join(masked_parts)

card_number = "1234 5678 9012 3456"
masked_card = credit_card(card_number)
print("credit card number:", masked_card)
```

```
credit card number: **** **** **** 3456
```

1.0.14 11) : Checking if the two strings are Anagram or not.

1.0.15 s1 = decimal and s2 = medical are Anagram

[5]:
```
def are_anagrams(s1, s2):

    s1 = s1.replace(" ", "").lower()
    s2 = s2.replace(" ", "").lower()
```

```
    return sorted(s1) == sorted(s2)


s1 = "decimal"
s2 = "medical"

if are_anagrams(s1, s2):
    print(f'"{s1}" and "{s2}" are anagrams.')
else:
    print(f'"{s1}" and "{s2}" are not anagrams.')
```

```
"decimal" and "medical" are anagrams.
```

1.0.16 12) : Rearrange the given string. First lowercase then uppercase alphabets.

1.0.17 input : EHlsarwiwhtwMV

1.0.18 output : lsarwiwhtwEHMV

[6]:
```
def rearrange_string(s):

    lowercase = "".join([ch for ch in s if ch.islower()])
    uppercase = "".join([ch for ch in s if ch.isupper()])

    return lowercase + uppercase


input_string = "EHlsarwiwhtwMV"
output_string = rearrange_string(input_string)
print("Rearranged string:", output_string)
```

```
Rearranged string: lsarwiwhtwEHMV
```

python-programming-lab-5

March 11, 2025

1 List

Python Programming - 2301CS404

Lab - 5

Deep Vamja

Roll no :236

1.0.1 01) WAP to find sum of all the elements in a List.

[3]:
```
l = [1,2,3,4,5]
sum = 0
for i in l:
    sum += i

print("Sum : " , sum)
```

```
Sum :  15
```

1.0.2 02) WAP to find largest element in a List.

[7]:
```
l = [1,2,3,5,4]
max = l[0]

for i in l :
    if max < i :
        max = i

print("Max : " , max)
```

```
Max :  5
```

1.0.3 03) WAP to find the length of a List.

[8]:
```
l = [1,2,3,4,5]
length = 0
```

```
for i in l :
    length += 1

print("Length : " , length)
```

```
Length :  5
```

1.0.4 04) WAP to interchange first and last elements in a list.

[34]:
```
l = [1,2,3,4,5]

print("Before Swap :->")
for i in l :
    print(i)

temp = l[0]
l[0] = l[- 1]
l[- 1] = temp

print("After Swap :->")
for i in l :
    print(i)
```

```
Before Swap :->
1
2
3
4
5
After Swap :->
5
2
3
4
1
```

1.0.5 05) WAP to split the List into two parts and append the first part to the end.

[29]:
```
l = [1,2,3,4,5,6]
l1 = []
l2 = []

for i in range(len(l) // 2) :
    l1.append(l[i])

for i in range(len(l) // 2 , len(l)) :
    l2.append(l[i])
```

```
l2.extend(l1)

l2
```

[29]: [4, 5, 6, 1, 2, 3]

1.0.6 06) WAP to interchange the elements on two positions entered by a user.

[33]:
```
a = int(input("Enter First Position : "))
b = int(input("Enter Second Position : "))

l = [1,2,3,4,5]

if (a > 0 and a < len(l)) and (b > 0 and b < len(l)) :
    temp = l[a]
    l[a] = l[b]
    l[b] = temp

l
```

[33]: [1, 2, 4, 3, 5]

1.0.7 07) WAP to reverse the list entered by user.

[ ]:
```
size = int(input("Enter Size Of List : "))
l = []

for i in range(size) :
    l.append(int(input("Enter Element : ")))

l = l [::-1]
l
```

[ ]: [1, 2, 5]

1.0.8 08) WAP to print even numbers in a list.

[39]:
```
l = [1,2,3,4,5]

for i in l :
    if i % 2 == 0 :
        print(i)
```

2
4

1.0.9 09) WAP to count unique items in a list.

[56]:
```
l = [1,2,1,4,3,5,5,2]

for i in l :
    if l.count(i) == 1:
        print(i)
```

```
4
3
```

1.0.10 10) WAP to copy a list.

[40]:
```
l = [1,2,3,4]

l1 = l.copy()

l1
```

[40]:
```
[1, 2, 3, 4]
```

1.0.11 11) WAP to print all odd numbers in a given range.

[41]:
```
a = int(input("Enter First Number : "))
b = int(input("Enter Second Number : "))

for i in range(a, (b+1)) :
    if i % 2 != 0 :
        print(i)
```

```
3
5
7
```

1.0.12 12) WAP to count occurrences of an element in a list.

[42]:
```
l = [1,2,3,1,2,3,2]

l.count(2)
```

[42]:
```
3
```

1.0.13 13) WAP to find second largest number in a list.

[47]:
```
l = [1,2,3,5,4]
max = l[0]
secondMax = l[0]

for i in l :
    if max < i :
        secondMax = max
        max = i
    elif i > secondMax and i != max :
        secondMax = i

print("Second Max : " , secondMax)
```

```
Second Max :  4
```

1.0.14 14) WAP to extract elements with frequency greater than K.

[59]:
```
l = [1,2,3,1,2,3,1,4,2,5]
unique = []

a = int(input("Enter Value : "))

for i in l :
    if i not in unique :
        unique.append(i)

for i in unique :
    if l.count(i) > a:
        print(i)
```

```
1
2
```

1.0.15 15) WAP to create a list of squared numbers from 0 to 9 with and without using List Comprehension.

[60]:
```
l = [i ** 2 for i in range(10)]
l
```

[60]:
```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

[61]:
```
l = []

for i in range(10) :
    l.append(i ** 2)
```

```
l
```

[61]: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

1.0.16 16) WAP to create a new list (fruit whose name starts with 'b') from the list of fruits given by user.

[65]:
```
size = int(input("Enter List Size :"))
l = []

for i in range(size) :
    l.append(input("Enter Fruit : "))

b =[]

for i in l :
    if i.startswith("b") or i.startswith("B") :
        b.append(i)

b
```

[65]: ['Banana']

1.0.17 17) WAP to create a list of common elements from given two lists.

[68]:
```
l1 = [3,2,4,5,3]
l2 = [3,1,5]

common = []
unique = []

for i in l1 :
    if i in l2 :
        common.append(i)

for i in common :
    if i not in unique :
        unique.append(i)

unique
```

[68]: [3, 5]

python-programming-lab-6

March 11, 2025

Python Programming - 2301CS404

Lab - 6

Deep Vamja

Roll no :236

1 Tuple

1.0.1 01) WAP to find sum of tuple elements.

[1]:
```
tuple_data = (10, 20, 30, 40)
sum_of_elements = sum(tuple_data)
print("Sum of tuple elements:", sum_of_elements)
```

```
Sum of tuple elements: 100
```

1.0.2 02) WAP to find Maximum and Minimum K elements in a given tuple.

[2]:
```
# Program to find Maximum and Minimum K elements in a tuple
tuple_data = (10, 20, 30, 40, 50, 60, 70)
k = 2

# Sorting the tuple
sorted_tuple = sorted(tuple_data)

# Finding minimum and maximum K elements
min_k_elements = sorted_tuple[:k]
max_k_elements = sorted_tuple[-k:]

print(f"Minimum {k} elements:", min_k_elements)
print(f"Maximum {k} elements:", max_k_elements)
```

```
Minimum 2 elements: [10, 20]
Maximum 2 elements: [60, 70]
```

1.0.3 03) WAP to find tuples which have all elements divisible by K from a list of tuples.

[3]:
```
# Program to find tuples with all elements divisible by K
list_of_tuples = [(10, 20), (15, 30), (40, 50), (12, 24)]
k = 5

result = [tup for tup in list_of_tuples if all(x % k == 0 for x in tup)]
print(f"Tuples with all elements divisible by {k}:", result)
```

```
Tuples with all elements divisible by 5: [(10, 20), (15, 30), (40, 50)]
```

1.0.4 04) WAP to create a list of tuples from given list having number and its cube in each tuple.

[4]:
```
# Program to create a list of tuples with number and its cube
numbers = [1, 2, 3, 4, 5]
result = [(num, num**3) for num in numbers]
print("List of tuples with number and its cube:", result)
```

```
List of tuples with number and its cube: [(1, 1), (2, 8), (3, 27), (4, 64), (5,
125)]
```

1.0.5 05) WAP to find tuples with all positive elements from the given list of tuples.

[5]:
```
# Program to find tuples with all positive elements
list_of_tuples = [(1, 2, 3), (-1, 4, 5), (6, -7, 8), (9, 10)]
positive_tuples = [tup for tup in list_of_tuples if all(x > 0 for x in tup)]
print("Tuples with all positive elements:", positive_tuples)
```

```
Tuples with all positive elements: [(1, 2, 3), (9, 10)]
```

1.0.6 06) WAP to add tuple to list and vice – versa.

[6]:
```
# Program to add a tuple to a list
my_list = [1, 2, 3]
my_tuple = (4, 5, 6)

# Adding tuple to the list
my_list.append(my_tuple)
print("List after adding tuple:", my_list)

# Program to add a list to a tuple
my_tuple = my_tuple + tuple(my_list)
print("Tuple after adding list:", my_tuple)
```

```
List after adding tuple: [1, 2, 3, (4, 5, 6)]
Tuple after adding list: (4, 5, 6, 1, 2, 3, (4, 5, 6))
```

1.0.7 07) WAP to remove tuples of length K.

[7]:
```
# Program to remove tuples of length K
list_of_tuples = [(1, 2), (3, 4, 5), (6,), (7, 8, 9, 10)]
k = 2
filtered_tuples = [tup for tup in list_of_tuples if len(tup) != k]
print("Tuples after removing length", k, ":", filtered_tuples)
```

```
Tuples after removing length 2 : [(3, 4, 5), (6,), (7, 8, 9, 10)]
```

1.0.8 08) WAP to remove duplicates from tuple.

[8]:
```
# Program to remove duplicates from a tuple
tuple_data = (10, 20, 20, 30, 40, 40)
unique_elements = tuple(set(tuple_data))
print("Tuple after removing duplicates:", unique_elements)
```

```
Tuple after removing duplicates: (40, 10, 20, 30)
```

1.0.9 09) WAP to multiply adjacent elements of a tuple and print that resultant tuple.

[9]:
```
# Program to multiply adjacent elements of a tuple
tuple_data = (1, 2, 3, 4, 5)
result = tuple(tuple_data[i] * tuple_data[i+1] for i in range(len(tuple_data) -␣
↪1))
print("Resultant tuple:", result)
```

```
Resultant tuple: (2, 6, 12, 20)
```

1.0.10 10) WAP to test if the given tuple is distinct or not.

[10]:
```
# Program to check if the given tuple has distinct elements
tuple_data = (10, 20, 30, 40, 10)
is_distinct = len(tuple_data) == len(set(tuple_data))
if is_distinct:
    print("The tuple is distinct.")
else:
    print("The tuple is not distinct.")
```

```
The tuple is not distinct.
```

python-programming-lab-7

March 11, 2025

Python Programming - 2301CS404

Lab - 7

Deep Vamja

Roll no :236

1 Set & Dictionary

1.0.1 01) WAP to iterate over a set.

[2]:
```
S = {1,2,3,4,5}

for i in S:
    print(i)
```

```
1
2
3
4
5
```

1.0.2 02) WAP to convert set into list, string and tuple.

[11]:
```
S = {11,2,3,4,5}

lists = list(S)
tuples = tuple(S)

str1 = " "

for i in S:
    str1 += str(i) + " "


print(str1)
print(lists)
print(tuples)
```

```
 2 3 4 5 11
[2, 3, 4, 5, 11]
(2, 3, 4, 5, 11)
```

1.0.3 03) WAP to find Maximum and Minimum from a set.

[13]:
```
s = {11,22,33,55}

S = max(s)
print(S)
```

```
55
```

1.0.4 04) WAP to perform union of two sets.

[15]:
```
S1 = {11,22,33,44,55}
S2 = {66,77,88,99,100}

sets = S1.union(S2)

print(sets)
```

```
{33, 66, 99, 100, 11, 44, 77, 22, 55, 88}
```

1.0.5 05) WAP to check if two lists have at-least one element common.

[21]:
```
l1 = [11,22,33,44,55]
l2 = [66,77,33,99,11]



sets = set(S1).intersection(set((S2)))
print(sets)
```

```
{33, 11}
```

1.0.6 06) WAP to remove duplicates from list.

[22]:
```
l = [1,2,3,4,5,6,6,4,]

n = list(set(l))

print(n)
```

```
[1, 2, 3, 4, 5, 6]
```

1.0.7 07) WAP to find unique words in the given string.

[23]:
```
s = input("enter the string:")

unique_word = set(s.split(" "))
print(unique_word)
```

```
{'hi', 'how', 'you', 'are'}
```

1.0.8 08) WAP to remove common elements of set A & B from set A.

[25]:
```
S1 = {11,22,33,44,55}
S2 = {66,77,33,99,11}

A = S1.difference(S2)
print(A)
```

```
{44, 22, 55}
```

1.0.9 09) WAP to check whether two given strings are anagram or not using set.

[45]:
```
string1 = "medicalm"
string2 = "demical"

if len(set(string1).difference(set(string2))) == 0  and len(string1) ==␣
 ↪len(string2):
    print("given string is anagram")
else:
    print("not anagram")
```

```
not anagram
```

1.0.10 10) WAP to find common elements in three lists using set.

[46]:
```
l1 = [11,22,33,44,55]
l2 = [66,77,33,99,11]
l3 = [11,77,66,33,99]

common_elements = set(l1).intersection(set(l2))
A = set(common_elements).intersection(set(l3))
print(A)
```

```
{33, 11}
```

1.0.11 11) WAP to count number of vowels in given string using set.

[54]:
```
vowels = "aeiou"
string = "enter the string".lower()

count_vowels = set(vowels).intersection(set(string))

vowel = 0
for i in count_vowels:
    vowel +=   string.count(i)

print(vowel)
```

```
4
```

1.0.12 12) WAP to check if a given string is binary string or not.

[55]:
```
def check(string):
        if all((letter in "01") for letter in string):
                return "Yes"
        return "No"


if __name__ == "__main__":

        string1 = "101011000111"
        string2 = "201000001"

        # function calling
        print(check(string1))
        print(check(string2))
```

1.0.13 13) WAP to sort dictionary by key or value.

[56]:
```
d = {"harsh":"vekariya","nikung":"rathod","deep":"vamja"}

sorted_keys = sorted(d)

sorted_dict = {}

for i in d:
    sorted_dict[i] = d.get(i)


print(sorted_keys)
```

```
['deep', 'harsh', 'nikung']
```

1.0.14 14) WAP to find the sum of all items (values) in a dictionary given by user. (Assume: values are numeric)

[59]:
```
dict=int(input("enter the dic size/length"))
d = {}


for i in range(dict):
    d[i] = int(input("enter number:"))

print(sum(d.values()))
```

```
3
```

1.0.15 15) WAP to handle missing keys in dictionaries.

Example : Given, dict1 = {'a': 5, 'c': 8, 'e': 2}

if you look for key = 'd', the message given should be 'Key Not Found', otherwise print the value of 'd' in dict1.

[60]:
```
d = { 'a': 5,'c':8,'e':2}

key_list = list(d.keys())

input_users = input("enter the keys")

if input_users not in key_list:
    print("keys not found")
else:
    print(d.get())
```

```
keys not found
```

User Defined Function

01) Write a function to calculate BMI given mass and height. (BMI = mass/h**2)

```
mass = 23
h=2
def bmi():

    ans = (mass/h**2)
    print(ans)
bmi()
```

```
5.75
```

02) Write a function that add first n numbers.

```
def add(n):
    return n * (n + 1) // 2

add(3)
```

```
6
```

03) Write a function that returns 1 if the given number is Prime or 0 otherwise.

```
def is_prime(n):
    if n <= 1:
        return 0
    if n == 2:
        return 1
    if n % 2 == 0:
        return 0


    for i in range(3, int(n ** 0.5) + 1, 2):
        if n % i == 0:
            return 0

    return 1
is_prime(5)
```

```
1
```

04) Write a function that returns the list of Prime numbers between given two numbers.

```
def primes_between(start, end):
    primes = []
    for num in range(start, end + 1):
        if is_prime(num):
            primes.append(num)
    return primes
primes_between(3,7)
```

```
[3, 5, 7]
```

05) Write a function that returns True if the given string is Palindrome or False otherwise.

```
def is_palindrome(s):
    return s == s[::-1]
is_palindrome('lel')
```

```
True
```

06) Write a function that returns the sum of all the elements of the list.

```
def sum_list_elements(lst):
    return sum(lst)
sum_list_elements([1,2,4])
```

```
7
```

07) Write a function to calculate the sum of the first element of each tuples inside the list.

```
def sum_first_elements_of_tuples(tuples_list):
    return sum(t[0] for t in tuples_list)
sum_list_elements((1,2,3,4,5))
```

```
15
```

08) Write a recursive function to find nth term of Fibonacci Series.

```
def fibonacci(n):
    if n <= 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)

fibonacci(4)
```

```
3
```

09) Write a function to get the name of the student based on the given rollno.

Example: Given dict1 = {101:'Ajay', 102:'Rahul', 103:'Jay', 104:'Pooja'} find name of student whose rollno = 103

```
def get_student_name(rollno, student_dict):
    return student_dict.get(rollno, "Roll number not found")

dict1 = {101: 'Ajay', 102: 'Rahul', 103: 'Jay', 104: 'Pooja'}
print(get_student_name(103, dict1))
```

```
Jay
```

10) Write a function to get the sum of the scores ending with zero.

Example : scores = [200, 456, 300, 100, 234, 678]

Ans = 200 + 300 + 100 = 600

```
def sum_scores_ending_with_zero(scores):
    total = 0
    for score in scores:
        if score % 10 == 0:
            total += score
    return total
print(sum_scores_ending_with_zero([10, 20, 33, 40, 55]))
```

```
70
```

11) Write a function to invert a given Dictionary.

hint: keys to values & values to keys

Before : {'a': 10, 'b':20, 'c':30, 'd':40}

After : {10:'a', 20:'b', 30:'c', 40:'d'}

```
def invert_dictionary(d):
    """
    Invert keys and values in a dictionary.

    :param d: Dictionary to invert
    :return: Inverted dictionary
    """
    return {v: k for k, v in d.items()}
invert_dictionary({'a':10,'b':20})
```

```
{10: 'a', 20: 'b'}
```

12 Write a function to check whether the given string is Pangram or not.

hint: Pangram is a string containing all the characters a-z atlest once.

"the quick brown fox jumps over the lazy dog" is a Pangram string.

```
def is_pangram(s):

    return set("abcdefghijklmnopqrstuvwxyz").issubset(s.lower())
is_pangram('the quick brown fox jumps over the lazy dog')
```

```
True
```

13) Write a function that returns the number of uppercase and lowercase letters in the given string.

example : Input : s1 = AbcDEfgh ,Ouptput : no_upper = 3, no_lower = 5

```
def count_upper_lower(s):
    """
    Count the number of uppercase and lowercase letters in a string.

    :param s: Input string
    :return: Tuple with counts of uppercase and lowercase letters
    """
    upper = sum(1 for c in s if c.isupper())
    lower = sum(1 for c in s if c.islower())
    return upper, lower
count_upper_lower('AbcDEfgh')
```

```
(3, 5)
```

14) Write a lambda function to get smallest number from the given two numbers.

```
x=2
y=5
smallest = lambda x, y: x if x < y else y

print(smallest(x,y))
```

```
2
```

15) For the given list of names of students, extract the names having more that 7 characters. Use filter().

```
def filter_long_names(names):

    return list(filter(lambda name: len(name) > 7, names))
filter_long_names(['harry','sejal','marko jansen'])


['marko jansen']
```

16) For the given list of names of students, convert the first letter of all the names into uppercase. use map().

```
def capitalize_names(names):
    return list(map(lambda name: name.capitalize(), names))

capitalize_names(['harry','sejal'])

['Harry', 'Sejal']
```

17) Write udfs to call the functions with following types of arguments:

1. Positional Arguments
2. Keyword Arguments
3. Default Arguments
4. Variable Legngth Positional(*args) & variable length Keyword Arguments (**kwargs)
5. Keyword-Only & Positional Only Arguments

```
def greet_positional(first_name, last_name):
    return f"Hello, {first_name} {last_name}!"

def greet_keyword(first_name="John", last_name="Doe"):
    return f"Hello, {first_name} {last_name}!"

def greet_default(first_name="Jane", last_name="Doe"):
    return f"Hello, {first_name} {last_name}!"

def greet_varargs(*args, **kwargs):
    greeting = "Hello"
    if args:
        greeting += ", " + " ".join(args)
    if kwargs:
        greeting += ", " + ", ".join(f"{key}={value}" for key, value
in kwargs.items())
    return greeting + "!"

def greet_advanced(first_name, middle_name, last_name):
    return f"Hello, {first_name} {middle_name} {last_name}!"
```

```
greet_positional({'harry'},{'sejal'})
greet_keyword()
greet_default()
greet_varargs()
greet_advanced({'harry'},{'sejal'},{'marko'})
```

```
"Hello, {'harry'} {'sejal'} {'marko'}!"
```

sgwc5ndms

March 12, 2025

Python Programming - 2301CS404

Lab - 9

Deep Vamja

Roll no :236

1 File I/O

1.0.1 01) WAP to read and display the contents of a text file. (also try to open the file in some other directory)

- in the form of a string

- line by line

- in the form of a list

[ ]:
```
fp = open("xyz.txt","r")
a = fp.read()
print(a)
fp.close()

fp = open("xyz.txt","r")
li = fp.readline()
print(li)
fp.close()

fp = open("xyz.txt","r")
li = fp.read()
print(li)
fp.close()
```

1.0.2 02) WAP to create file named “new.txt” only if it doesn’t exist.

[ ]:
```
fp = open("file1.txt","w")
s1 = "Hello Students"
fp.write(s1)
fp.close()
```

1.0.3 03) WAP to read first 5 lines from the text file.

[ ]:
```
fp = open("abc.txt","r")
a = fp.read(5)
print(a)
fp.close()
```

```
Ayush
```

1.0.4 04) WAP to find the longest word(s) in a file

[ ]:
```
fp = open("abc.txt","r")
wl = fp.read().split()
w_len = list(map(len,wl))
maxi = max(w_len)
ans = [i for i in wl if len(i) == maxi]
print(ans)
```

```
['University']
```

1.0.5 05) WAP to count the no. of lines, words and characters in a given text file.

[ ]:
```
fp = open("abc.txt","r")
li1 = fp.read()
print(len(li1))
fp.close()
```

```
34
```

[ ]:
```
fp = open("abc.txt","r")
li1 = fp.read()
li1= li1.split()
l=len(li1)
print(l)
fp.close()
```

```
5
```

[ ]:
```
fp = open("abc.txt","r")
li = fp.readlines()
num = len(li)
print(num)
fp.close()
```

```
2
```

1.0.6 06) WAP to copy the content of a file to the another file.

[ ]:
```
with open('abc.txt', 'r') as src:
    content = src.read()

with open('file2.txt', 'w') as dest:
    dest.write(content)

print("File copied successfully.")
```

```
File copied successfully.
```

1.0.7 07) WAP to find the size of the text file.

[ ]:
```
import os
s = os.path.getsize("abc.txt")
print("size of this file : ",s,"B")


word = input("Enter words:")
fp = open("abc.txt","r")
def frequency(f,w):
    c = 0
    for i in f:
        for j in i.split():
            if(j == w):
                c += 1
    return c
print(frequency(fp,word))
fp.close()
```

```
size of this file :  35 B
Enter words:Ayush
1
```

1.0.8 08) WAP to create an UDF named frequency to count occurances of the specific word in a given text file.

[ ]:
```
word = input("enter a word to find:")
count = 0
with open("abc.txt", 'r') as f:
    for line in f:
        words = line.split()
        for i in words:
            if(i==word):
                count=count+1
print("Occurrences of the word", word, ":", count)
```

1.0.9 09) WAP to get the score of five subjects from the user, store them in a file. Fetch those marks and find the highest score.

[ ]:
```python
def get_scores():
    """Get scores for five subjects from the user"""
    scores = []
    subjects = ["Math", "Science", "English", "History", "Computer Science"]

    print("Enter scores for five subjects (0-100):")
    for subject in subjects:
        while True:
            try:
                score = float(input(f"Enter score for {subject}: "))

                if 0 <= score <= 100:
                    scores.append(score)
                    break
                else:
                    print("Score should be between 0 and 100. Try again.")
            except ValueError:
                print("Invalid input. Please enter a number.")

    return scores

def store_scores(scores):
    """Store the scores in a file"""
    with open("scores.txt", "w") as file:
        for score in scores:
            file.write(f"{score}\n")
    print("Scores have been stored in 'scores.txt'")

def fetch_and_find_highest():
    """Fetch scores from the file and find the highest one"""
    try:
        with open("scores.txt", "r") as file:
            scores = [float(line.strip()) for line in file]

        if scores:
            highest_score = max(scores)
            print(f"The scores are: {scores}")
            print(f"The highest score is: {highest_score}")
        else:
            print("No scores found in the file.")
    except FileNotFoundError:
        print("Error: The scores file was not found.")
    except Exception as e:
        print(f"An error occurred: {e}")
```

```
def main():

    scores = get_scores()


    store_scores(scores)

    fetch_and_find_highest()

if __name__ == "__main__":
    main()
```

1.0.10 10) WAP to write first 100 prime numbers to a file named primenumbers.txt

(Note: each number should be in new line)

[ ]:
```
with open("primenumbers.txt", "w") as file:
    count = 0
    num = 2
    while count < 100:
        is_prime = True
        for i in range(2, int(num ** 0.5) + 1):
            if num % i == 0:
                is_prime = False
                break
        if is_prime:
            file.write(str(num) + "\n")
            count += 1
        num += 1
```

1.0.11 11) WAP to merge two files and write it in a new file.

[ ]:
```
data = data2 = "";
with open('abc.txt') as fp:
    data = fp.read()
with open('file2.txt') as fp:
    data2 = fp.read()
data += "\n"
data += data2
with open ('file3.txt', 'w') as fp:
    fp.write(data)
```

1.0.12 12) WAP to replace word1 by word2 of a text file. Write the updated data to new file.

[ ]:
```
data = data2 = "";
with open('abc.txt') as fp:
    data = fp.read()
with open('file2.txt') as fp:
    data2 = fp.read()
data += "\n"
data += data2
with open ('Ayush.txt', 'w') as fp:
    fp.write(data)
```

1.0.13 13) Demonstrate tell() and seek() for all the cases(seek from beginning-end-current position) taking a suitable example of your choice.

[ ]:
```
fp = open("abc.txt","rb")
fp.read(5)
print(fp.tell())
fp.seek(0,2)
fp.seek(-4,1)
print(fp.tell())
fp.close()
```

```
5
31
```

python-programming-lab-10

March 11, 2025

Python Programming - 2301CS404

Lab - 10

Deep Vamja

Roll no :236

1 Exception Handling

1.0.1 01) WAP to handle following exceptions:

1. ZeroDivisionError
2. ValueError
3. TypeError #### Note: handle them using separate except blocks and also using single except block too.

[4]:
```
def handle_exceptions_together():
    try:
        num1 = int(input("Enter number1: "))
        num2 = int(input("Enter enter2: "))
        result = num1 / num2
        print("Result:", result)
    except (ZeroDivisionError, ValueError, TypeError) as e:
        print(f"Error: {e}")

handle_exceptions_together()
```

```
Error: division by zero
```

1.0.2 02) WAP to handle following exceptions:

1. IndexError
2. KeyError

[ ]:
```
try:

    my_list = [1, 2, 3]
    print("Accessing 5th element:", my_list[4])
except IndexError as e:
```

```
    print("IndexError handled:", e)

try:

    my_dict = {"name": "deep", "age": 30}
    print("Accessing 'address':", my_dict["address"])
except KeyError as e:
    print("KeyError handled:", e)
```

```
Result: 0.4
```

1.0.3 03) WAP to handle following exceptions:

1. FileNotFoundError
2. ModuleNotFoundError

[ ]:
```
try:

    with open("non.txt", "r") as file:
        content = file.read()


    import m
except FileNotFoundError as e:
    print("FileNotFoundError handled:", e)
except ModuleNotFoundError as e:
    print("ModuleNotFoundError handled:", e)
```

1.0.4 04) WAP that catches all type of exceptions in a single except block.

[ ]:
```
try:

    my_list = [1, 2, 3]
    print("Accessing 5th element:", my_list[4])
    my_dict = {"name": "ayush", "age": 30}
    print("Accessing 'address':", my_dict["address"])
except Exception as e:
    print("Exception handled:", e)
```

1.0.5 05) WAP to demonstrate else and finally block.

[ ]:
```
try:

    num1 = int(input("Enter num1: "))
    num2 = int(input("Enter mum2: "))
    result = num1 / num2
except ZeroDivisionError as e:
```

```
    print("ZeroDivisionError handled:", e)
except ValueError as e:
    print("ValueError handled:", e)
else:
    print("Division successful! Result:", result)
finally:
    print("Execution completed, finally block executed.")
```

1.0.6 06) Create a short program that prompts the user for a list of grades separated by commas.

1.0.7 Split the string into individual grades and use a list comprehension to convert each string to an integer.

1.0.8 You should use a try statement to inform the user when the values they entered cannot be converted.

[ ]:
```
try:

    grades_input = input("Enter a grades ")
    grades = [int(grade.strip()) for grade in grades_input.split(",")]
    print("Grades entered:", grades)
except ValueError as e:
    print("ValueError handled: Please enter only numbers separated by commas.",␣
 ↪e)
finally:
    print("Execution completed, finally block executed.")
```

1.0.9 07) WAP to create an udf divide(a,b) that handles ZeroDivisionError.

[ ]:
```
def handle_exceptions_together():
    try:
        a = int(input("Enter number1: "))
        b = int(input("Enter number2: "))
        result = a / b
        print("Result:", result)
    except (ZeroDivisionError) as e:
        print(f"Error: {e}")

handle_exceptions_together()
```

1.0.10 08) WAP that gets an age of a person form the user and raises ValueError with error message: "Enter Valid Age" :

If the age is less than 18.

otherwise print the age.

[ ]:
```
try:

    age = int(input("Enter your age: "))
    if age < 18:
        raise ValueError("Enter Valid Age")
    print("Your age is:", age)
except ValueError as e:
    print("ValueError handled:", e)
finally:
    print("Execution completed")
```

1.0.11 09) WAP to raise your custom Exception named InvalidUsernameError with the error message : "Username must be between 5 and 15 characters long":

if the given name is having characters less than 5 or greater than 15.

otherwise print the given username.

[12]:
```
class InvalidUsernameError(Exception):
    pass

def validate_username(username):
    if len(username) < 5 or len(username) > 15:
        raise InvalidUsernameError("Username must be between 5 and 15␣
 ↪characters long")
    else:
        print(f"Given username is: {username}")


username = input("Enter your username: ")

try:
    validate_username(username)
except InvalidUsernameError as e:
    print(e)
```

```
Username must be between 5 and 15 characters long
```

1.0.12 10) WAP to raise your custom Exception named NegativeNumberError with the error message : "Cannot calculate the square root of a negative number" :

if the given number is negative.

otherwise print the square root of the given number.

[15]:
```
class NegativeNumberError(Exception):
    pass
```

```
def validate_num(num):
    if num < 0:
        raise NegativeNumberError("Cannot be calculated the square root of a␣
 ↪negative number")
    else:
        print(f"Given num is: {num}")


names = int(input("Enter your num: "))

try:
    validate_num(names)
except NegativeNumberError as e:
    print(e)
```

```
Cannot be calculated the square root of a negative number
```

[ ]:

python-programming-lab-11

March 11, 2025

Python Programming - 2301CS404

Lab - 11

Deep Vamja

Roll no :236

1 Modules

1.0.1 01) WAP to create Calculator module which defines functions like add, sub,mul and div.

1.0.2 Create another .py file that uses the functions available in Calculator module.

[1]:
```
import cal

print(cal.add(5, 3))
print(cal.sub(10, 4))
print(cal.mul(6, 7))
print(cal.div(20, 5))
```

```
8
6
42
4.0
```

1.0.3 02) WAP to pick a random character from a given String.

[11]:
```
import random;
S  = 'deep vamja'
print(random.choice(S))
```

```
a
```

1.0.4 03) WAP to pick a random element from a given list.

[12]:
```
import random;
l1=[1,45,2,6,7,34,2]
print(random.choices(l1))
```

```
[2]
```

1.0.5 04) WAP to roll a dice in such a way that every time you get the same number.

[ ]:
```
random.seed(5)
def roll_dice():
    return random.randint(1, 6)
print(roll_dice())
```

```
5
```

1.0.6 05) WAP to generate 3 random integers between 100 and 999 which is divisible by 5.

[34]:
```
nums = [random.choice(range(100, 1000, 5)) for nums in range(3)]
print(nums)
```

```
[425, 555, 980]
```

1.0.7 06) WAP to generate 100 random lottery tickets and pick two lucky tickets from it and announce them as Winner and Runner up respectively.

[49]:
```
lottery_tickets = list(range(1000, 1100))
winners = random.sample(lottery_tickets, 2)
print("Winner:", winners[0], "Runner-up:", winners[1])
```

```
Winner: 1059 Runner-up: 1099
```

1.0.8 07) WAP to print current date and time in Python.

[50]:
```
from datetime import datetime
print(datetime.now())
```

```
2025-02-10 12:55:24.932643
```

1.0.9 08) Subtract a week (7 days) from a given date in Python.

[51]:
```
from datetime import timedelta
date_today = datetime.now()
new_date = date_today - timedelta(days=7)
print(new_date)
```

```
2025-02-03 12:55:24.939031
```

1.0.10 09) WAP to Calculate number of days between two given dates.

[52]:
```
date1 = datetime(2024, 2, 1)
date2 = datetime(2024, 2, 10)
difference = (date2 - date1).days
print("Days gap between two given date:", difference)
```

```
Days gap between two given date: 9
```

1.0.11 10) WAP to Find the day of the week of a given date.(i.e. wether it is sunday/monday/tuesday/etc.)

[53]:
```
given_date = datetime(2024, 2, 10)
print("Day of the given  week is :", given_date.strftime("%A"))
```

```
Day of the given  week is : Saturday
```

1.0.12 11) WAP to demonstrate the use of date time module.

[54]:
```
from datetime import date

today = date.today()

print("Current year:", today.year)
print("Current month:", today.month)
print("Current day:", today.day)
```

```
Current year: 2025
Current month: 2
Current day: 10
```

1.0.13 12) WAP to demonstrate the use of the math module.

[55]:
```
import math
print("Square root of 25:", math.sqrt(25))
print("Factorial of 5:", math.factorial(5))
print("Ceiling of 4.3:", math.ceil(4.3))
print("Floor of 4.8:", math.floor(4.8))
```

```
Square root of 25: 5.0
Factorial of 5: 120
Ceiling of 4.3: 5
Floor of 4.8: 4
```

python-programming-lab-12

March 11, 2025

Python Programming - 2301CS404

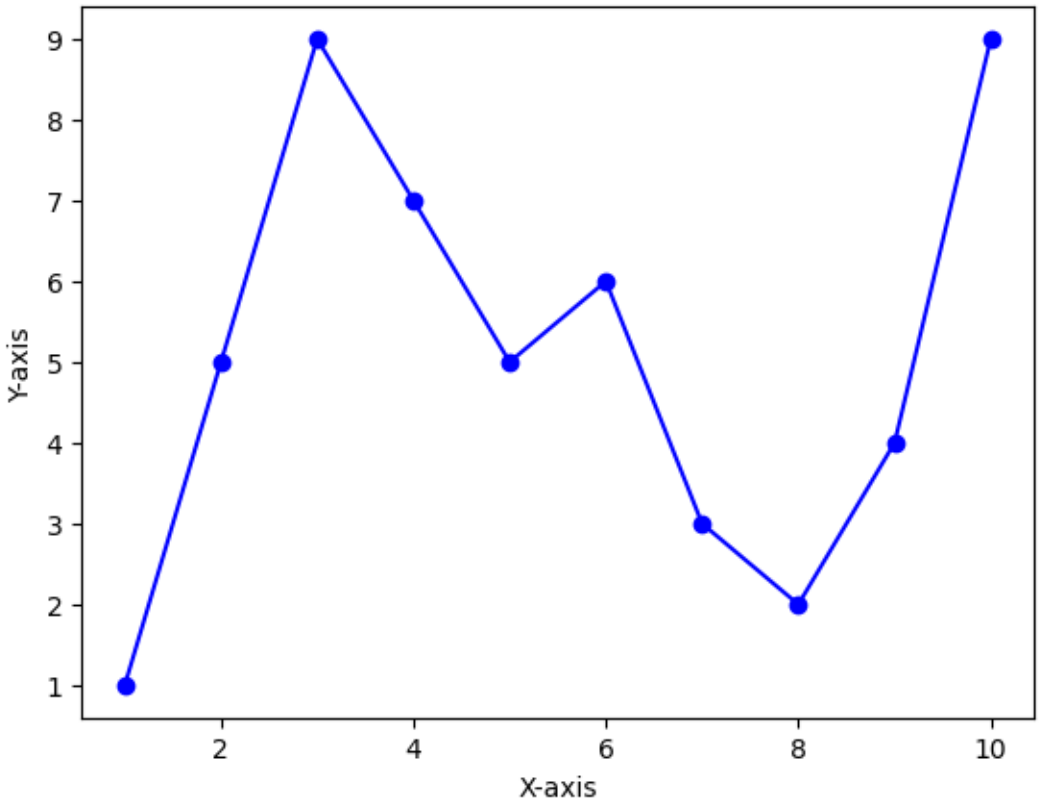
Lab - 12

Deep Vamja

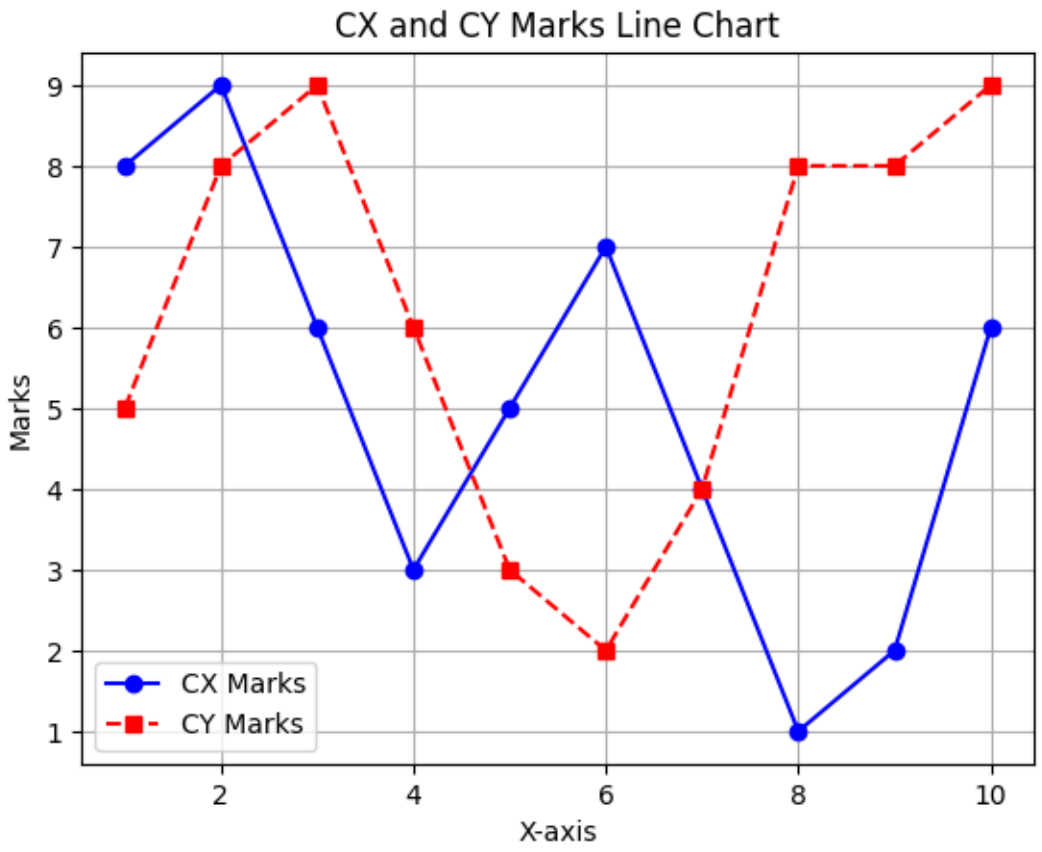
Roll no :236

[59]:
```
#import matplotlib below
import matplotlib.pyplot as plt
```

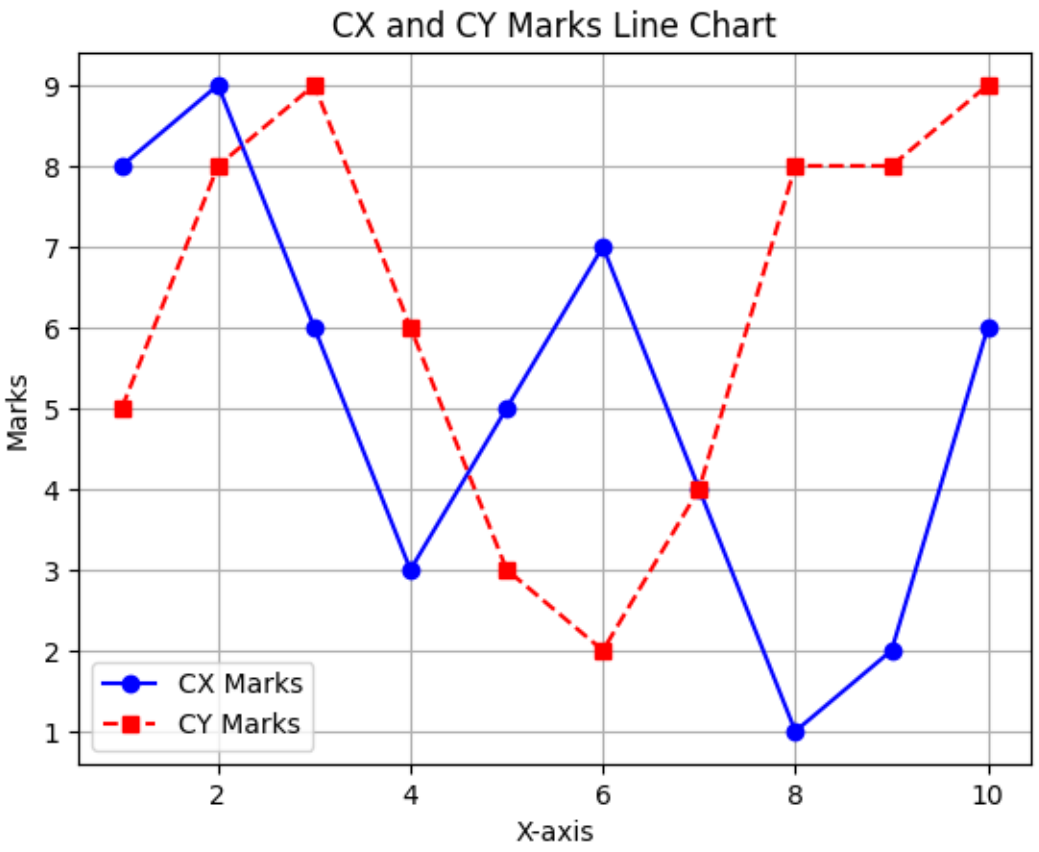
[60]:
```
# write a code to display the line chart of above x & y
import matplotlib.pyplot as plt

x = range(1, 11)
y = [1, 5, 9, 7, 5, 6, 3, 2, 4, 9]


plt.plot(x, y, marker='o', linestyle='-', color='b')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')



plt.show()
```



[ ]:
```
import matplotlib.pyplot as plt


x = range(1, 11, 1)
cxMarks = [8, 9, 6, 3, 5, 7, 4, 1, 2, 6]
cyMarks = [5, 8, 9, 6, 3, 2, 4, 8, 8, 9]


plt.plot(x, cxMarks, marker='o', linestyle='-', color='b', label='CX Marks')
plt.plot(x, cyMarks, marker='s', linestyle='--', color='r', label='CY Marks')
plt.xlabel('X-axis')
plt.ylabel('Marks')
plt.title('CX and CY Marks Line Chart')
plt.legend()
plt.grid(True)


plt.show()
```



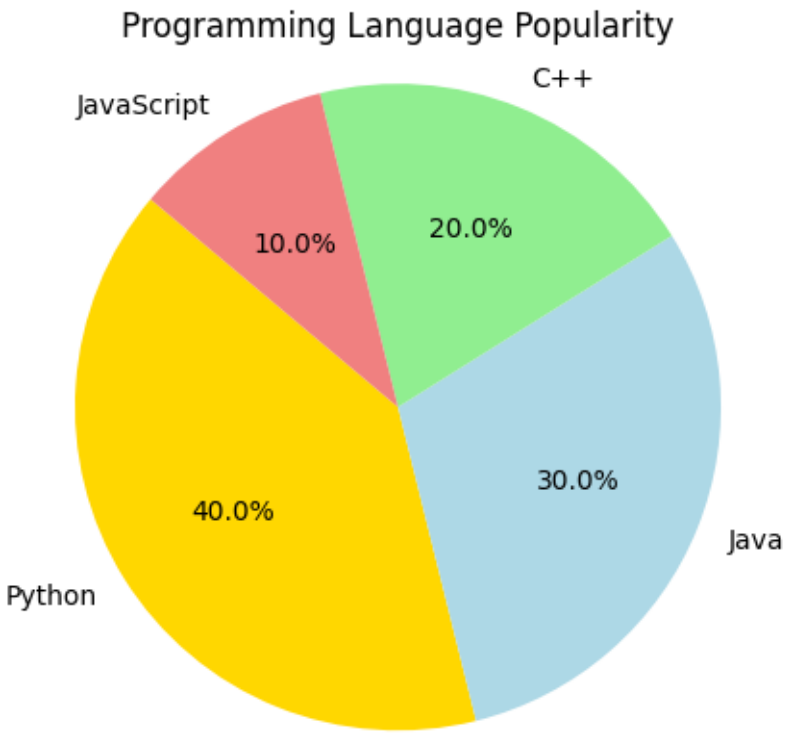
[66]:
```
import matplotlib.pyplot as plt


x = range(1, 11, 1)
cxMarks = [8, 9, 6, 3, 5, 7, 4, 1, 2, 6]
cyMarks = [5, 8, 9, 6, 3, 2, 4, 8, 8, 9]

plt.plot(x, cxMarks, marker='o', linestyle='-', color='b', label='CX Marks')
plt.plot(x, cyMarks, marker='s', linestyle='--', color='r', label='CY Marks')
plt.xlabel('X-axis')
plt.ylabel('Marks')
plt.title('CX and CY Marks Line Chart')
plt.legend()
plt.grid(True)


plt.show()
```



[ ]:

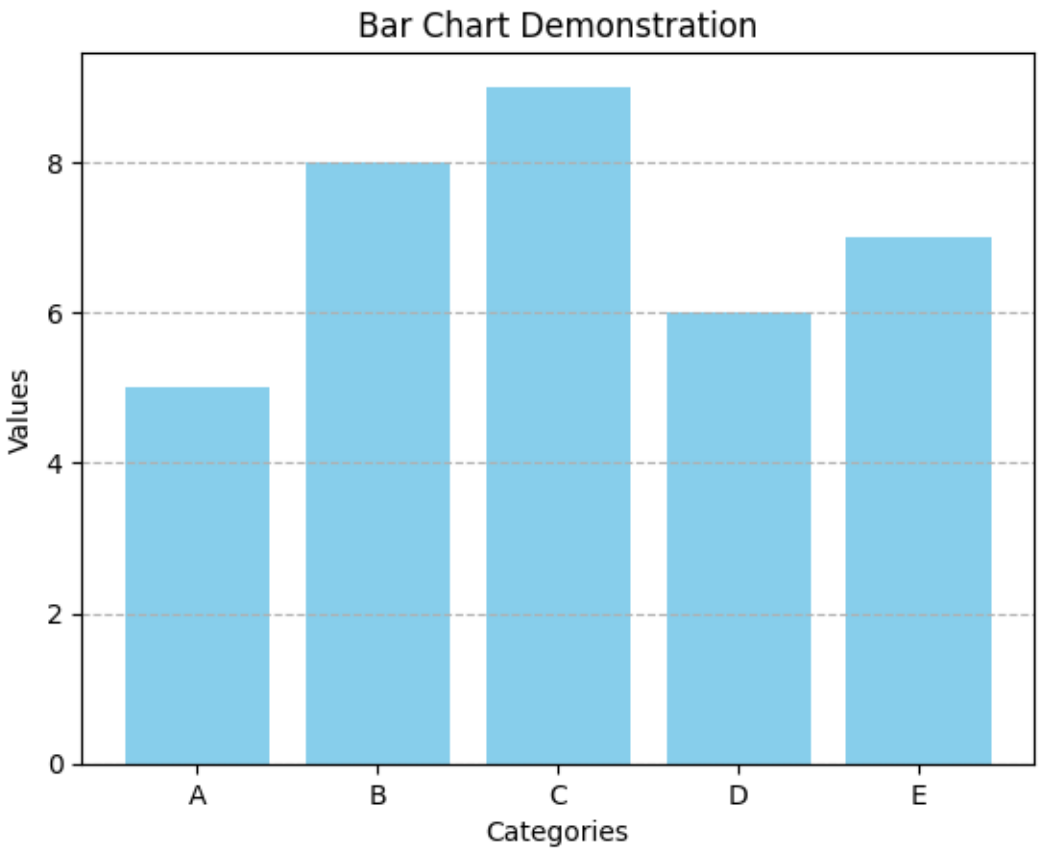
0.0.1 04) WAP to demonstrate the use of Pie chart.

[57]:
```
import matplotlib.pyplot as plt

labels = ['Python', 'Java', 'C++', 'JavaScript']
sizes = [40, 30, 20, 10]
colors = ['gold', 'lightblue', 'lightgreen', 'lightcoral']



plt.pie(sizes, labels=labels, colors=colors,  autopct='%1.1f%%', ␣
 ↪startangle=140)
plt.axis('equal')
plt.title('Programming Language Popularity')

plt.show()
```



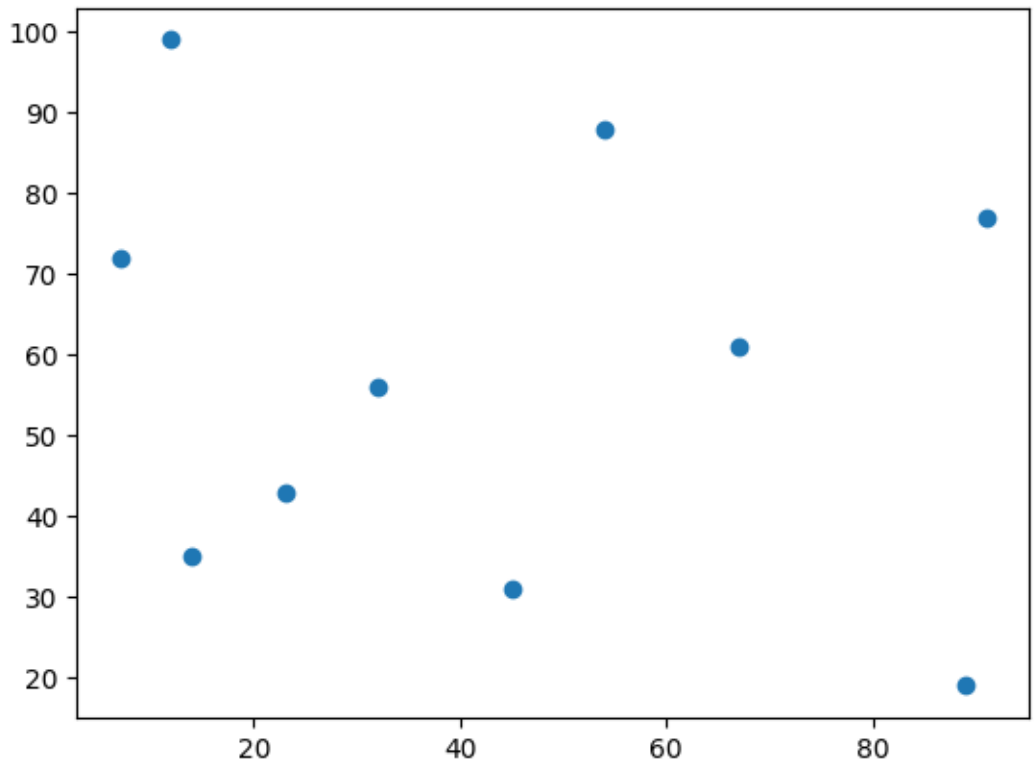
0.0.2 05) WAP to demonstrate the use of Bar chart.

[58]:
```
import matplotlib.pyplot as plt


categories = ['A', 'B', 'C', 'D', 'E']
values = [5, 8, 9, 6, 7]


plt.bar(categories, values, color='skyblue')
plt.xlabel('Categories')
plt.ylabel('Values')
plt.title('Bar Chart Demonstration')
plt.grid(axis='y', linestyle='--')

plt.show()
```



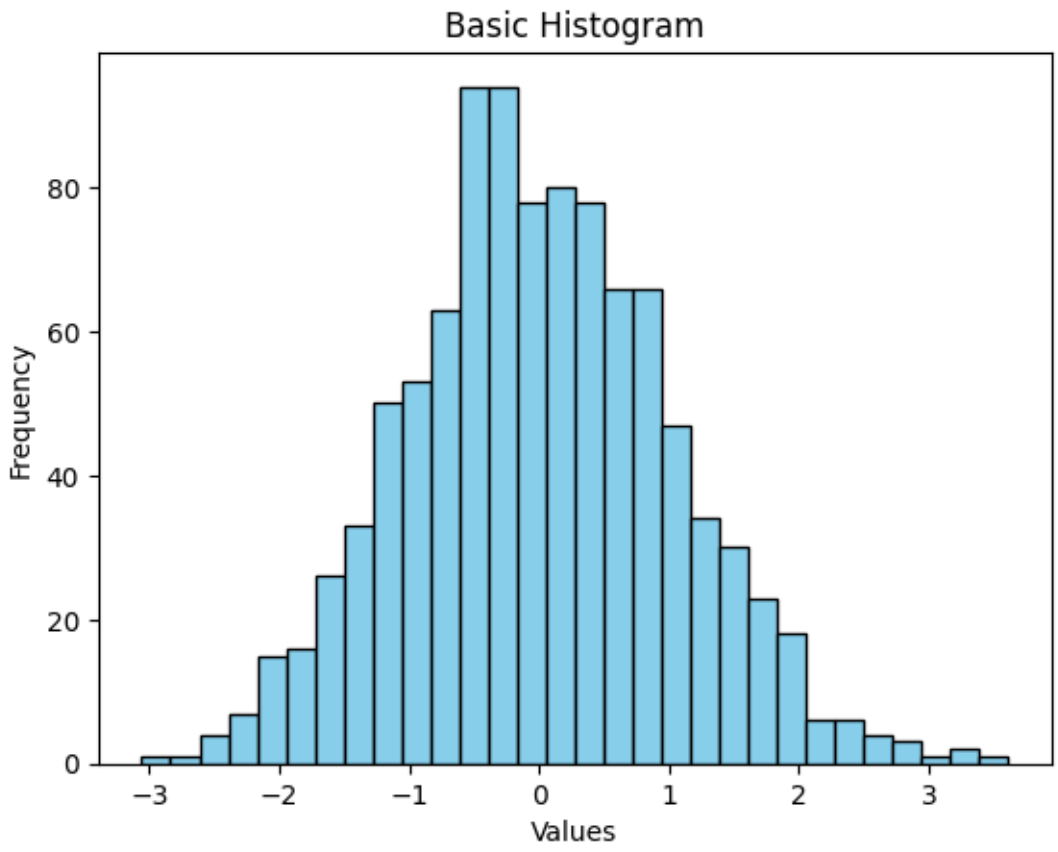
0.0.3 06) WAP to demonstrate the use of Scatter Plot.

[59]:
```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([12, 45, 7, 32, 89, 54, 23, 67, 14, 91])
y = np.array([99, 31, 72, 56, 19, 88, 43, 61, 35, 77])

plt.scatter(x, y)
plt.show()
```



0.0.4 07) WAP to demonstrate the use of Histogram.

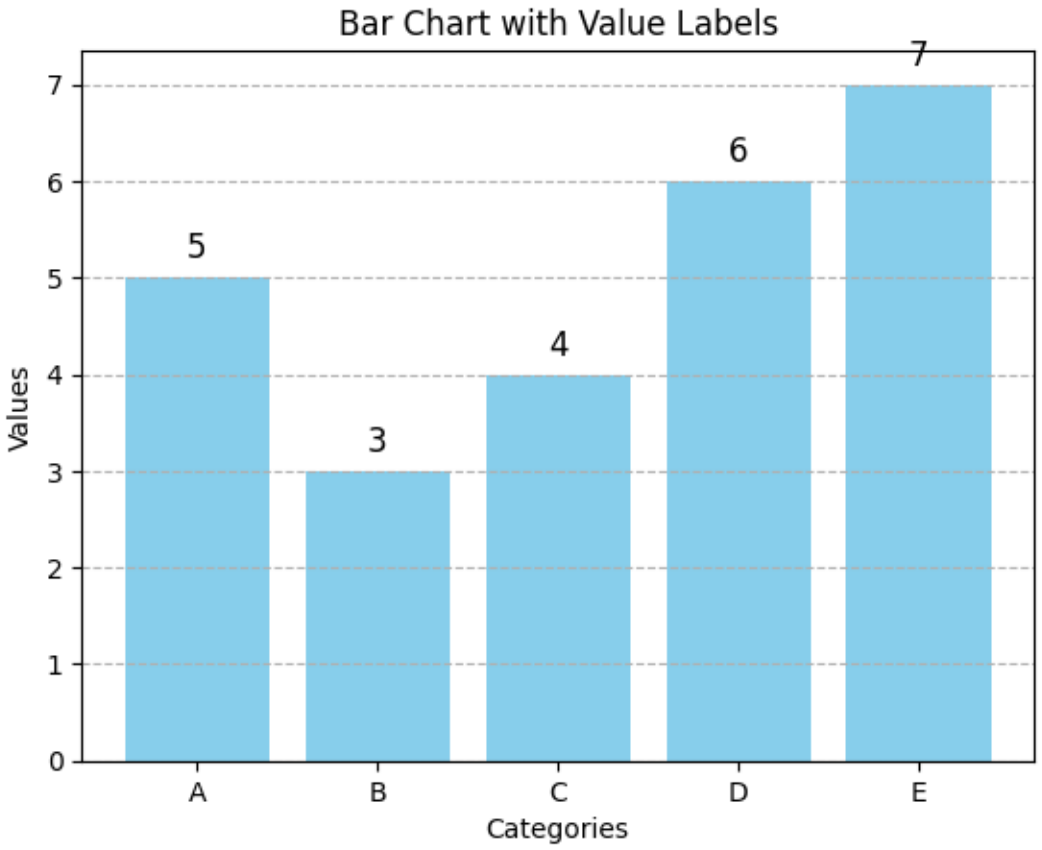
[60]:
```
import matplotlib.pyplot as plt
import numpy as np


data = np.random.randn(1000)

plt.hist(data, bins=30, color='skyblue', edgecolor='black')


plt.xlabel('Values')
plt.ylabel('Frequency')
plt.title('Basic Histogram')


plt.show()
```



0.0.5 08) WAP to display the value of each bar in a bar chart using Matplotlib.

[61]:
```
import matplotlib.pyplot as plt


categories = ['A', 'B', 'C', 'D', 'E']
values = [5, 3, 4, 6, 7]


plt.bar(categories, values, color='skyblue')
plt.xlabel('Categories')
plt.ylabel('Values')
plt.title('Bar Chart with Value Labels')
plt.grid(axis='y', linestyle='--')


for i in range(len(values)):
    plt.text(i, values[i] + 0.2, str(values[i]), ha='center', fontsize=12)
```

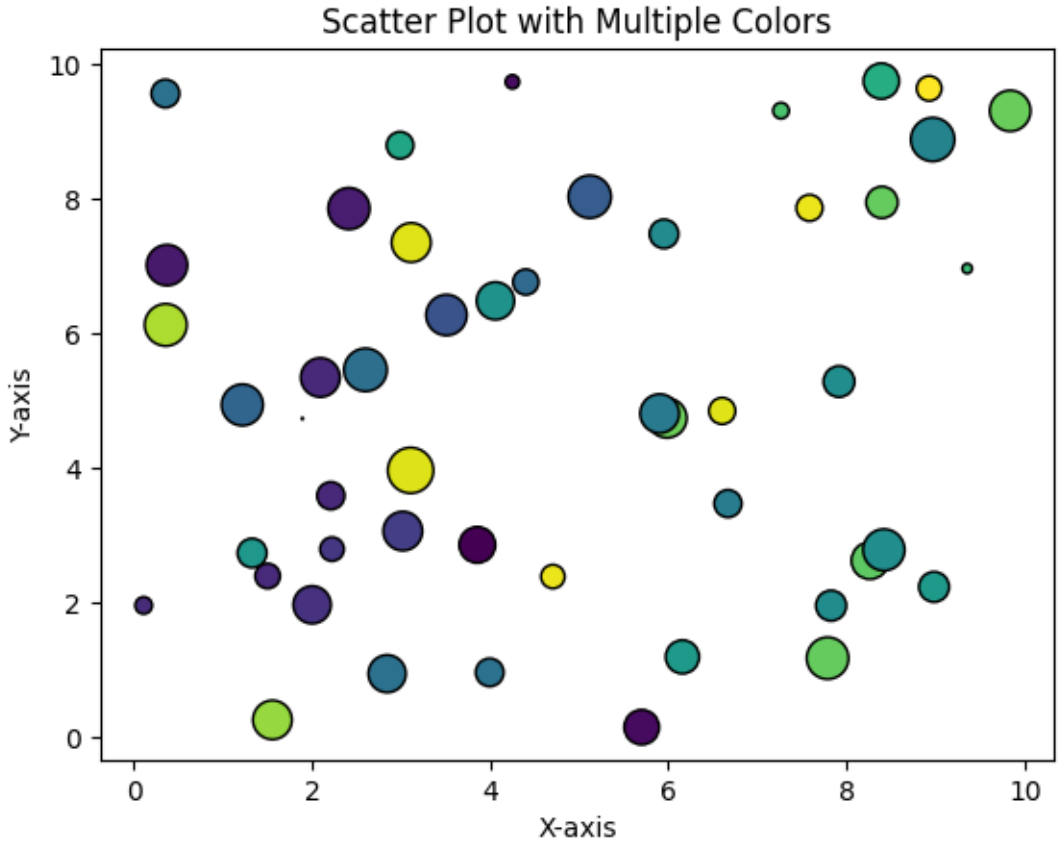
```
plt.show()
```



0.0.6 09) WAP create a Scatter Plot with several colors in Matplotlib?

[62]:
```
import matplotlib.pyplot as plt
import numpy as np


x = np.random.rand(50) * 10
y = np.random.rand(50) * 10
colors = np.random.rand(50)
sizes = np.random.rand(50) * 300


plt.scatter(x, y, c=colors, s=sizes,  edgecolors='black')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
```

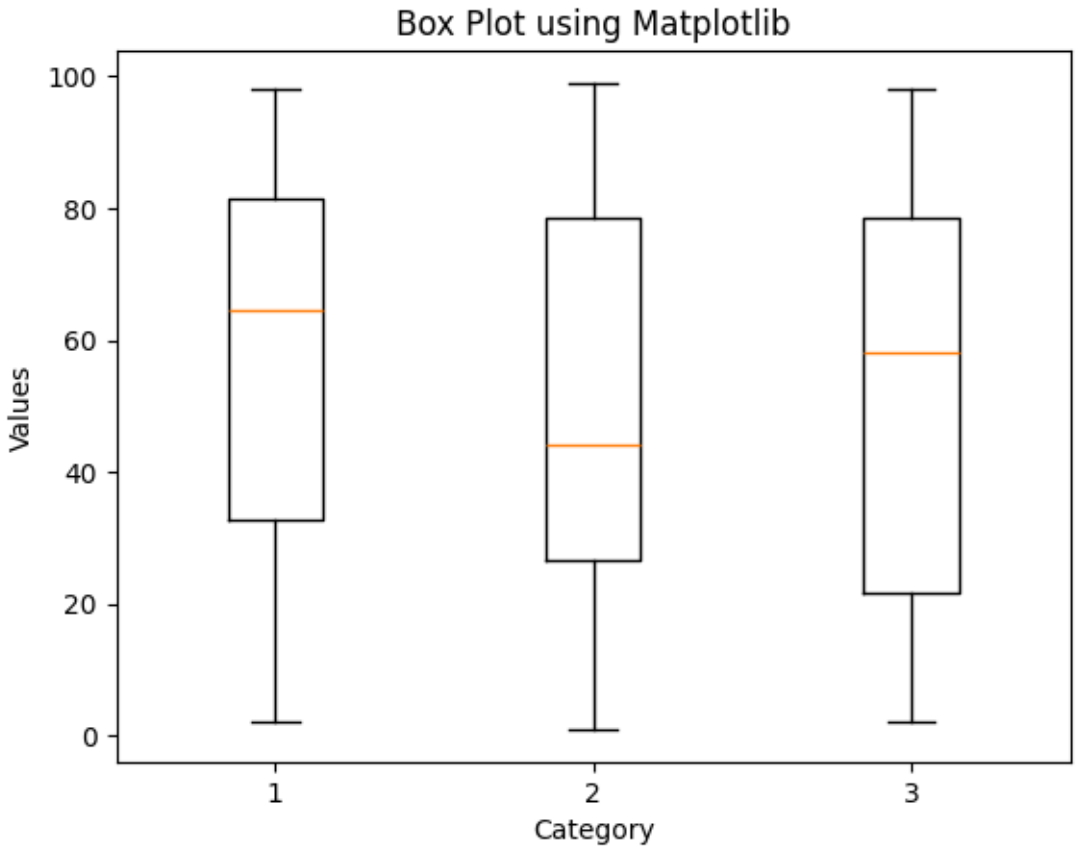
```
plt.title('Scatter Plot with Multiple Colors')



# Show the chart
plt.show()
```



0.0.7 10) WAP to create a Box Plot.

[63]:
```
import matplotlib.pyplot as plt
import numpy as np

# Sample data using random integers
data = [np.random.randint(1, 100, 100) for i in range(3)]

# Create the box plot
plt.boxplot(data)

# Add labels
```

```
plt.title('Box Plot using Matplotlib')
plt.xlabel('Category')
plt.ylabel('Values')

# Show the plot
plt.show()
```



OOP

01) Write a Program to create a class by name Students, and initialize attributes like name, age, and grade while creating an object.

```
class Students:
    def __init__(self, name, age, grade):
        self.name = name
        self.age = age
        self.grade = grade

    def display_info(self):
        print(f"Student Name: {self.name}")
        print(f"Age: {self.age}")
        print(f"Grade: {self.grade}")


student1 = Students("deep", 19, "12th Grade")

student1.display_info()
```

```
Student Name: deep
Age: 19
Grade: 12th Grade
```

02) Create a class named Bank_Account with Account_No, User_Name, Email,Account_Type and Account_Balance data members. Also create a method GetAccountDetails() and DisplayAccountDetails(). Create main method to demonstrate the Bank_Account class.

```
class Bank_Account:
    def __init__(self, account_no, user_name, email, account_type,
account_balance):
        self.account_no = account_no
        self.user_name = user_name
        self.email = email
        self.account_type = account_type
        self.account_balance = account_balance
```

```
    def GetAccountDetails(self):
        self.account_no = input("Enter Account Number: ")
        self.user_name = input("Enter User Name: ")
        self.email = input("Enter Email: ")
        self.account_type = input("Enter Account Type
(Savings/Current): ")
        self.account_balance = float(input("Enter Account Balance: "))

    def DisplayAccountDetails(self):
        print("\n----- Account Details -----")
        print(f"Account Number: {self.account_no}")
        print(f"User Name: {self.user_name}")
        print(f"Email: {self.email}")
        print(f"Account Type: {self.account_type}")
        print(f"Account Balance: ${self.account_balance:.2f}")

userdetails =
Bank_Account(123,'deep','deep@gmail.com','saving','34000')
userdetails.DisplayAccountDetails


def main():

    account = Bank_Account("", "", "", "", 0.0)


    account.GetAccountDetails()


    account.DisplayAccountDetails()

if __name__ == "__main__":
    main()


----- Account Details -----
Account Number: 123
User Name: deep
Email: deep@gmail.com
Account Type: saving
Account Balance: $34000.00
```

03) WAP to create Circle class with area and perimeter function to find area and perimeter of circle.

```
import math

class Circle:
    def __init__(self, radius):
        self.radius = radius
```

```
    def area(self):
        return math.pi * self.radius ** 2

    def perimeter(self):
        return 2 * math.pi * self.radius

def main():
    r = float(input("Enter the radius of the circle: "))
    circle = Circle(r)

    print(f"\nArea of Circle: {circle.area():.2f}")
    print(f"Perimeter of Circle: {circle.perimeter():.2f}")


if __name__ == "__main__":
    main()
```

```
Area of Circle: 50.27
Perimeter of Circle: 25.13
```

04) Create a class for employees that includes attributes such as name, age, salary, and methods to update and display employee information.

```
class Employee:
    def __init__(self, name, age, salary):
        self.name = name
        self.age = age
        self.salary = salary

    def update_info(self, name=None, age=None, salary=None):
        if name:
            self.name = name
        if age:
            self.age = age
        if salary:
            self.salary = salary
        print("\nEmployee details updated successfully!")

    def display_info(self):
        print("\n----- Employee Details -----")
        print(f"Name: {self.name}")
        print(f"Age: {self.age}")
        print(f"Salary: ${self.salary:.2f}")


def main():
```

```
    emp1 = Employee("deep", 19, 30000)

    emp1.display_info()

    emp1.update_info(age=20, salary=45000)


    emp1.display_info()

# Calling the main function
if __name__ == "__main__":
    main()
```

```
----- Employee Details -----
Name: deep
Age: 19
Salary: $30000.00

Employee details updated successfully!

----- Employee Details -----
Name: deep
Age: 20
Salary: $45000.00
```

05) Create a bank account class with methods to deposit, withdraw, and check balance.

```
class BankAccount:
    def __init__(self, account_number, account_holder, balance=0.0):
        self.account_number = account_number
        self.account_holder = account_holder
        self.balance = balance

    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
            print(f"\n${amount:.2f} deposited successfully!")
        else:
            print("\nInvalid deposit amount!")

    def withdraw(self, amount):
        if 0 < amount <= self.balance:
            self.balance -= amount
            print(f"\n${amount:.2f} withdrawn successfully!")
        elif amount > self.balance:
            print("\nInsufficient balance!")
        else:
            print("\nInvalid withdrawal amount!")
```

```
    def check_balance(self):
        print(f"\nCurrent Balance: ${self.balance:.2f}")

def main():
    account = BankAccount("123456789", "Deep", 2000.0)

    account.check_balance()

    account.deposit(5000)

    account.check_balance()

    account.withdraw(3000)
    account.check_balance()

    account.withdraw(2000)
if __name__ == "__main__":
    main()
```

```
Current Balance: $2000.00
$5000.00 deposited successfully!
Current Balance: $7000.00
$3000.00 withdrawn successfully!
Current Balance: $4000.00
$2000.00 withdrawn successfully!
```

06) Create a class for managing inventory that includes attributes such as item name, price, quantity, and methods to add, remove, and update items.

```
class Inventory:
    def __init__(self):
        self.items = {}

    def add_item(self, name, price, quantity):
```

```
        if name in self.items:
            self.items[name]['quantity'] += quantity
        else:
            self.items[name] = {'price': price, 'quantity': quantity}
        print(f"\n{name} added successfully! Current quantity:
{self.items[name]['quantity']}")

    def remove_item(self, name, quantity):
        if name in self.items:
            if self.items[name]['quantity'] >= quantity:
                self.items[name]['quantity'] -= quantity
                print(f"\n{quantity} units of {name} removed!")
                if self.items[name]['quantity'] == 0:
                    del self.items[name]
                    print(f"{name} is now out of stock and removed
from inventory.")
            else:
                print(f"\nInsufficient quantity of {name} to remove!")
        else:
            print(f"\n{name} not found in inventory!")

    def update_price(self, name, new_price):
        if name in self.items:
            self.items[name]['price'] = new_price
            print(f"\nPrice of {name} updated to ${new_price:.2f}!")
        else:
            print(f"\n{name} not found in inventory!")

    def display_inventory(self):
        if not self.items:
            print("\nInventory is empty!")
        else:
            print("\n----- Inventory Details -----")
            for name, details in self.items.items():
                print(f"Item: {name}, Price: ${details['price']:.2f},
Quantity: {details['quantity']}")


def main():
    store = Inventory()


    store.add_item("Laptop", 1000, 5)
    store.add_item("Mouse", 20, 10)
    store.add_item("Keyboard", 50, 7)

    store.display_inventory()

    store.remove_item("Mouse", 5)
```

```
    store.update_price("Laptop", 950)


    store.display_inventory()


if __name__ == "__main__":
    main()
```

```
Laptop added successfully! Current quantity: 5

Mouse added successfully! Current quantity: 10

Keyboard added successfully! Current quantity: 7

----- Inventory Details -----
Item: Laptop, Price: $1000.00, Quantity: 5
Item: Mouse, Price: $20.00, Quantity: 10
Item: Keyboard, Price: $50.00, Quantity: 7

5 units of Mouse removed!

Price of Laptop updated to $950.00!

----- Inventory Details -----
Item: Laptop, Price: $950.00, Quantity: 5
Item: Mouse, Price: $20.00, Quantity: 5
Item: Keyboard, Price: $50.00, Quantity: 7
```

07) Create a Class with instance attributes of your choice.

```
class Car:
    def __init__(self, brand, model):
        self.brand = brand
        self.model = model


    def display_details(self):
        print("\n----- Car Details -----")
        print(f"Brand: {self.brand}")
        print(f"Model: {self.model}")


def main():

    car1 = Car("Tesla", "Model S")
    car2 = Car("Toyota", "Camry")


    car1.display_details()
```

```
    car2.display_details()

# Calling the main function
if __name__ == "__main__":
    main()
```

```
----- Car Details -----
Brand: Tesla
Model: Model S

----- Car Details -----
Brand: Toyota
Model: Camry
```

08) Create one class student_kit

Within the student_kit class create one class attribute principal name (Mr ABC)

Create one attendance method and take input as number of days.

While creating student take input their name .

Create one certificate for each student by taking input of number of days present in class.

```
class StudentKit:

    principal_name = "Mr. ABC"

    def __init__(self, student_name):
        self.student_name = student_name
        self.attendance_days = 0

    def attendance(self, days):
        self.attendance_days = days
        print(f"\nAttendance recorded for {self.student_name}:
{self.attendance_days} days")

    def generate_certificate(self):
        print("\n-------------------- CERTIFICATE
--------------------")
        print(f"Principal: {StudentKit.principal_name}")
        print(f"Student Name: {self.student_name}")
        print(f"Total Attendance: {self.attendance_days} days")
        print("Congratulations on your participation!")

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

def main():
```

```
    student_name = input("Enter student name: ")

    student = StudentKit(student_name)


    days_present = int(input(f"Enter number of days {student_name} was
present: "))

    student.attendance(days_present)


    student.generate_certificate()

# Calling the main function
if __name__ == "__main__":
    main()
```

```
  Cell In[19], line 4
    def atten
             ^
SyntaxError: expected '('
```

09) Define Time class with hour and minute as data member. Also define addition method to add two time objects.

```
class Time:
    def __init__(self, hour, minute):
        self.hour = hour
        self.minute = minute

    def add_time(self, other):
        total_minutes = self.minute + other.minute
        extra_hours = total_minutes // 60
        final_minutes = total_minutes % 60
        final_hours = self.hour + other.hour + extra_hours

        return Time(final_hours, final_minutes)

    def display(self):
        print(f"{self.hour} hour(s) and {self.minute} minute(s)")
s
def main():

    time1 = Time(2, 45)
    time2 = Time(1, 30)


    print("\nTime 1:")
```

```
    time1.display()

    print("\nTime 2:")
    time2.display()

    total_time = time1.add_time(time2)


    print("\nTotal Time after addition:")
    total_time.display()

if __name__ == "__main__":
    main()
```

Continued..

10) Calculate area of a ractangle using object as an argument to a method.

```
class Rectangle:
    def __init__(self, length, width):
        self.length = length
        self.width = width

def calculate_area(rect):
    return rect.length * rect.width


rect = Rectangle(5, 3)
area = calculate_area(rect)
print(f"Area of rectangle: {area}")
```

11) Calculate the area of a square.

Include a Constructor, a method to calculate area named area() and a method named output() that prints the output and is invoked by area().

```
class Square:
    def __init__(self, side):
        self.side = side

    def area(self):
        area_value = self.side ** 2
        self.output(area_value)
        return area_value

    def output(self, area_value):
        print(f"Area of square with side {self.side} is:
{area_value}")


square = Square(4)
square.area()
```

12) Calculate the area of a rectangle.

Include a Constructor, a method to calculate area named area() and a method named output() that prints the output and is invoked by area().

Also define a class method that compares the two sides of reactangle. An object is instantiated only if the two sides are different; otherwise a message should be displayed : THIS IS SQUARE.

```
class Rectangle:
    def __init__(self, length, width):
        self.length = length
        self.width = width

    def area(self):
        area_value = self.length * self.width
        self.output(area_value)
        return area_value

    def output(self, area_value):
        print(f"Area of rectangle with length {self.length} and width
{self.width} is: {area_value}")

    @classmethod
    def create_rectangle(cls, length, width):
        if length == width:
            print("THIS IS SQUARE.")
            return None
        else:
            return cls(length, width)


rect1 = Rectangle.create_rectangle(5, 3)
if rect1:
    rect1.area()

rect2 = Rectangle.create_rectangle(4, 4)
```

13) Define a class Square having a private attribute "side".

Implement get_side and set_side methods to accees the private attribute from outside of the class.

```
class Square:
    def __init__(self, side):
        self.__side = side

    def get_side(self):
```

```
        return self.__side

    def set_side(self, side):
        self.__side = side

    def calculate_area(self):
        return self.__side ** 2


sq = Square(5)
print(f"Side of square: {sq.get_side()}")
print(f"Area of square: {sq.calculate_area()}")

sq.set_side(7)
print(f"New side of square: {sq.get_side()}")
print(f"New area of square: {sq.calculate_area()}")
```

14) Create a class Profit that has a method named getProfit that accepts profit from the user.

Create a class Loss that has a method named getLoss that accepts loss from the user.

Create a class BalanceSheet that inherits from both classes Profit and Loss and calculates the balanace. It has two methods getBalance() and printBalance().

```
class Profit:
    def __init__(self):
        self.profit = 0

    def getProfit(self):
        self.profit = float(input("Enter profit amount: "))
        return self.profit

class Loss:
    def __init__(self):
        self.loss = 0

    def getLoss(self):
        self.loss = float(input("Enter loss amount: "))
        return self.loss

class BalanceSheet(Profit, Loss):
    def __init__(self):
        Profit.__init__(self)
        Loss.__init__(self)
        self.balance = 0
```

```
    def getBalance(self):
        self.balance = self.profit - self.loss
        return self.balance

    def printBalance(self):
        print(f"Profit: ${self.profit}")
        print(f"Loss: ${self.loss}")
        print(f"Balance: ${self.balance}")

# Uncomment to test
# balance_sheet = BalanceSheet()
# balance_sheet.getProfit()
# balance_sheet.getLoss()
# balance_sheet.getBalance()
# balance_sheet.printBalance()
```

15) WAP to demonstrate all types of inheritance.

```
class Parent:
    def __init__(self):
        self.parent_attribute = "This is from parent"

    def parent_method(self):
        print("This is parent method")

class Child(Parent):
    def __init__(self):
        super().__init__()
        self.child_attribute = "This is from child"

    def child_method(self):
        print("This is child method")

class Father:
    def father_method(self):
        print("This is father method")

class Mother:
    def mother_method(self):
        print("This is mother method")

class Child2(Father, Mother):
    def child_method(self):
        print("This is child2 method")

class Grandparent:
    def grandparent_method(self):
        print("This is grandparent method")
```

```
class Parent2(Grandparent):
    def parent_method(self):
        print("This is parent2 method")

class Child3(Parent2):
    def child_method(self):
        print("This is child3 method")

class Parent3:
    def parent_method(self):
        print("This is parent3 method")

class ChildA(Parent3):
    def child_a_method(self):
        print("This is childA method")

class ChildB(Parent3):
    def child_b_method(self):
        print("This is childB method")


class Base:
    def base_method(self):
        print("This is base method")

class Derived1(Base):
    def derived1_method(self):
        print("This is derived1 method")

class Derived2(Base):
    def derived2_method(self):
        print("This is derived2 method")

class DerivedOfDerived(Derived1, Derived2):
    def derived_of_derived_method(self):
        print("This is derived of derived method")


print("\nSingle Inheritance:")
child = Child()
child.parent_method()
child.child_method()

print("\nMultiple Inheritance:")
child2 = Child2()
child2.father_method()
child2.mother_method()
child2.child_method()

print("\nMultilevel Inheritance:")
```

```
child3 = Child3()
child3.grandparent_method()
child3.parent_method()
child3.child_method()

print("\nHierarchical Inheritance:")
childA = ChildA()
childB = ChildB()
childA.parent_method()
childA.child_a_method()
childB.parent_method()
childB.child_b_method()

print("\nHybrid Inheritance:")
derived_of_derived = DerivedOfDerived()
derived_of_derived.base_method()
derived_of_derived.derived1_method()
derived_of_derived.derived2_method()
derived_of_derived.derived_of_derived_method()
```

16) Create a Person class with a constructor that takes two arguments name and age.

Create a child class Employee that inherits from Person and adds a new attribute salary.

Override the **init** method in Employee to call the parent class's **init** method using the super() and then initialize the salary attribute.

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def display_info(self):
        print(f"Name: {self.name}, Age: {self.age}")

class Employee(Person):
    def __init__(self, name, age, salary):
        super().__init__(name, age)
        self.salary = salary

    def display_info(self):
        super().display_info()
        print(f"Salary: ${self.salary}")

# Create an employee
```

```
employee = Employee("John Doe", 30, 50000)
employee.display_info()
```

17) Create a Shape class with a draw method that is not implemented.

Create three child classes Rectangle, Circle, and Triangle that implement the draw method with their respective drawing behaviors.

Create a list of Shape objects that includes one instance of each child class, and then iterate through the list and call the draw method on each object.

```
class Shape:
    def draw(self):
        raise NotImplementedError("Subclass must implement abstract
method")

class Rectangle(Shape):
    def draw(self):
        print("Drawing a rectangle")

class Circle(Shape):
    def draw(self):
        print("Drawing a circle")

class Triangle(Shape):
    def draw(self):
        print("Drawing a triangle")


shapes = [Rectangle(), Circle(), Triangle()]

for shape in shapes:
    shape.draw()
```

```
Drawing a rectangle
Drawing a circle
Drawing a triangle
```