Python Programming - 2301CS404

Lab - 1

01) WAP to print “Hello World”

```
print("Hello World")
```

```
Hello World
```

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

```
a = 3
b = 2
print(a+b)
```

```
5
```

```
# a = 3
# b = 2
# print(a+b)

x = int(input("Enter First Value : "))
y = int(input("Enter Second Value : "))
print("Sum of 2 Number is : ",(x+y))
```

```
Enter First Value :  1
Enter Second Value :  2
Sum of 2 Number is :  3
```

03) WAP to check the type of the variable.

```
a = 1.2
b = "Darshan University"
print(type(a))
print(type(b))
```

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

04) WAP to calculate simple interest.

```
p = float(input("Enter the principal amount: "))
r = float(input("Enter the rate of interest: "))
t = float(input("Enter the time period in years: "))
interest = (p*r*t)/100
print(interest)
```

```
Enter the principal amount:  20
Enter the rate of interest:  32
Enter the time period in years:  10
64.0
```

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

```
import math
r = 10
print("Area = ",math.pi*r*r)
print("Perimeter = ",2*math.pi*r)
```

```
Area =  314.1592653589793
Perimeter =  62.83185307179586
```

06) WAP to calculate area of a triangle.

```
b = float(input("Enter the base of the triangle: "))
h = float(input("Enter the height of the triangle: "))
print("Area Of Triangle = ",(1/2*b*h))
```

```
Enter the base of the triangle:  20
Enter the height of the triangle:  10
Area Of Triangle =  100.0
```

07) WAP to compute quotient and remainder.

```
a = int(input("Enter 1st Number : "))
b = int(input("Enter 2nd Number : "))
print("Remainder = ",a%b)
```

```
Enter 1st Number :  7
Enter 2nd Number :  3
Remainder =  1
```

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

```
print("Select conversion type:")
print("Enter 1 For Celsius to Fahrenheit")
print("Enter 2 For Fahrenheit to Celsius")

choice = int(input("Enter Your Choice (1 or 2): "))
if choice == 1:
    cel = float(input("Enter temperature in Celsius: "))
    far = (cel * 9/5) + 32
    print("Fahrenheit : ",far)
elif choice == 2:
    far = float(input("Enter temperature in Fahrenheit: "))
    cel = (far - 32) * 5/9
    print("Celsius : ",cel)
else:
    print("Invalid Choice")
```

```
Select conversion type:
Enter 1 For Celsius to Fahrenheit
Enter 2 For Fahrenheit to Celsius
Enter Your Choice (1 or 2):  2
Enter temperature in Fahrenheit:  23
Celsius :  -5.0
```

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

```
x1 = float(input("Enter x1 for first point"))
y1 = float(input("Enter y1 for first point"))
x2 = float(input("Enter x2 for second point"))
y2 = float(input("Enter y2 for second point"))

distance = (((x1-x2)*2)+((y1-y2)*2))**1/2
print("distance is :",distance)
```

```
Enter x1 for first point 1
Enter y1 for first point 2
Enter x2 for second point 1
Enter y2 for second point 3
```

```
distance is : -1.0
```

10) WAP to print sum of n natural numbers.

```
n = int(input("Enter a number: "))
sum_of_n = n * (n + 1) // 2
print("The Sum of First ",n ,"natural numbers is: ",sum_of_n)
```

```
Enter a number:  4
The Sum of First 4 natural numbers is: 10
```

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

```
n = int(input("Enter a number: "))
sum_of_squares = (n * (n + 1) * (2 * n + 1)) // 6
print("The Sum of Squares of First ",n," natural numbers is: ",sum_of_squares)
```

```
Enter a number:  3
The Sum of Squares of First 3 natural numbers is: 14
```

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

```
fname = input("Enter the First Name of the Student: ")
lname = input("Enter the Last Name of the Student: ")

print("Full Name Of Student is : ",fname+" "+lname)
```

```
Enter the first name of the student:  Nency
Enter the last name of the student:  Parmar
Full Name Of Student is :  Nency Parmar
```

13) WAP to swap two numbers.

```
a = int(input("Enter 1st Number : "))
b = int(input("Enter 2nd Number : "))
print("Before Swap : ",a,b)
temp = a
a = b
b = temp
print("After Swap : ",a,b)
```

```
Enter 1st Number :  2
Enter 2nd Number :  3
Before Swap :  2 3
After Swap :  3 2
```

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

```
km = float(input("Enter the distance in kilometers: "))

m = km * 1000
feet = km * 3280.84
inch = km * 39370.1
cm = km * 100000
print("Distance in meters:", m)
print("Distance in feet:", feet)
print("Distance in inches:", inch)
print("Distance in centimeters:", cm)
```

```
The history saving thread hit an unexpected error (OperationalError('attempt to write a readonly database')).History will not be writter
Enter the distance in kilometers:  1
Distance in meters: 1000.0
Distance in feet: 3280.84
Distance in inches: 39370.1
Distance in centimeters: 100000.0
```

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

```
day = int(input("Enter the day (1-31): "))
month = int(input("Enter the month (1-12): "))
year = int(input("Enter the year: "))
if 1 <= day <= 31 and 1 <= month <= 12 and year > 0:
    print("The date is: ",day,"-",month,"-",year)
else:
    print("Invalid date! Please enter valid day, month, and year.")
```

```
Enter the day (1-31):  10
Enter the month (1-12):  06
Enter the year:  2004
The date is:  10 - 6 - 2004
```

Start coding or generate with AI.

Python Programming - 2301CS404

Lab - 2

Nency parmar | 23010101193 | 27/11/2024

if..else..

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

```
a = int(input("Enter a Number : "))
if(a < 0):
    print(a,"is Negative Number.")
else:
    print(a,"is Positive Number.")
```

```
Enter a Number :  2
2 is Positive Number.
```

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

```
a = int(input("Enter a Number : "))
if(a%2 == 0 ):
    print(a,"is Even Number.")
else:
    print(a,"is Odd Number.")
```

```
Enter a Number :  2
2 is Even Number.
```

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

```
a = int(input("Enter 1st Number : "))
b = int(input("Enter 2nd Number : "))
if(b < a):
```

```
    print(a,"is Largest Number.")
else:
    print(b,"is Largest Number.")
```

```
Enter 1st Number :  3
Enter 2nd Number :  2
3 is Largest Number.
```

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

```
a = int(input("Enter 1st Number : "))
b = int(input("Enter 2nd Number : "))
c = int(input("Enter 3rd Number : "))
if(a > b):
    print(a,"is Largest Number.")
elif(b > c):
    print(b,"is Largest Number.")
else:
    print(c,"is Largest Number.")
```

```
Enter 1st Number :  4
Enter 2nd Number :  3
Enter 3rd Number :  2
4 is Largest Number.
```

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]

```
year = int(input("Enter a Year : "))
if(year % 4 == 0 and year % 100 != 0 or year % 400 == 0):
    print(year,"is Leap Year.")
else:
    print(year,"is not Leap Year.")
```

```
Enter a Year :  2024
2024 is Leap Year.
```

```
day_number = int(input("Enter a number (1-7) to get the day of the week: "))
match day_number:
    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 number! Please enter a number between 1 and 7.")
```

```
Enter a number (1-7) to get the day of the week:  2
Tuesday
```

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

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

```
a = int(input("Enter 1st Number : "))
b = int(input("Enter 2nd Number : "))

print("Enter 1 for Addition.")
print("Enter 2 for Substraction.")
print("Enter 3 for Multiplication.")
```

```
print("Enter 4 for Division.")
choice = int(input("Enter Your Choice : "))
if(choice == 1):
    sum = a + b
    print("Result is : ",sum)
elif(choice == 2):
    sub = a - b
    print("Result is : ",sub)
elif(choice == 3):
    multi = a * b
    print("Result is : ",multi)
elif(choice == 4):
    div = a / b
    print("Result is : ",div)
else:
    print("Please Enter Valid Choice!!")
```

```
Enter 1st Number :  2
Enter 2nd Number :  1
Enter 1 for Addition.
Enter 2 for Substraction.
Enter 3 for Multiplication.
Enter 4 for Division.
Enter Your Choice :  4
Result is :  2.0
```

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

```
m1 = int(input("Enter 1st Subject Marks : "))
m2 = int(input("Enter 2nd Subject Marks : "))
m3 = int(input("Enter 3rd Subject Marks : "))
m4 = int(input("Enter 4th Subject Marks : "))
m5 = int(input("Enter 5th Subject Marks : "))

total = 500
sum = m1 + m2 + m3 + m4 + m5
per = (sum*100)/total
print("Your Percentage is : ",per)
if(per > 70):
    print("Distinction!!!")
elif(per <= 70 and per > 60):
    print("First Class!!!")
elif(per <= 60 and per > 45):
    print("Second Class!!!")
elif(per <= 45 and per > 35):
    print("Pass!!")
else:
    print("Fail!!!!")
```

```
Enter 1st Subject Marks :  98
Enter 2nd Subject Marks :  96
Enter 3rd Subject Marks :  94
Enter 4th Subject Marks :  63
Enter 5th Subject Marks :  65
Your Percentage is :  83.2
Distinction!!!
```

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.

```
import math
s1 = int(input("Enter 1st Side : "))
s2 = int(input("Enter 2nd Side : "))
s3 = int(input("Enter 3rd Side : "))
```

```
power = (math.pow(s1,2) + math.pow(s2,2)) == math.pow(s3,2)
power1 = (math.pow(s2,2) + math.pow(s3,2)) == math.pow(s1,2)
power2 = (math.pow(s1,2) + math.pow(s3,2)) == math.pow(s2,2)

if(s1 == s2 and s2 == s3 and s3 == s1):
    print("Triangle is Equilateral.")
elif(s1 == s2 or s2 == s3 or s3 == s1):
    print("Triangle is Isosceles.")
elif(power or power1 or power2):
    print("Triangle is Right-Angled.")
elif(s1 != s2 and s2 != s3 and s3 != s1):
    print("Triangle is Scalene.")
```

```
Enter 1st Side :  45
Enter 2nd Side :  45
Enter 3rd Side :  90
Triangle is Isosceles.
```

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

```
a = int(input("Enter 1st Number: "))
b = int(input("Enter 2nd Number: "))
c = int(input("Enter 3rd Number: "))

# Check for the second largest number
if (a > b and a < c) or (a > c and a < b):
    print(a, "is the Second Largest.")
elif (b > a and b < c) or (b > c and b < a):
    print(b, "is the Second Largest.")
else:
    print(c, "is the Second Largest.")
```

```
Enter 1st Number:  16
Enter 2nd Number:  10
Enter 3rd Number:  8
10 is the Second Largest.
```

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

```
units = int(input("Enter the Electricity Units consumed: "))
if(units <= 50):
    bill = units * 2.60
elif(units <= 100):
    bill = (50 * 2.60) + ((units - 50) * 3.25)
elif(units <= 200):
    bill = (50 * 2.60) + (50 * 3.25) + ((units - 100) * 5.26)
else:
    bill = (50 * 2.60) + (50 * 3.25) + (100 * 5.26) + ((units - 200) * 8.45)
print("The total electricity bill for {units} units is : ",bill)
```

```
Enter the Electricity Units consumed:  230
The total electricity bill for {units} units is :  1072.0
```

Python Programming - 2301CS404

Lab - 3

for and while loop

01) WAP to print 1 to 10.

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

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

```
9
10
```

02) WAP to print 1 to n.

```
n=int(input("Enter number :"))
for i in range (1,n+1):
    print(i)
```

```
Enter number : 8
1
2
3
4
5
6
7
8
```

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

```
n=int(input("Enter integer :"))
for i in range (1,n+1):
    if(i%2!=0):
        print(i)
```

```
Enter integer : 9
1
3
5
7
9
```

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

```
n=int(input("Enter integer :"))
for i in range (1,n+1):
    if(i%2==0 and i%3!=0):
        print(i)
```

```
Enter integer : 9
2
4
8
```

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

```
n=int(input("Enter integer :"))
total=0
for i in range (1,n+1):
    total=total+i
print(total)
```

```
Enter integer : 10
55
```

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

```
n=int(input("Enter integer :"))
total=0
for i in range(1,n+1):
    total=total+(i**2)
total
```

```
Enter integer : 4
30
```

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

```
n=int(input("Enter integer :"))
total=0
for i in range(1,n+1):
    if(i%2==0):
        total=total-i
    else:
        total=total+i
total
```

```
Enter integer : 5
3
```

08) WAP to print multiplication table of given number.

```
n=int(input("Enter integer :"))
for i in range(1,10):
    print(n,"*",i,"=",n*i)
```

```
Enter integer : 5
5 * 1 = 5
```

```
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
```

09) WAP to find factorial of the given number.

```
n=int(input("Enter integer :"))
factorial=1
for i in range (1,n+1):
    factorial=factorial*i
factorial
```

```
Enter integer : 4
24
```

10) WAP to find factors of the given number.

```
n=int(input("Enter integer :"))

for i in range(1,n+1):
    if(n%i==0):
        print(i)
```

```
Enter integer : 8
1
2
4
8
```

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

```
n=int(input("Enter integer :"))
temp=0
for i in range(1,n+1):
    if(n%i==0):
        temp=temp+1
if(temp==2):
    print("Prime")
else:
    print("Not a Prime")
```

```
Enter integer : 7
Prime
```

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

```
digit=int(input("Enter digit :"))
num=str(digit)
sum=0
for i in num:
    sum=sum+int(i)
print(sum)
```

```
Enter digit : 9889
34
```

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

```
a=int(input("Enter integer"))
no=str(a)
b=no[::-1]
if(no==b):
    print("Palindrome")
else:
    print("Not Pal.")
```

```
Enter integer 110011
Palindrome
```

14) WAP to print GCD of given two numbers.

```
import math
a=int(input("Enter first digit"))
b=int(input("Enter second digit"))
ans=math.gcd(a,b)
print(f"GCD of {a} and {b} is {ans}")
```

```
Enter first digit 12
Enter second digit 32
GCD of 12 and 32 is 4
```

```
a=int(input("Enter first integer"))
b=int(input("Enter second integer"))
while(b!=0):
```

```
    a,b=b,a%b
print(a)
```

```
Enter first integer 12
Enter second integer 32
4
```

Start coding or generate with AI.

Python Programming - 2301CS404

Lab - 4

String

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

```
a=input("Enter string :")
b=a[::-1]
if(b==a):
    print("Palindrome String.")
else:
    print("Not pal.")
```

```
Enter string : hello
Not pal.
```

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

```
a=input("Enter string :")
b=a[::-1]
b
```

03) WAP to remove ith character from given string.

```
a=input("Enter string :")
c=int(input("Enter ith value :"))
for i in range(len(a)+1):
    if(c==i):
        b=a.replace(a[i],"")
b
```

```
Enter string : hello world
Enter ith value : 5
'helloworld'
```

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

```
a=input("Enter string :")
temp=0
for i in a:
    temp=temp+1
print("Length :",temp)
```

```
Enter string : hello
Length : 5
```

05) WAP to print even length word in string.

```
a=input("Enter string :")
b=''
temp=0
for i in a:
    if(a.split(" ")):
        b=a.split(" ")


for i in b:
    if(len(b[temp])%2==0):
```

```
        print(b[temp])
    temp=temp+1
```

```
Enter string : asdf darshan university city diet
asdf
university
city
diet
```

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

```
a=input("Enter string :")
a=a.lower()
temp=0
for i in a:
    if(i in ['a','e','i','o','u']):
        temp=temp+1
print("Vowels :",temp)
```

```
Enter string : darshan university
Vowels : 6
```

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

```
a=input("Enter string :")
b=a.replace(a[0],a[0].upper())
c=b.replace(a[len(a)-1],a[len(a)-1].upper())
print(c)
```

```
Enter string : asdf
AsdF
```

08) WAP to convert given array to string.

```
a=''
n=int(input("Enter length of array"))
for i in range(n):
    a[i]=input("Enter element :")
b=str('')
for i in range(len)
```

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

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

```
password=input("Enter password :")
confirm=input("Reenter password :")
if(password in confirm):
    print("Password Matched👍 ")
else:
    print("Password doesn't match !")
```

```
Enter password : ABC
Reenter password : BCA
Password doesn't match !
```

10) : Display credit card number.

card no. : 1234 5678 9012 3456

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

```
a="1234 1234 8765 7654"
b="**** **** **** "+a[-4:]
b
```

```
a=1287763454237623
b=str(a)
c=' '.join([b[i:i+4] for i in range(0,len(b),4)])
d='**** **** **** '+c[-4:]
d
```

```
'**** **** **** 7623'
```

```
a=int(input("Enter 16 digits :"))
b=str(a)
c=' '.join([b[i:i+4] for i in range(0,len(b),4)])
d=" "
for i in c:
    d=c.split(" ")
print(str(d))
```

```
Enter 16 digits : 36
['36']
```

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

s1 = decimal and s2 = medical are Anagram

```
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.
```

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

input : EHlsarwiwhtwMV

output : lsarwiwhtwEHMV

```
def rearrange_string(s):
    lower_case = [ch for ch in s if ch.islower()]
    upper_case = [ch for ch in s if ch.isupper()]

    return ''.join(lower_case) + ''.join(upper_case)

# Example usage
input_str = "EHlsarwiwhtwMV"
output_str = rearrange_string(input_str)

print("Output:", output_str)
```

```
Output: lsarwiwhtwEHMV
```

Start coding or generate with AI.

Python Programming - 2301CS404

Lab - 5

List

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

```
a=[]
b=int(input("size"))
for i in range(b):
    a.append(int(input("Int")))
a
temp=0
for i in a:
     temp=temp+i
print(temp)
```

```
size 4
Int 1
Int 2
Int 3
Int 4
10
```

02) WAP to find largest element in a List.

```
b=int(input("BInt"))
a=[int(input()) for i in range(b)]
a.sort(reverse=True)
# a.reverse()
print("Largest element",a[0])
```

```
BInt 4
 3
 4
 2
 1
4
```

03) WAP to find the length of a List.

```
b=int(input())
a=[input() for i in range(b)]
temp=0
for i in a:
    temp=temp+1
print("Length",temp)
```

```
 5
 asdf
 zxcv
 qwer
 poiu
 ,mn
Length 5
```

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

```
b=int(input())
a=[input() for i in range(b)]
temp=a[0]
a[0]=a[len(a)-1]
```

```
a[len(a)-1]=temp
a
```

```
4
1
2
3
4
['4', '2', '3', '1']
```

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

```
b=int(input())
a=[int(input()) for i in range(b)]
c=a.insert(int(len(a)/2),a)
print(a,c)
```

```
4
1
2
3
4
[1, 2, [...], 3, 4] None
```

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

```
a=[1,2,3,4,5,6]
temp=0
pos1=int(input("Enter first position :"))
pos2=int(input("Enter second position :"))
temp=a[pos1]
a[pos1]=a[pos2]
a[pos2]=temp
print(a)
```

```
Enter first position : 3
Enter second position : 1
[1, 4, 3, 2, 5, 6]
```

07) WAP to reverse the list entered by user.

```
a=[]
for i in range(0,6):
    a.append(int(input("Enter number: ")))
b=a[::-1]
print(a," _ . _ ",b)
```

```
Enter number:  1
Enter number:  2
Enter number:  3
Enter number:  4
Enter number:  5
Enter number:  6
[1, 2, 3, 4, 5, 6]  _ _  [6, 5, 4, 3, 2, 1]
```

08) WAP to print even numbers in a list.

```
a=[i for i in range(1,11) if(i%2==0)]
print(a)
```

```
[2, 4, 6, 8, 10]
```

09) WAP to count unique items in a list.

```
a=[1,11,2,3,4,4,3,2,5,11]
b=len(set(a))
print(b)
```

```
6
```

10) WAP to copy a list.

```
a=[i for i in range(1,7)]
b=a
print(b)
```

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

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

```
n=int(input("Enter range :"))
a=[i for i in range(1,n) if(i%2!=0)]
print(a)
```

```
Enter range : 6
[1, 3, 5]
```

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

```
a=[1,2,3,4,3,2,1,2,1,2,1]
b=a.count(2)
print(b)
```

```
4
```

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

```
a=[9,3,2,5,6,2,1]
a.sort(reverse=True)
print(a[1])
```

```
6
```

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

```
a = [1, 2, 2, 3, 3, 3, 4, 4, 5, 5, 5, 5]
K = 1

result = []
for item in set(a):
    if a.count(item) > K:
        result.append(item)

print(result)
```

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

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

```
#with list comprehension
a=[i**2 for i in range(10)]
print(a)
#wwithout list comprehension
a=[]
for i in range(10):
    a.append(i**2)
print(a)
```

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

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

```
a=input("Enter list of fruits :").split(',')
b=[i for i in a if(i.strip().lower().startswith('b'))]
print(b)
```

```
Enter list of fruits : apple,peach,banana,blueberry,cherry
['banana', 'blueberry']
```

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

```
a=[1,2,3,4,5]
b=[9,5,3,2,4]
c=list(set(a) & set(b))
c
```

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

Start coding or generate with AI.

Python Programming - 2301CS404

Lab - 6

Tuple

01) WAP to find sum of tuple elements.

```
a=(1,2,3)
temp=0
for i in a:
    temp=temp+i
print(temp)
```

```
6
```

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

```
a = (10, 20, 4, 45, 99, 5, 4, 23, 50)

k = 3

b = sorted(a)

minelement = b[:k]

maxelement = b[-k:]

print("Minimum K elements:", minelement)
print("Maximum K elements:", maxelement)
```

```
Minimum K elements: [4, 4, 5]
Maximum K elements: [45, 50, 99]
```

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

```
def find_tuples_divisible_by_k(tuples_list, k):
    result = [t for t in tuples_list if all(element % k == 0 for element in t)]
    return result

tuples_list = [(10, 20, 30), (12, 15, 18), (5, 10, 20), (8, 16, 24)]
k = 5

output = find_tuples_divisible_by_k(tuples_list, k)
print("Tuples where all elements are divisible by", k, ":", output)
```

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

```
def create_tuple_with_cube(numbers):
    return [(num, num**3) for num in numbers]

numbers = [1, 2, 3, 4, 5]
output = create_tuple_with_cube(numbers)

print("List of tuples with number and its cube:", output)
```

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

```
def find_positive_tuples(tuple_list):

    result = [t for t in tuple_list if all(i > 0 for i in t)]
    return result

tuple_list = [(1, 2, 3), (-1, 2, 3), (4, 5, 6), (-2, -3, 4), (7, 8, 9)]
positive_tuples = find_positive_tuples(tuple_list)

print("Tuples with all positive elements:", positive_tuples)
```

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

```
# list to tuple
a=[1,2,3,4]
b=tuple(a)
print(b)

# tuple to list
c=(1,2,3,4,4)
f=list(c)
print(f)
```

```
(1, 2, 3, 4)
[1, 2, 3, 4, 4]
```

07) WAP to remove tuples of length K.

```
def remove_tuples_of_length_k(tuples_list, k):
    return [t for t in tuples_list if len(t) != k]

tuples_list = [(1, 2, 3), (4, 5), (6, 7, 8), (9, 10)]
k = 3

output = remove_tuples_of_length_k(tuples_list, k)
print("List after removing tuples of length", k, ":", output)
```

08) WAP to remove duplicates from tuple.

```
def remove_duplicates(input_tuple):

    return tuple(set(input_tuple))

input_tuple = (1, 2, 2, 3, 4, 4, 5, 5, 6)
output_tuple = remove_duplicates(input_tuple)

print("Original tuple:", input_tuple)
print("Tuple after removing duplicates:", output_tuple)
```

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

```
def multiply_adjacent_elements(tup):
    result = [tup[i] * tup[i + 1] for i in range(len(tup) - 1)]
    return tuple(result)

input_tuple = (1, 2, 3, 4, 5)
output_tuple = multiply_adjacent_elements(input_tuple)

print("Resultant tuple after multiplying adjacent elements:", output_tuple)
```

```
Resultant tuple after multiplying adjacent elements: (2, 6, 12, 20)
```

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

```
def is_distinct(input_tuple):
    return len(input_tuple) == len(set(input_tuple))
input_tuple = (1, 2, 3, 4, 5)
print("Is the tuple distinct?", is_distinct(input_tuple))  # Output: True

input_tuple_with_duplicates = (1, 2, 2, 3, 4)
print("Is the tuple distinct?", is_distinct(input_tuple_with_duplicates))  # Output: False
```

```
Is the tuple distinct? True
Is the tuple distinct? False
```

Start coding or generate with AI.

Python Programming - 2301CS404

Lab - 7

Set & Dictionary

01) WAP to iterate over a set.

```
def iterate_set(my_set):
    for element in my_set:
        print(element)

# Example usage
my_set = {1, 2, 3, 4, 5}
iterate_set(my_set)
```

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

```
# Function to convert a set into list, string, and tuple
def convert_set(input_set):
    # Convert set to list
    list_version = list(input_set)

    # Convert set to string
    string_version = str(input_set)

    # Convert set to tuple
    tuple_version = tuple(input_set)

    return list_version, string_version, tuple_version

# Example usage
input_set = {1, 2, 3, 4, 5}
list_version, string_version, tuple_version = convert_set(input_set)

print("Set:", input_set)
print("List:", list_version)
print("String:", string_version)
print("Tuple:", tuple_version)
```

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

```
def find_max_min(my_set):
    # Finding the maximum and minimum values from the set
    max_value = max(my_set)
    min_value = min(my_set)
    return max_value, min_value

# Example usage
my_set = {10, 20, 30, 40, 50}
max_value, min_value = find_max_min(my_set)

print("Maximum value:", max_value)
print("Minimum value:", min_value)
```

04) WAP to perform union of two sets.

```
# Function to perform union of two sets
def union_of_sets(set1, set2):
```

```
    # Using union() method
    union_set = set1.union(set2)

    # Alternatively, you can use the | operator
    # union_set = set1 | set2

    return union_set

# Example usage
set1 = {1, 2, 3, 4}
set2 = {3, 4, 5, 6}

union_result = union_of_sets(set1, set2)

print("Set 1:", set1)
print("Set 2:", set2)
print("Union of Set 1 and Set 2:", union_result)
```

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

```
def have_common_element(list1, list2):
    # Convert both lists to sets and check if there is any intersection
    return bool(set(list1) & set(list2))

# Example usage
list1 = [1, 2, 3, 4, 5]
list2 = [5, 6, 7, 8, 9]

if have_common_element(list1, list2):
    print("The lists have at least one element in common.")
else:
    print("The lists have no elements in common.")
```

06) WAP to remove duplicates from list.

```
def remove_duplicates(input_list):
    # Convert the list to a set to remove duplicates, then convert back to a list
    return list(set(input_list))

# Example usage
input_list = [1, 2, 3, 4, 5, 2, 3, 4]
output_list = remove_duplicates(input_list)

print("List after removing duplicates:", output_list)
```

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

```
def find_unique_words(input_string):
    words = input_string.lower().split()

    unique_words = set(words)

    return unique_words

input_string = "Python is great and python is fun"
unique_words = find_unique_words(input_string)

print("Unique words in the string:", unique_words)
```

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

```
def remove_common_elements(A, B):
    A.difference_update(B)
    return A

A = {1, 2, 3, 4, 5}
B = {3, 4, 6, 7}

result = remove_common_elements(A, B)
print("Set A after removing common elements with set B:", result)
```

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

```
def are_anagrams(str1, str2):
    str1 = str1.replace(" ", "").lower()
    str2 = str2.replace(" ", "").lower()

    return set(str1) == set(str2) and len(str1) == len(str2)

str1 = "listen"
str2 = "silent"

if are_anagrams(str1, str2):
    print(f'"{str1}" and "{str2}" are anagrams.')
else:
    print(f'"{str1}" and "{str2}" are not anagrams.')
```

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

```
def find_common_elements(list1, list2, list3):
    set1 = set(list1)
    set2 = set(list2)
    set3 = set(list3)

    common_elements = set1 & set2 & set3

    return common_elements

list1 = [1, 2, 3, 4, 5]
list2 = [3, 4, 5, 6, 7]
list3 = [5, 8, 9, 3, 4]

common_elements = find_common_elements(list1, list2, list3)

print("Common elements in the three lists:", common_elements)
```

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

```
def count_vowels(s):
    vowels = {'a', 'e', 'i', 'o', 'u'}

    count = 0

    for char in s.lower():
        if char in vowels:
            count += 1
```

```
    return count

input_string = "Hello World"
vowel_count = count_vowels(input_string)

print(f"Number of vowels in '{input_string}': {vowel_count}")
```

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

```
def is_binary_string(input_string):
    return set(input_string).issubset({'0', '1'})

input_string = "10201"
if is_binary_string(input_string):
    print(f"'{input_string}' is a binary string.")
else:
    print(f"'{input_string}' is not a binary string.")
```

13) WAP to sort dictionary by key or value.

```
def sort_dict_by_value(d):
    return dict(sorted(d.items(), key=lambda item: item[1]))

my_dict = {'apple': 5, 'banana': 3, 'cherry': 8, 'date': 2}
sorted_dict_by_value = sort_dict_by_value(my_dict)

print("Sorted by value:", sorted_dict_by_value)
```

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

```
def sum_of_dict_values(input_dict):
    return sum(input_dict.values())

input_dict = {}

n = int(input("Enter the number of items in the dictionary: "))

for i in range(n):
    key = input(f"Enter key {i+1}: ")
```

```
        value = float(input(f"Enter value for key '{key}': "))
        input_dict[key] = value

    total_sum = sum_of_dict_values(input_dict)

    print("The sum of all values in the dictionary is:", total_sum)
```

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.

```
def handle_missing_key(d, key):
    value = d.get(key, "Key Not Found")
    print(value)

dict1 = {'a': 5, 'c': 8, 'e': 2}
key_to_lookup = 'd'

handle_missing_key(dict1, key_to_lookup)
```

Python Programming - 2301CS404

Lab - 8

User Defined Function

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

```
def calculate_bmi(mass, height):

    if height <= 0:
        return "Height must be greater than zero."
    bmi = mass / (height ** 2)
    return bmi

mass = 70
```

```
height = 1.75
bmi = calculate_bmi(mass, height)
print(f"The BMI is: {bmi}")
```

02) Write a function that add first n numbers.

```
def sum_of_first_n_numbers(n):
    return sum(range(1, n + 1))

n = 5
result = sum_of_first_n_numbers(n)

print(f"The sum of the first {n} numbers is: {result}")
```

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
    for i in range(2, int(n ** 0.5) + 1):
        if n % i == 0:
            return 0
    return 1
number = 11
result = is_prime(number)

if result == 1:
    print(f"{number} is a prime number.")
else:
    print(f"{number} is not a prime number.")
```

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

```
def is_prime(n):
    if n <= 1:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
```

```
    return True

def prime_numbers_in_range(start, end):
    return [num for num in range(start, end + 1) if is_prime(num)]

start = 10
end = 50
primes = prime_numbers_in_range(start, end)

print(f"Prime numbers between {start} and {end} are: {primes}")
```

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

```
def is_palindrome(input_string):
    input_string = input_string.replace(" ", "").lower()

    return input_string == input_string[::-1]

string = "A man a plan a canal Panama"
if is_palindrome(string):
    print("The string is a palindrome.")
else:
    print("The string is not a palindrome.")
```

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

```
def sum_of_elements(lst):
    return sum(lst)

numbers = [1, 2, 3, 4, 5]
result = sum_of_elements(numbers)

print(f"The sum of all elements in the list is: {result}")
```

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

```
def sum_of_first_elements(tuple_list):
    return sum([t[0] for t in tuple_list])
tuple_list = [(10, 20), (5, 15), (8, 25), (12, 30)]
```

```
result = sum_of_first_elements(tuple_list)

print("The sum of the first elements is:", result)
```

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)

n = 6
print(f"The {n}th term of Fibonacci series is: {fibonacci(n)}")
```

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):
    if rollno in student_dict:
        return student_dict[rollno]
    else:
        return "Roll number not found"

dict1 = {101: 'Ajay', 102: 'Rahul', 103: 'Jay', 104: 'Pooja'}
rollno = 103

student_name = get_student_name(rollno, dict1)

print(f"The student with roll number {rollno} is: {student_name}")
```

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):
    return sum(filter(lambda score: score % 10 == 0, scores))

scores = [200, 456, 300, 100, 234, 678]
result = sum_scores_ending_with_zero(scores)

print("Sum of scores ending with zero:", result)
```

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_dict(input_dict):
    inverted_dict = {v: k for k, v in input_dict.items()}
    return inverted_dict

input_dict = {'a': 10, 'b': 20, 'c': 30, 'd': 40}
inverted_dict = invert_dict(input_dict)

print("Before:", input_dict)
print("After:", inverted_dict)
```

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):
    s = s.lower()

    alphabet_set = set('abcdefghijklmnopqrstuvwxyz')

    char_set = set(filter(str.isalpha, s))

    return char_set == alphabet_set
```

```
s1 = "the quick brown fox jumps over the lazy dog"
if is_pangram(s1):
    print("The string is a Pangram.")
else:
    print("The string is not a Pangram.")
```

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):
    no_upper = 0
    no_lower = 0

    for char in s:
        if char.isupper():
            no_upper += 1
        elif char.islower():
            no_lower += 1

    return no_upper, no_lower

s1 = "AbcDEfgh"
no_upper, no_lower = count_upper_lower(s1)

print(f"no_upper = {no_upper}, no_lower = {no_lower}")
```

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

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

num1 = 10
num2 = 20

print("The smallest number is:", smallest(num1, num2))
```

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))

names_list = ['Alice', 'Bob', 'Charlotte', 'Michael', 'Jonathan', 'Eve']
long_names = filter_long_names(names_list)

print("Names with more than 7 characters:", long_names)
```

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))

names_list = ['alice', 'bob', 'charlie', 'dave']
capitalized_names = capitalize_names(names_list)

print("Capitalized names:", capitalized_names)
```

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(name, age):
    print(f"Hello {name}, you are {age} years old.")

greet("Alice", 25)

def greet(name, age):
    print(f"Hello {name}, you are {age} years old.")

greet(name="Alice", age=25)

def greet(name, age=30):
    print(f"Hello {name}, you are {age} years old.")

greet("Alice")
greet("Bob", 25)
```

```
def display_info(*args, **kwargs):
    print("Positional Arguments:", args)
    print("Keyword Arguments:", kwargs)

display_info(1, 2, 3, name="Alice", age=25)

def process_data(a, b, /, c, d, *, e=10):
    print(f"a={a}, b={b}, c={c}, d={d}, e={e}")

process_data(1, 2, c=3, d=4)
process_data(1, 2, 3, 4, e=15)
```

Python Programming - 2301CS404

Lab - 9

File I/O

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

```
def read_file_as_string(file_path):
    try:
```

```
            with open(file_path, 'r') as file:
                content = file.read()
            return content
        except FileNotFoundError:
            return f"File not found at {file_path}"

    def read_file_line_by_line(file_path):
        try:
            with open(file_path, 'r') as file:
                lines = file.readlines()
            return lines
        except FileNotFoundError:
            return f"File not found at {file_path}"

    def read_file_as_list(file_path):
        try:
            with open(file_path, 'r') as file:
                lines = [line.strip() for line in file.readlines()]
            return lines
        except FileNotFoundError:
            return f"File not found at {file_path}"

    file_path = 'path/to/your/file.txt'

    print("File content as string:")
    print(read_file_as_string(file_path))

    print("\nFile content line by line:")
    print(read_file_line_by_line(file_path))

    print("\nFile content as list:")
    print(read_file_as_list(file_path))
```

02) WAP to create file named "new.txt" only if it doesn't exist.

```
def create_file_if_not_exists(filename):
    try:
        with open(filename, 'x') as file:
            print(f"File '{filename}' created successfully.")
    except FileExistsError:
        print(f"File '{filename}' already exists.")

create_file_if_not_exists("new.txt")
```

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

```
def read_first_5_lines(file_path):
    try:
        with open(file_path, 'r') as file:
            lines = [file.readline().strip() for _ in range(5)]
        return lines
    except FileNotFoundError:
        return f"File not found at {file_path}"
    except Exception as e:
        return str(e)

file_path = 'path/to/your/file.txt'

lines = read_first_5_lines(file_path)
print("First 5 lines from the file:")
for line in lines:
    print(line)
```

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

```
def find_longest_word_in_file(filename):
    try:
        with open(filename, 'r') as file:
            content = file.read()

            words = content.split()

            max_length = max(len(word) for word in words)

            longest_words = [word for word in words if len(word) == max_length]

            return longest_words

    except FileNotFoundError:
        return f"The file '{filename}' was not found."
    except Exception as e:
        return f"An error occurred: {e}"

filename = "sample.txt"
longest_words = find_longest_word_in_file(filename)

if isinstance(longest_words, list):
    print(f"The longest word(s): {', '.join(longest_words)}")
else:
    print(longest_words)
```

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

```
def count_lines_words_characters(file_path):
    try:
        lines_count = 0
        words_count = 0
        characters_count = 0

        with open(file_path, 'r') as file:
            for line in file:
                lines_count += 1
                words_count += len(line.split())
                characters_count += len(line)

        return lines_count, words_count, characters_count
    except FileNotFoundError:
        return f"File not found at {file_path}"
    except Exception as e:
        return str(e)

file_path = 'path/to/your/file.txt'

lines, words, characters = count_lines_words_characters(file_path)

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

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

```
def copy_file_content(source_file, destination_file):
    try:
        with open(source_file, 'r') as source:
            content = source.read()

        with open(destination_file, 'w') as destination:
            destination.write(content)

        print(f"Content successfully copied from '{source_file}' to '{destination_file}'.")

    except FileNotFoundError:
        print(f"The file '{source_file}' was not found.")
    except Exception as e:
        print(f"An error occurred: {e}")

source_file = "source.txt"
```

```
destination_file = "destination.txt"
copy_file_content(source_file, destination_file)
```

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

```
import os

def file_size(file_path):
    try:
        size_in_bytes = os.path.getsize(file_path)
        return size_in_bytes
    except FileNotFoundError:
        return f"File not found at {file_path}"
    except Exception as e:
        return str(e)

file_path = 'path/to/your/file.txt'

size = file_size(file_path)
print(f"The size of the file is: {size} bytes")
```

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

```
def frequency(filename, word_to_count):
    try:
        with open(filename, 'r') as file:
            content = file.read()

            words = content.lower().split()

            word_count = words.count(word_to_count.lower())

            return word_count

    except FileNotFoundError:
        return f"The file '{filename}' was not found."
    except Exception as e:
        return f"An error occurred: {e}"

filename = "sample.txt"
word_to_count = "the"

result = frequency(filename, word_to_count)
```

```
if isinstance(result, int):
    print(f"The word '{word_to_count}' appears {result} times in the file.")
else:
    print(result)
```

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.

```
def store_scores_in_file(file_path):
    scores = []
    for i in range(1, 6):
        score = float(input(f"Enter the score for subject {i}: "))
        scores.append(score)

    try:
        with open(file_path, 'w') as file:
            for score in scores:
                file.write(str(score) + '\n')
        print(f"Scores successfully stored in {file_path}")
    except Exception as e:
        print(f"Error while storing the scores: {e}")

def fetch_scores_from_file(file_path):
    try:
        with open(file_path, 'r') as file:
            scores = [float(line.strip()) for line in file.readlines()]
        return scores
    except FileNotFoundError:
        return f"File not found at {file_path}"
    except Exception as e:
        return f"Error while reading the file: {e}"

def find_highest_score(scores):
    return max(scores)

file_path = 'scores.txt'

store_scores_in_file(file_path)

scores = fetch_scores_from_file(file_path)

if isinstance(scores, list):
    highest_score = find_highest_score(scores)
    print(f"The highest score is: {highest_score}")
else:
    print(scores)
```

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

(Note: each number should be in new line)

```
def is_prime(n):
    if n <= 1:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True

def write_prime_numbers_to_file(filename, count=100):
    prime_numbers = []
    num = 2

    while len(prime_numbers) < count:
        if is_prime(num):
            prime_numbers.append(num)
        num += 1

    with open(filename, 'w') as file:
        for prime in prime_numbers:
            file.write(f"{prime}\n")

    print(f"The first {count} prime numbers have been written to '{filename}'.")

write_prime_numbers_to_file("primenumbers.txt")
```

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

```
def merge_files(file1_path, file2_path, output_file_path):
    try:
        with open(file1_path, 'r') as file1:
            file1_content = file1.read()

        with open(file2_path, 'r') as file2:
            file2_content = file2.read()

        with open(output_file_path, 'w') as output_file:
            output_file.write(file1_content + "\n")
            output_file.write(file2_content)

        print(f"Contents of {file1_path} and {file2_path} have been merged into {output_file
    except FileNotFoundError as e:
        print(f"Error: {e}")
    except Exception as e:
```

```
        print(f"An error occurred: {e}")

file1_path = 'file1.txt'
file2_path = 'file2.txt'
output_file_path = 'merged_output.txt'

merge_files(file1_path, file2_path, output_file_path)
```

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

```
def replace_word_in_file(input_filename, output_filename, word1, word2):
    try:
        with open(input_filename, 'r') as file:
            content = file.read()
        updated_content = content.replace(word1, word2)

        with open(output_filename, 'w') as output_file:
            output_file.write(updated_content)

        print(f"Word '{word1}' has been replaced by '{word2}' and written to '{output_filena

    except FileNotFoundError:
        print(f"The file '{input_filename}' was not found.")
    except Exception as e:
        print(f"An error occurred: {e}")

input_filename = "sample.txt"
output_filename = "updated_sample.txt"
word1 = "oldword"
word2 = "newword"

replace_word_in_file(input_filename, output_filename, word1, word2)
```

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

```
Start coding or generate with AI.
```

Python Programming - 2301CS404

Lab - 10

Exception Handling

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.

```
def handle_exceptions():
    try:
```

```
        num1 = int(input("Enter the first number: "))
        num2 = int(input("Enter the second number: "))
        result = num1 / num2
        print(f"The result of division is: {result}")

        invalid_input = input("Enter a number to convert to an integer: ")
        converted_value = int(invalid_input)
        print(f"Converted value is: {converted_value}")

        invalid_operation = "Hello" + 5

    except ZeroDivisionError as e:
        print("Error: Cannot divide by zero!")

    except ValueError as e:
        print("Error: Invalid value entered for conversion!")

    except TypeError as e:
        print("Error: Invalid operation between incompatible types!")

    except (ZeroDivisionError, ValueError, TypeError) as e:
        print(f"An error occurred: {str(e)}")

handle_exceptions()
```

02) WAP to handle following exceptions:

1. IndexError
2. KeyError

```
def handle_exceptions():
    try:
        my_list = [1, 2, 3]
        print(my_list[5])
    except IndexError as e:
        print(f"IndexError occurred: {e}")

    try:
        my_dict = {'name': 'Alice', 'age': 25}
        print(my_dict['address'])
    except KeyError as e:
        print(f"KeyError occurred: {e}")

handle_exceptions()
```

03) WAP to handle following exceptions:

1. FileNotFoundError
2. ModuleNotFoundError

```
def handle_file_and_module_exceptions():
    try:
        # Attempt to open a file that doesn't exist
        file_name = input("Enter the name of the file to open: ")
        with open(file_name, 'r') as file:
            print(file.read())

    except FileNotFoundError as e:
        print(f"Error: The file '{file_name}' was not found!")

    try:
        # Attempt to import a module that might not exist
        module_name = input("Enter the name of the module to import: ")
        __import__(module_name)
        print(f"Module '{module_name}' imported successfully.")

    except ModuleNotFoundError as e:
        print(f"Error: The module '{module_name}' was not found!")

# Example usage
handle_file_and_module_exceptions()
```

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

```
def handle_all_exceptions():
    try:
        result = 10 / 0

        number = int("abc")

        my_list = [1, 2, 3]
        print(my_list[5])

        my_dict = {'name': 'Alice'}
        print(my_dict['age'])

    except Exception as e:
        print(f"An error occurred: {e}")

handle_all_exceptions()
```

05) WAP to demonstrate else and finally block.

```
def demonstrate_else_finally():
    try:
        num1 = int(input("Enter the first number: "))
        num2 = int(input("Enter the second number: "))

        result = num1 / num2
        print(f"The result of division is: {result}")

    except ZeroDivisionError as e:
        print("Error: Cannot divide by zero!")

    except ValueError as e:
        print("Error: Invalid input! Please enter a valid number.")

    else:
        print("Division was successful. No errors occurred.")

    finally:
        print("This block always runs, even if there is an error or not.")

demonstrate_else_finally()
```

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

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

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

```
def get_grades():
    try:
        grades_input = input("Enter a list of grades separated by commas: ")

        grades = [int(grade.strip()) for grade in grades_input.split(',')]

        print(f"Grades: {grades}")

    except ValueError:
        print("Error: Please enter valid numbers for the grades.")
```

```
get_grades()
```

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

```
def divide(a, b):
    try:
        result = a / b
        return result
    except ZeroDivisionError:
        print("Error: Division by zero is not allowed!")
        return None

numerator = float(input("Enter the numerator: "))
denominator = float(input("Enter the denominator: "))

result = divide(numerator, denominator)

if result is not None:
    print(f"The result of the division is: {result}")
else:
    print("Division could not be performed.")
```

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.

```
def get_age():
    try:
        # Prompt the user to enter their age
        age = int(input("Please enter your age: "))

        # Check if the age is less than 18
        if age < 18:
            raise ValueError("Enter Valid Age")
        else:
            print(f"Your age is: {age}")

    except ValueError as e:
        print(f"Error: {e}")
```

```
get_age()
```

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.

```
class InvalidUsernameError(Exception):
    # Custom Exception for invalid username
    def __init__(self, message):
        self.message = message
        super().__init__(self.message)

def check_username(username):
    try:
        # Check if username length is valid (between 5 and 15 characters)
        if len(username) < 5 or len(username) > 15:
            raise InvalidUsernameError("Username must be between 5 and 15 characters long")
        else:
            print(f"Valid username: {username}")

    except InvalidUsernameError as e:
        # Handle the custom exception
        print(f"Error: {e}")

# Example usage
username_input = input("Enter your username: ")
check_username(username_input)
```

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.

```
import math

class NegativeNumberError(Exception):
    def __init__(self, message):
```

```
            self.message = message
            super().__init__(self.message)

    # Function to calculate square root and handle custom exception
    def calculate_square_root():
        try:
            # Prompt the user to enter a number
            number = float(input("Enter a number to calculate its square root: "))

            # Check if the number is negative
            if number < 0:
                raise NegativeNumberError("Cannot calculate the square root of a negative number

            # Calculate and print the square root
            sqrt_value = math.sqrt(number)
            print(f"The square root of {number} is {sqrt_value}")

        except NegativeNumberError as e:
            print(f"Error: {e}")
        except ValueError:
            print("Error: Please enter a valid number.")

    calculate_square_root()
```

Python Programming - 2301CS404

Lab - 11

Modules

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

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

```
import mthmod as maths

a=int(input("Enter integer :"))
b=int(input("Enter integer :"))

c=maths.addnm(a,b)
```

```
d=maths.subnm(a,b)
e=maths.mulnm(a,b)
f=maths.dvnm(a,b)
print(c,d,e,f)
```

```
Enter integer :4
Enter integer :5
9 -1 20 0.8
```

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

```
import random
a=input("Enter string :")
print(random.choice(a))
```

```
Enter string :fgrgrg
g
```

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

```
a=[1,2,3,4,5,6,7,8,10]
import random
print(random.choice(a))
```

```
8
```

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

```
random.seed(4)
a=random.randint(1,4)
a
```

```
2
```

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

```
for i in range(0,3):
    a=random.randrange(100,999,5)
    print(a)
```

```
485
230
605
```

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

```
import random

def generate_lottery_tickets(num_tickets=100):
    tickets = [f"TICKET{random.randint(1000, 9999)}" for _ in range(num_tickets)]
    return tickets

def pick_winners(tickets):
    winner = random.choice(tickets)
    tickets.remove(winner)
    runner_up = random.choice(tickets)
    return winner, runner_up

def main():
    tickets = generate_lottery_tickets()

    winner, runner_up = pick_winners(tickets)

    print("Lottery Results:")
    print(f"Winner Ticket: {winner}")
    print(f"Runner-up Ticket: {runner_up}")

main()
```

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

```
import datetime
print(datetime.datetime.now())
```

```
2025-02-12 13:29:45.119015
```

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

```
from datetime import datetime, timedelta

date_str = "2025-03-11"

date = datetime.strptime(date_str, "%Y-%m-%d")
```

```
new_date = date - timedelta(days=7)

print("Original date:", date.strftime("%Y-%m-%d"))
print("Date after subtracting a week:", new_date.strftime("%Y-%m-%d"))
```

```
Original date: 2025-03-11
Date after subtracting a week: 2025-03-04
```

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

```
from datetime import datetime

def calculate_days_between_dates(date1, date2):
    date1 = datetime.strptime(date1, "%Y-%m-%d")
    date2 = datetime.strptime(date2, "%Y-%m-%d")

    difference = date2 - date1

    return abs(difference.days)

def main():
    date1 = input("Enter the first date (YYYY-MM-DD): ")
    date2 = input("Enter the second date (YYYY-MM-DD): ")

    days_between = calculate_days_between_dates(date1, date2)
    print(f"The number of days between {date1} and {date2} is: {days_between} days")

main()
```

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

```
from datetime import datetime

date_str = "2025-03-11"

date = datetime.strptime(date_str, "%Y-%m-%d")

day_of_week = date.strftime("%A")

print(f"The day of the week for {date_str} is: {day_of_week}")
```

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

```
from datetime import datetime, timedelta

current_datetime = datetime.now()
print("Current Date and Time: ", current_datetime)

current_date = current_datetime.date()
print("Current Date: ", current_date)

current_time = current_datetime.time()
print("Current Time: ", current_time)

formatted_datetime = current_datetime.strftime("%Y-%m-%d %H:%M:%S")
print("Formatted Date and Time: ", formatted_datetime)

date_string = "2025-03-11 15:30:00"
parsed_datetime = datetime.strptime(date_string, "%Y-%m-%d %H:%M:%S")
print("Parsed Date and Time: ", parsed_datetime)
```

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

```
import math

number = 16
sqrt_value = math.sqrt(number)
print(f"The square root of {number} is: {sqrt_value}")

n = 5
factorial_value = math.factorial(n)
print(f"The factorial of {n} is: {factorial_value}")

pi_value = math.pi
print(f"The value of pi is: {pi_value}")

angle_rad = math.radians(45)
sin_value = math.sin(angle_rad)
cos_value = math.cos(angle_rad)
tan_value = math.tan(angle_rad)

print(f"The sine of 45 degrees is: {sin_value}")
print(f"The cosine of 45 degrees is: {cos_value}")
print(f"The tangent of 45 degrees is: {tan_value}")

log_value = math.log(100, 10)
print(f"The logarithm of 100 with base 10 is: {log_value}")

number_to_round = 5.6789
rounded_value = round(number_to_round, 2)
print(f"The number {number_to_round} rounded to 2 decimal places is: {rounded_value}")
```

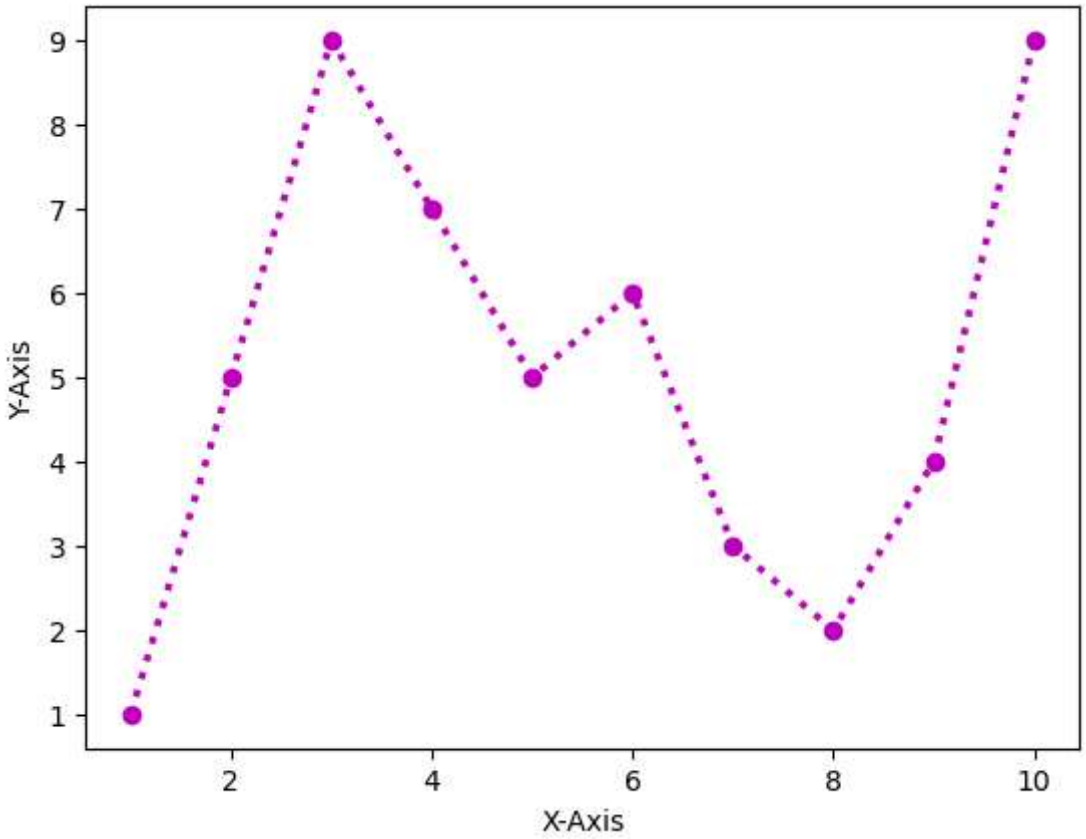
```
gcd_value = math.gcd(56, 98)
print(f"The greatest common divisor of 56 and 98 is: {gcd_value}")
```

Python Programming - 2301CS404

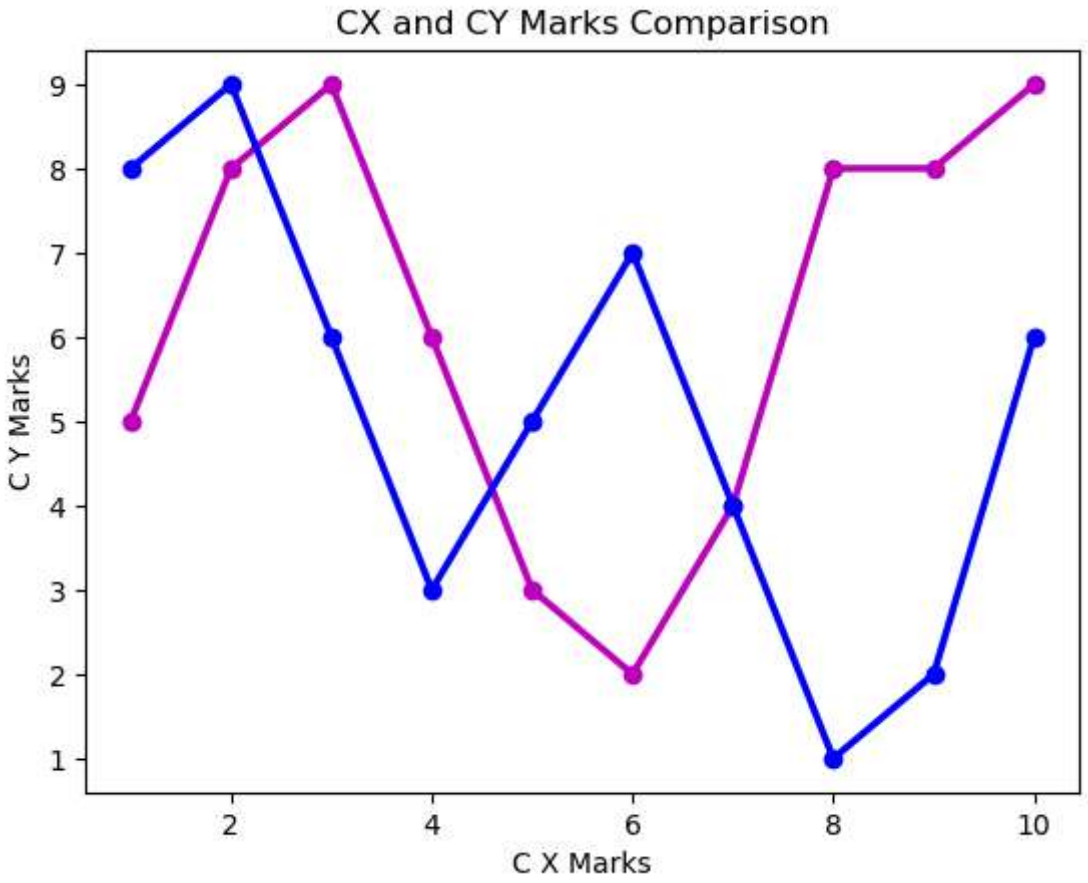
Lab - 12

```
#import matplotlib below
import matplotlib.pyplot as plt


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

plt.plot(x,y,ls=":",c="m",marker="o",lw=2.5)
plt.xlabel('X-Axis')
plt.ylabel('Y-Axis')
plt.show
# write a code to display the line chart of above x & y
```

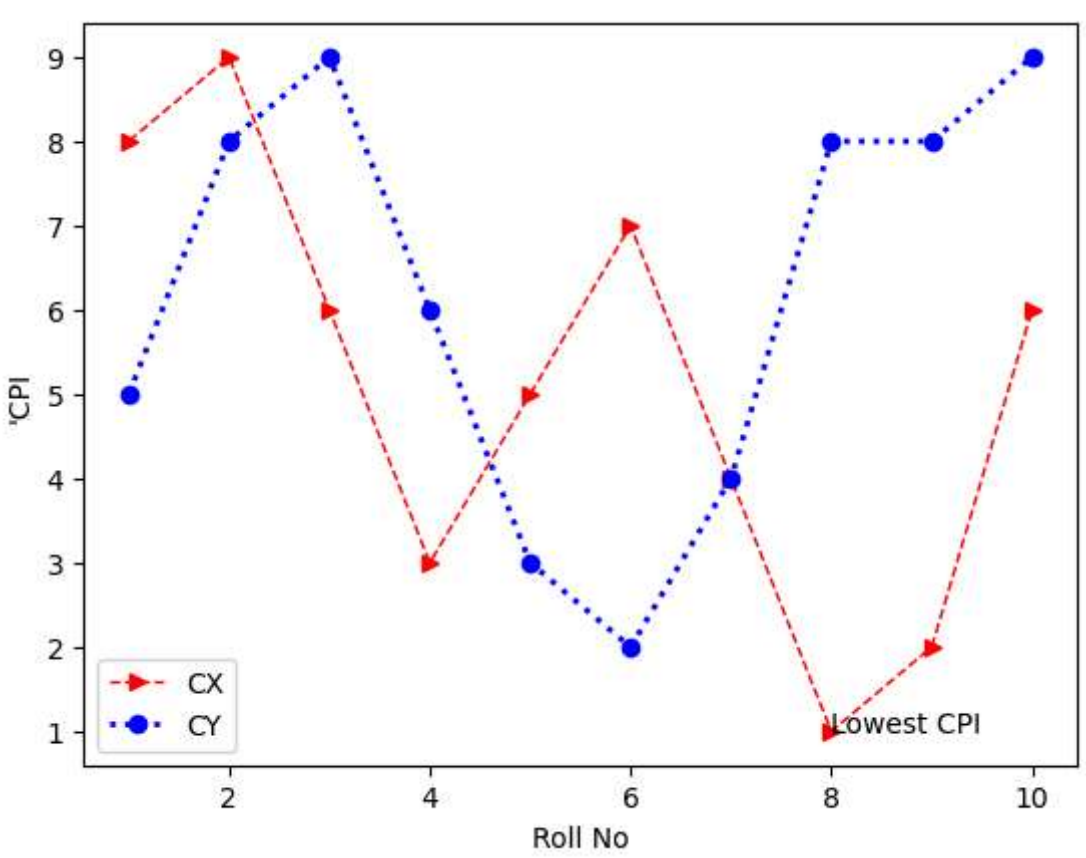
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



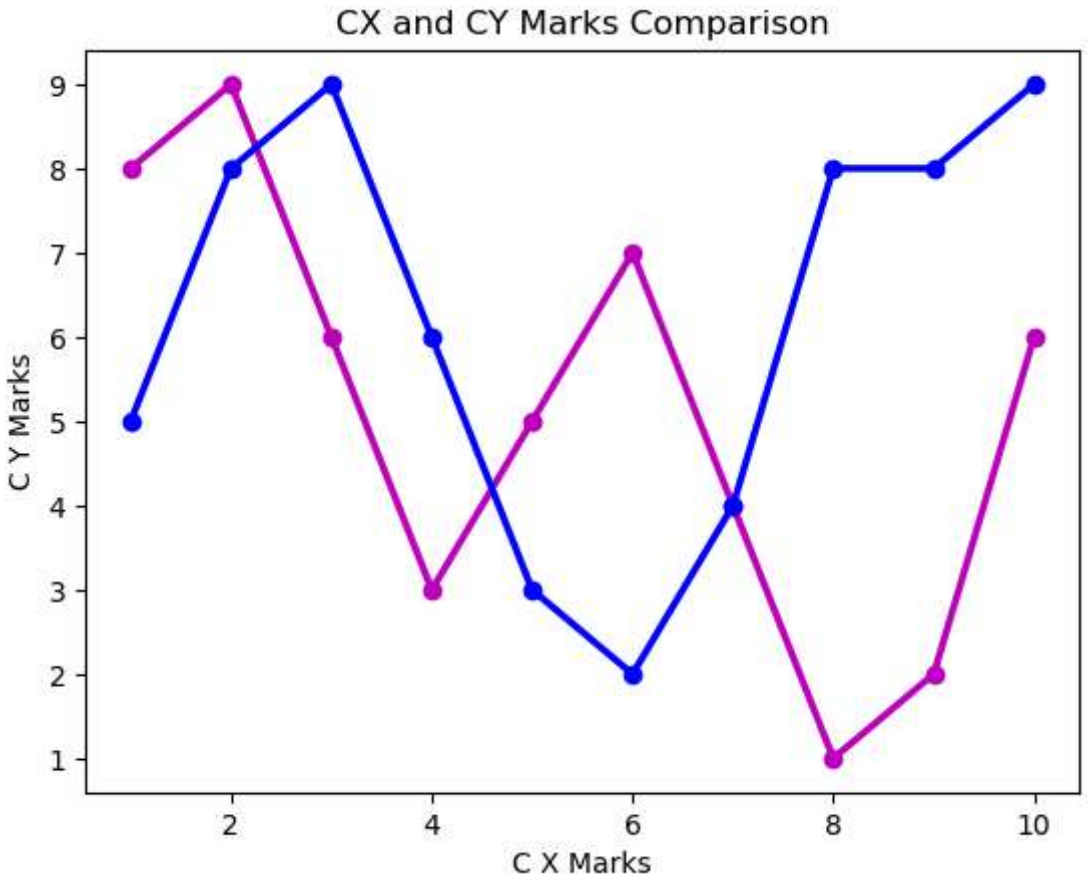

```
x = [1,2,3,4,5,6,7,8,9,10]
cxMarks = [5,8,9,6,3,2,4,8,8,9]
cyMarks = [8,9,6,3,5,7,4,1,2,6]

plt.plot(x,cxMarks,ls="-",c="m",marker="o",lw=2.5)
plt.plot(x,cyMarks,ls="-",c="b",marker="o",lw=2.5)
plt.xlabel("C X Marks")
plt.ylabel("C Y Marks")
plt.title("CX and CY Marks Comparison")
plt.show()
# write a code to display two lines in a line chart (data given above)
```




```
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,ls="-",c="m",marker="o",lw=2.5)
plt.plot(x,cyMarks,ls="-",c="b",marker="o",lw=2.5)
plt.xlabel("C X Marks")
plt.ylabel("C Y Marks")
plt.title("CX and CY Marks Comparison")
plt.show()
# write a code to generate below graph
```

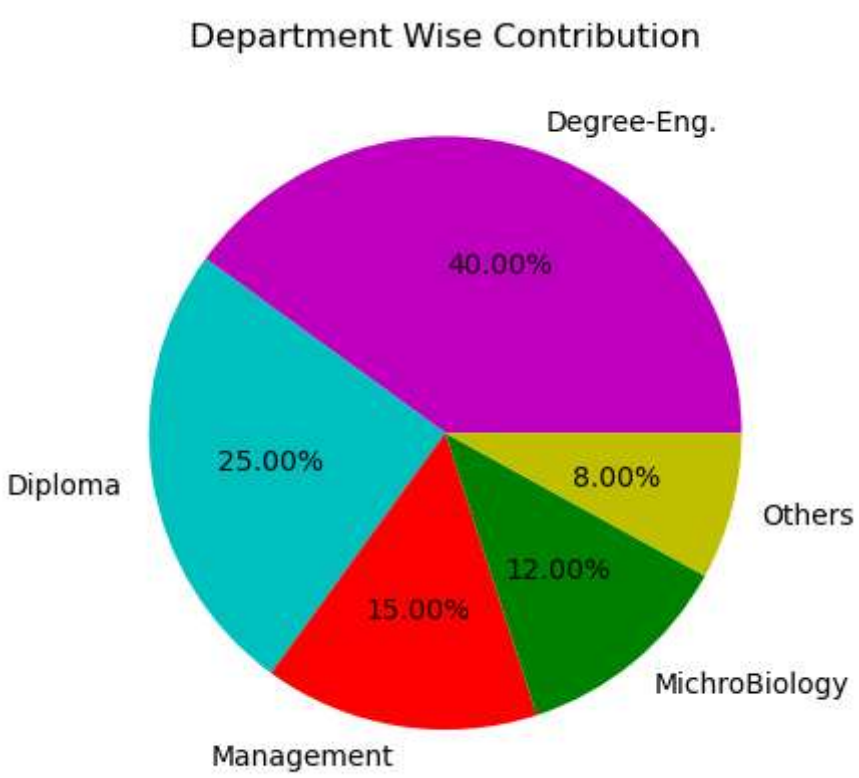



```
x = range(1,11,1)
cxMarks= [8,9,6,3,5,7,4,1,2,6]
cyMarks= [5,8,9,6,3,2,4,8,8,9]

plt.plot(x,cxMarks,ls="-",c="m",marker="o",lw=2.5)
plt.plot(x,cyMarks,ls="-",c="b",marker="o",lw=2.5)
plt.xlabel("C X Marks")
plt.ylabel("C Y Marks")
plt.title("CX and CY Marks Comparison")
plt.show()
```



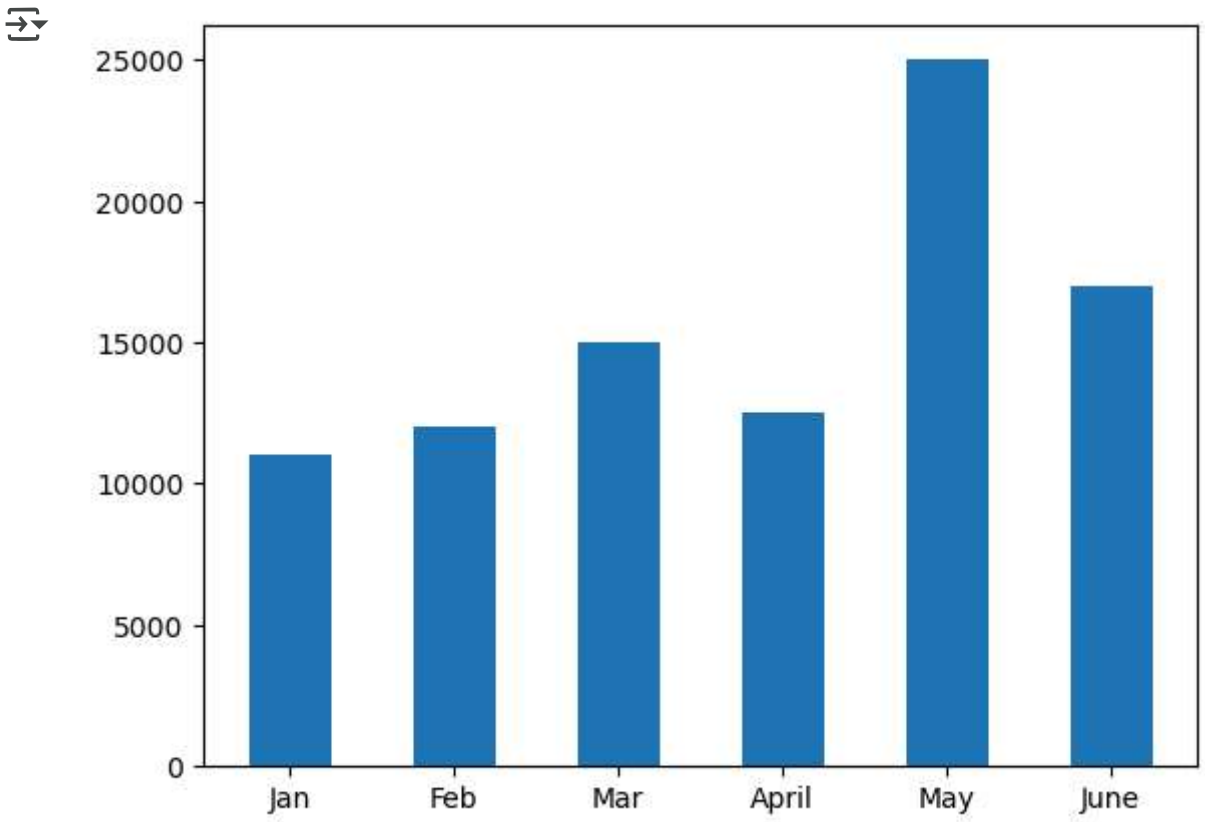
04) WAP to demonstrate the use of Pie chart.

```
dept = ["Degree-Eng.","Diploma","Management","MichroBiology","Others"]
students = [4000,2500,1500,1200,800]
col = ["m","c","r","g","y"]
plt.pie(students,labels=dept,autopct="%1.2f%%",colors = col)
plt.title("Department Wise Contribution")
plt.show()
```



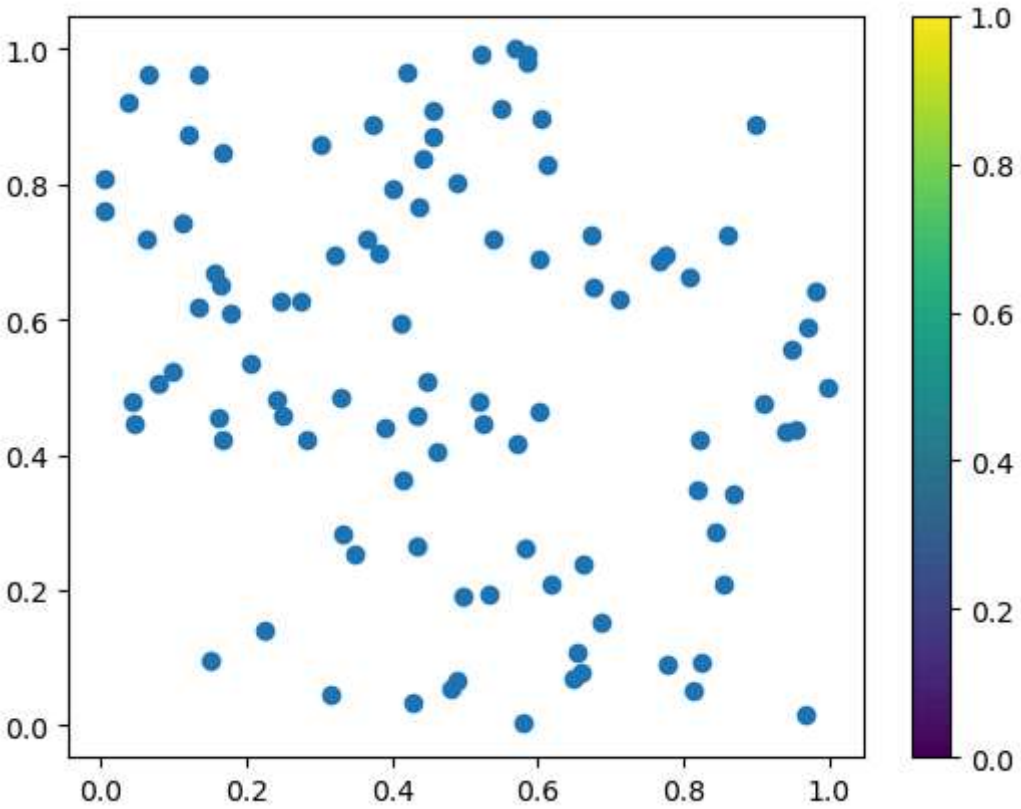
05) WAP to demonstrate the use of Bar chart.

```
month = ["Jan","Feb","Mar","April","May","June"]
visitors = [11000,12000,15000,12500,25000,17000]
bars = plt.bar(month,visitors,width=0.5)
plt.show()
```



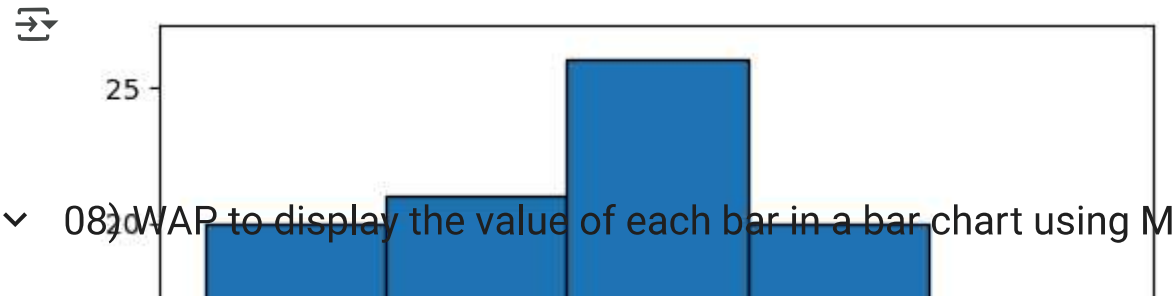
06) WAP to demonstrate the use of Scatter Plot.

```
import random
random.seed(10)
x = [random.random() for i in range(100)]
y = [random.random() for i in range(100)]
plt.scatter(x,y)
plt.colorbar()
plt.show()
```



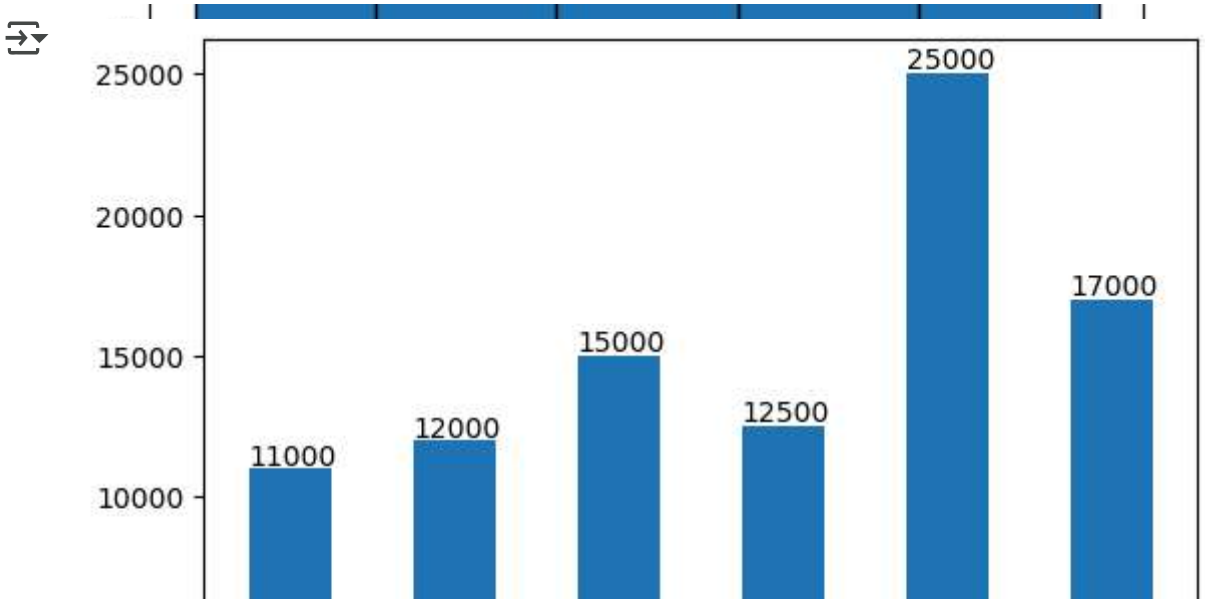
07) WAP to demonstrate the use of Histogram.

```
random.seed(10)
age = [random.randint(1,50) for i in range(100)]
plt.hist(age,edgecolor="k",bins = 5)
plt.xticks([1,11,21,31,41,51])
plt.show()
```



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

```
month = ["Jan","Feb","Mar","April","May","June"]
visitors = [11000,12000,15000,12500,25000,17000]
bars = plt.bar(month,visitors,width=0.5)
for i in bars:
    yc = i.get_height()
    plt.text(i.get_x(),yc+150,f"{yc}")
plt.show()
```



Python Programming - 2301CS404

Lab - 13

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 Student:
    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 = Student("John Doe", 16, "A")
student2 = Student("Jane Smith", 17, "B+")
student3 = Student("Alice Brown", 15, "A-")

print("Student 1 Info:")
student1.display_info()

print("\nStudent 2 Info:")
student2.display_info()

print("\nStudent 3 Info:")
student3.display_info()
```

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 (e.g., Savings, Checking): ")
        self.Account_Balance = float(input("Enter Account Balance: "))

    def DisplayAccountDetails(self):
        print("\nAccount 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}")

def main():
    print("Bank Account Creation\n")
```

```
    account = Bank_Account('', '', '', '', 0.0)

    account.GetAccountDetails()

    account.DisplayAccountDetails()

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

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

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

print(f"Area of the circle: {circle.area():.2f}")
print(f"Perimeter (Circumference) of the circle: {circle.perimeter():.2f}")
```

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_employee_info(self, name=None, age=None, salary=None):
        if name is not None:
            self.name = name
        if age is not None:
            self.age = age
```

```
            if salary is not None:
                self.salary = salary

        def display_employee_info(self):
            print("\nEmployee Details:")
            print(f"Name: {self.name}")
            print(f"Age: {self.age}")
            print(f"Salary: {self.salary}")

    def main():
        employee = Employee("John Doe", 30, 50000)

        employee.display_employee_info()

        employee.update_employee_info(name="Jane Smith", age=32, salary=55000)

        employee.display_employee_info()

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

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

```
class BankAccount:
    def __init__(self, account_holder, initial_balance=0):
        self.account_holder = account_holder
        self.balance = initial_balance

    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
            print(f"Deposited: ${amount}")
        else:
            print("Deposit amount must be greater than 0.")

    def withdraw(self, amount):
        if amount > 0 and amount <= self.balance:
            self.balance -= amount
            print(f"Withdrew: ${amount}")
        elif amount > self.balance:
            print("Insufficient balance.")
        else:
            print("Withdrawal amount must be greater than 0.")

    def check_balance(self):
        print(f"Current balance: ${self.balance}")
```

```
account_holder = input("Enter account holder's name: ")
account = BankAccount(account_holder)

account.deposit(1000)
account.check_balance()

account.withdraw(500)
account.check_balance()

account.withdraw(600)
account.check_balance()

account.deposit(200)
account.check_balance()
```

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, item_name, price, quantity):
        if item_name in self.items:
            print(f"Item {item_name} already exists in inventory. Use 'update_item' to updat
        else:
            self.items[item_name] = {'price': price, 'quantity': quantity}
            print(f"Item '{item_name}' added successfully to the inventory.")

    def remove_item(self, item_name):
        if item_name in self.items:
            del self.items[item_name]
            print(f"Item '{item_name}' removed successfully from the inventory.")
        else:
            print(f"Item '{item_name}' not found in inventory.")

    def update_item(self, item_name, price=None, quantity=None):
        if item_name in self.items:
            if price is not None:
                self.items[item_name]['price'] = price
            if quantity is not None:
                self.items[item_name]['quantity'] = quantity
            print(f"Item '{item_name}' updated successfully.")
        else:
            print(f"Item '{item_name}' not found in inventory.")

    def display_inventory(self):
        if not self.items:
            print("Inventory is empty.")
```

```
            else:
                print("\nInventory Details:")
                for item_name, details in self.items.items():
                    print(f"Item Name: {item_name}, Price: {details['price']}, Quantity: {detail

    def main():
        inventory = Inventory()

        inventory.add_item("Laptop", 1000, 10)
        inventory.add_item("Smartphone", 500, 25)

        inventory.display_inventory()

        inventory.update_item("Laptop", price=950, quantity=15)

        inventory.remove_item("Smartphone")

        inventory.display_inventory()

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

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

```
class Car:
    def __init__(self, make, model, year, color):
        self.make = make
        self.model = model
        self.year = year
        self.color = color

    def display_info(self):
        print(f"Car Information:")
        print(f"Make: {self.make}")
        print(f"Model: {self.model}")
        print(f"Year: {self.year}")
        print(f"Color: {self.color}")

car1 = Car("Toyota", "Corolla", 2020, "Blue")
car2 = Car("Honda", "Civic", 2018, "Red")
car3 = Car("Tesla", "Model 3", 2022, "Black")

print("Car 1 Info:")
car1.display_info()

print("\nCar 2 Info:")
car2.display_info()
```

```
print("\nCar 3 Info:")
car3.display_info()
```

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):
        self.attendance_days = int(input(f"Enter the number of days {self.student_name} was

    def generate_certificate(self):
        if self.attendance_days >= 75:
            certificate_status = "Eligible"
        else:
            certificate_status = "Not Eligible"

        print("\nCertificate")
        print(f"Student Name: {self.student_name}")
        print(f"Principal: {StudentKit.principal_name}")
        print(f"Attendance: {self.attendance_days} days")
        print(f"Certificate Status: {certificate_status}")

student_name = input("Enter the student's name: ")
student = StudentKit(student_name)

student.attendance()

student.generate_certificate()
```

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=0, minute=0):
        self.hour = hour
        self.minute = minute

    def addition(self, other_time):
        total_minutes = self.minute + other_time.minute
        total_hours = self.hour + other_time.hour

        if total_minutes >= 60:
            total_hours += total_minutes // 60
            total_minutes = total_minutes % 60

        return Time(total_hours, total_minutes)

    def display(self):
        return f"{self.hour} hours and {self.minute} minutes"

def main():
    time1 = Time(2, 50)  # 2 hours and 50 minutes
    time2 = Time(3, 45)  # 3 hours and 45 minutes

    print(f"Time 1: {time1.display()}")
    print(f"Time 2: {time2.display()}")

    result_time = time1.addition(time2)

    print(f"Result of addition: {result_time.display()}")

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

Python Programming - 2301CS404

Lab - 13

Continued..

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

```
class ractangle:
    def __init__(self,a,b):
        self.h=a
        self.w=b
    def area(self):
        area=self.h*self.w
        print("Area of ractangle :",area)

ob=ractangle(10,30)
ob.area()
```

```
Area of ractangle : 300
```

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,a):
        self.h=a

    def area(self):
        area=self.h**2
        print("Area =",area)

ob=square(10)
ob.area()
```

```
Area = 100
```

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 ractangle:
    def __init__(self,a,b):
        self.h=a
        self.w=b

    def area(self):
        area=self.h*self.w
        self.output(area)

    def output(self,x):
        print("Area of ractangle is ",x)

    def define(self):
```

```
        if(self.h==self.w):
            print("THIS IS SQUARE.")
            print("Area of square :",self.h*self.w)
        else:
            self.area()

ob=ractangle(12,12)
ob.define()
```

```
THIS IS SQUARE.
Area of square : 144
```

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  # Private attribute

    # Getter method to access the private attribute
    def get_side(self):
        return self.__side

    # Setter method to modify the private attribute
    def set_side(self, side):
        if side > 0:  # Ensuring side is positive
            self.__side = side
        else:
            print("Side length must be positive.")

# Example usage
square = Square(5)

# Accessing the private attribute using the getter method
print("Side of the square:", square.get_side())

# Modifying the private attribute using the setter method
square.set_side(10)
print("New side of the square:", square.get_side())

# Trying to set a negative side length
square.set_side(-3)  # This will print an error message
```

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 that gets profit from the user
class Profit:
    def __init__(self):
        self.profit = 0

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

# Class Loss that gets loss from the user
class Loss:
    def __init__(self):
        self.loss = 0

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

# Class BalanceSheet that inherits from both Profit and Loss
class BalanceSheet(Profit, Loss):
    def __init__(self):
        Profit.__init__(self)
        Loss.__init__(self)
        self.balance = 0

    # Method to calculate the balance (profit - loss)
    def getBalance(self):
        self.balance = self.profit - self.loss

    # Method to print the balance
    def printBalance(self):
        print(f"Profit: {self.profit}")
        print(f"Loss: {self.loss}")
        print(f"Balance: {self.balance}")

# Example usage
balance_sheet = BalanceSheet()

# Get profit and loss from the user
```

```
balance_sheet.getProfit()
balance_sheet.getLoss()

# Calculate and print the balance
balance_sheet.getBalance()
balance_sheet.printBalance()
```

15) WAP to demonstrate all types of inheritance.

```
# Single Inheritance
class Animal:
    def speak(self): print("Animal speaks")

class Dog(Animal):
    def speak(self): print("Dog barks")

# Multiple Inheritance
class Bird:
    def fly(self): print("Bird flies")
class Fish:
    def swim(self): print("Fish swims")
class Duck(Dog, Bird, Fish): pass

# Multilevel Inheritance
class Vehicle:
    def drive(self): print("Vehicle drives")
class Car(Vehicle): pass
class SportsCar(Car): pass

# Hierarchical Inheritance
class Shape:
    def draw(self): print("Shape drawn")
class Circle(Shape): pass
class Rectangle(Shape): pass

# Hybrid Inheritance
class Parent:
    def greet(self): print("Hello from Parent")
class Child(Parent): pass
class GrandChild(Child): pass

Dog().speak()
Duck().speak()
Duck().fly()
SportsCar().drive()
Circle().draw()
GrandChild().greet()
```

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.

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

    def __str__(self):
        return f"Name: {self.name}, Age: {self.age}"

# Child class Employee that inherits from Person
class Employee(Person):
    def __init__(self, name, age, salary):
        # Calling the parent's __init__ method using super()
        super().__init__(name, age)
        self.salary = salary

    def __str__(self):
        # Returning the string representation including salary
        return f"{super().__str__()}, Salary: {self.salary}"

# Creating an instance of Employee
employee = Employee("Alice", 30, 50000)

# Print the details of the employee
print(employee)
```

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.

```
from abc import ABC, abstractmethod

# Parent class Shape with an unimplemented draw method
class Shape(ABC):
    @abstractmethod
    def draw(self):
        pass

# Child class Rectangle that implements the draw method
class Rectangle(Shape):
    def draw(self):
        print("Drawing a rectangle")

# Child class Circle that implements the draw method
class Circle(Shape):
    def draw(self):
        print("Drawing a circle")

# Child class Triangle that implements the draw method
class Triangle(Shape):
    def draw(self):
        print("Drawing a triangle")

# Create a list of Shape objects
shapes = [Rectangle(), Circle(), Triangle()]

# Iterate through the list and call the draw method on each object
for shape in shapes:
    shape.draw()
```