1.**Basics of Python and Python Modules.**

* 1. **Write a python program to find the best of two test average marks out of three test’s marks accepted from the user.**

m1 = int(input("Enter marks for test1 : "))

m2 = int(input("Enter marks for test2 : "))

m3 = int(input("Enter marks for test3 : "))

if m1 <= m2 and m1 <= m3:

avgMarks = (m2+m3)/2

elif m2 <= m1 and m2 <= m3:

avgMarks = (m1+m3)/2

elif m3 <= m1 and m2 <= m2:

avgMarks = (m1+m2)/2

print("Average of best two test marks out of three test’s marks is", avgMarks);

**Output**

Enter marks for test1 : 45

Enter marks for test2 : 39

Enter marks for test3 : 48

Average of best two test marks out of three test’s marks is 46.5

**1.2.Develop a Python program to check whether a given number is palindrome or not and also count the number of occurrences of each digit in the input number.**

val = int(input("Enter a value : "))

str\_val = str(val)

if str\_val == str\_val[::-1]:

print("Palindrome")

else:

print("Not Palindrome")

for i in range(10):

if str\_val.count(str(i)) > 0:

print(str(i),"appears", str\_val.count(str(i)), "times");

**Output**

Enter a value : 1234234

Not Palindrome

1 appears 1 times

2 appears 2 times

3 appears 2 times

4 appears 2 times

Enter a value : 12321

Palindrome

1 appears 2 times

2 appears 2 times

3 appears 1 times

**1.3 Fibonacci Sequence**

**Defined as a function F as Fn = Fn-1 + Fn-2. Write a Python program which accepts a value for N (where N >0) as input and pass this value to the function. Display suitable error message if the condition for input value is not followed.**

def f(n):

if n<=1:

return n

else:

return f(n-1)+f(n-2)

num=int(input("Number of terms : "))

for i in range(num):

print(f(i))

**Output:**

Number of terms : 9

0

1

1

2

3

5

8

13

21

**1.4 Develop a python program to convert binary to decimal, octal to hexadecimal using functions.**

def bin2Dec(val):

rev=val[::-1]

dec = 0

i = 0

for dig in rev:

dec += int(dig) \* 2\*\*i

i += 1

return dec

def oct2Hex(val):

rev=val[::-1]

dec = 0

i = 0

for dig in rev:

dec += int(dig) \* 8\*\*i

i += 1

list=[]

while dec != 0:

list.append(dec%16)

dec = dec // 16

nl=[]

for elem in list[::-1]:

if elem <= 9:

nl.append(str(elem))

else:

nl.append(chr(ord('A') + (elem -10)))

hex = "".join(nl)

return hex

num1 = input("Enter a binary number : ")

print(bin2Dec(num1))

num2 = input("Enter a octal number : ")

print(oct2Hex(num2))

**Output**

Enter a binary number : 10111001

185

Enter a octal number : 675

1BD

**1.5 Develop a python program to Convert Decimal to binary using functions:**

print("Enter the Decimal Number: ")

dnum = int(input())

i = 0

bnum = []

while dnum!=0:

rem = dnum%2

bnum.insert(i, rem)

i = i+1

dnum = int(dnum/2)

i = i-1

print("\nEquivalent Binary Value is:")

while i>=0:

print(end=str(bnum[i]))

i = i-1

print()

**output:**

Enter the Decimal Number:

10

Equivalent Binary Value is:

1010

**1.6 Develop a python program to Convert binary to Decimal using functions:**

print("Enter the Binary Number: ")

bnum = int(input())

dnum = 0

i = 1

while bnum!=0:

rem = bnum%10

dnum = dnum + (rem\*i)

i = i\*2

bnum = int(bnum/10)

print("\nEquivalent Decimal Value = ", dnum)

**output:**

Enter the Binary Number:

1010

Equivalent Decimal Value = 10

**ACTIVITY:**

* 1. **Write a python program to implement insertion sort and merge sort using lists.**
  2. **Develop a python program to Convert Octal to hexadecimal and hexadecimal to octal using functions:**

**2. OOP Concepts in Python Programming: Classes, Objects and Inheritance:**

* 1. **Declare a base class to calculate Resistance from voltage and current and extend the class to calculate inductance and capacitance with varying voltage and current values with respect to time**

class ResistanceCalculator:

def \_\_init\_\_(self, voltage, current):

self.voltage = voltage

self.current = current

def calculate\_resistance(self):

if self.current != 0:

return self.voltage / self.current

else:

return float('inf')

class InductanceCalculator(ResistanceCalculator):

def \_\_init\_\_(self, voltage, current):

super().\_\_init\_\_(voltage, current)

def calculate\_inductance(self, frequency):

if self.current != 0 and frequency != 0:

return self.voltage / (2 \* 3.14159 \* frequency \* self.current)

else:

return float('inf')

class CapacitanceCalculator(ResistanceCalculator):

def \_\_init\_\_(self, voltage, current):

super().\_\_init\_\_(voltage, current)

def calculate\_capacitance(self, frequency):

if self.current != 0 and frequency != 0:

return self.current / (2 \* 3.14159 \* frequency \* self.voltage)

else:

return float('inf')

# Usage example:

voltage\_value = 220 # Volts

current\_value = 0.1 # Amperes

frequency\_value = 50 # Hertz

# Calculate resistance

resistance\_calculator = ResistanceCalculator(voltage\_value, current\_value)

resistance\_result = resistance\_calculator.calculate\_resistance()

print("Resistance:", resistance\_result, "ohms")

# Calculate inductance

inductance\_calculator = InductanceCalculator(voltage\_value, current\_value)

inductance\_result = inductance\_calculator.calculate\_inductance(frequency\_value)

print("Inductance:", inductance\_result, "Henries")

# Calculate capacitance

capacitance\_calculator = CapacitanceCalculator(voltage\_value, current\_value)

capacitance\_result = capacitance\_calculator.calculate\_capacitance(frequency\_value)

print("Capacitance:", capacitance\_result, "Farads")

**2.2 By using the concept of inheritance write a python program to find the area of triangle, circle and rectangle.**

import math

class Shape:

def \_\_init\_\_(self):

self.area = 0

self.name = ""

def showArea(self):

print("The area of the", self.name, "is", self.area, "units")

class Circle(Shape):

def \_\_init\_\_(self,radius):

self.area = 0

self.name = "Circle"

self.radius = radius

def calcArea(self):

self.area = math.pi \* self.radius \* self.radius

class Rectangle(Shape):

def \_\_init\_\_(self,length,breadth):

self.area = 0

self.name = "Rectangle"

self.length = length

self.breadth = breadth

def calcArea(self):

self.area = self.length \* self.breadth

class Triangle(Shape):

def \_\_init\_\_(self,base,height):

self.area = 0

self.name = "Triangle"

self.base = base

self.height = height

def calcArea(self):

self.area = self.base \* self.height / 2

c1 = Circle(5)

c1.calcArea()

c1.showArea()

r1 = Rectangle(5, 4)

r1.calcArea()

r1.showArea()

t1 = Triangle(3, 4)

t1.calcArea()

t1.showArea()

**Output:**

The area of the Circle is 78.53981633974483 units

The area of the Rectangle is 20 units

The area of the Triangle is 6.0 units

**3.Python Programming for Probability and Statistics.**

**Generate a python program to calculate mean, median, mode, standard deviation and variance.**

import math

#marks

xi=[4,8,11,17,20,24,32]

#number of students

fi=[3,5,9,5,4,3,1]

#mean

n=len(xi)

sum\_xifi=0

sum\_fi=0

for i in range(n):

sum\_xifi+=(xi[i]\*fi[i])

sum\_fi+=fi[i]

mean=(sum\_xifi/sum\_fi)

print("Mean = ",mean)

#median

xi1=xi

for i in range(n):

for j in range(fi[i]-1):

xi1.append(xi[i])

xi\_asc=sorted(xi1)

n1=len(xi\_asc)

if n1%2==0:

median=(xi\_asc[int(n1/2)]+xi\_asc[int((n1/2)+1)])/2

else :

median=xi\_asc[(n1+1)/2]

print("Median = ",median)

#mode

max1=fi[1]

for i in range(1,n):

if fi[i]>max1:

mode=xi[i]

print("Mode = ",mode)

#varience

sum2=0

for i in range(n):

sum2+=(fi[i]\*(xi[i]-mean)\*\*2)

var=sum2/n1

print("Varience = ",var)

#Standard Deviation

std=math.sqrt(var)

print("Standard Deviation = ",std)

**3.1 Write a Python Program To Generate Random Numbers:**

# import random

import random

# prints a random value from the list

list1 = [1, 2, 3, 4, 5, 6]

print(random.choice(list1))

# prints a random item from the string

string = "striver"

print(random.choice(string))

**Output:**

6

R

**3.2 Generate a random floating-point number between 0 and 1**

import random

random\_float = random.random()

print("Random float:", random\_float)

# Generate a random integer between a given range (e.g., 1 and 100)

random\_int = random.randint(1, 100)

print("Random integer:", random\_int)

**OUTPUT:**

Random float: 0.9747897223038477

Random integer: 97

**ACTIVITY**

### **3.1 Generating a Random Number using inbuilt functions randrange(), shuffle, uniform.**

**3.4 A Python program to print all permutations using list**

from itertools import permutations

def calculate\_permutations(elements, r):

perms = permutations(elements, r)

return list(perms)

# Example usage

elements = [1, 2, 3]

r = 2

permutations = calculate\_permutations(elements, r)

from itertools import permutations

def calculate\_permutations(elements, r):

perms = permutations(elements, r)

return list(perms)

# Example usage

elements = [1, 2, 3]

r = 2

permutations = calculate\_permutations(elements, r)

print(permutations)

**Output:**

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

**A Python program to print all permutations using library function**

from itertools import permutations

# Get all permutations of [1, 2, 3]

perm = permutations([1, 2, 3])

# Print the obtained permutations

for i in list(perm):

print (i)

**Output:**

(1, 2, 3)

(1, 3, 2)

(2, 1, 3)

(2, 3, 1)

(3, 1, 2)

(3, 2, 1)

**A Python program to print all permutations of coin tossing**

import random

flips = 0

heads = 0

tails = 0

while flips < 100:

if random.randint(1,2) == 1:

heads += 1

else:

tails += 1

flips += 1

print("you got ", heads," heads, and ", tails," tails!")

input ("exit")

**output:**

48 heads, and 52 tails!

**3.5 A Python program to print all combinations**

**from itertools import combinations**

**def calculate\_combinations(elements, r):**

**combos = combinations(elements, r)**

**return list(combos)**

**# Example usage**

**elements\_str = input("Enter a list of space-separated integers: ")**

**elements = [int(num) for num in elements\_str.split()]**

**r = int(input("Enter the value of r: "))**

**combinations = calculate\_combinations(elements, r)**

**print(combinations)**

**output:**

Enter a list of space-separated integers: 8 6 7

Enter the value of r: 2

[(8, 6), (8, 7), (6, 7)]

**A Python program to print all combinations of the dice dial using recursion and memoization:**

**Algorithm:**

1. If k = 1, create a list with all possible rolls of one die and return it.
2. If k > 1 and the list of all possible rolls of k-1 dice has already been computed, for each combination of rolls of k-1 dice, add all possible rolls of 3.
3. .one die to get all possible rolls of k dice.
4. Store the result in a memoization table to avoid recomputing already computed values.
5. Return the list of all possible rolls of k dice.

def dice\_combinations(k, memo={}):

    if k == 1:

        memo[1] = [(i,) for i in range(1, 7)]

        return memo[1]

    elif k in memo:

        return memo[k]

    else:

        prev\_res = dice\_combinations(k-1, memo)

        res = []

        for comb in prev\_res:

            for j in range(1, 7):

                res.append(comb + (j,))

        memo[k] = res

        return res

k = 2

print(dice\_combinations(k))

**output:**

[(1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (2, 1), (2, 2), (2, 3), (2, 4), (2, 5), (2, 6), (3, 1), (3, 2), (3, 3), (3, 4), (3, 5), (3, 6), (4, 1), (4, 2), (4, 3), (4, 4), (4, 5), (4, 6), (5, 1), (5, 2), (5, 3), (5, 4), (5, 5), (5, 6), (6, 1), (6, 2), (6, 3), (6, 4), (6, 5), (6, 6)]

**ACTIVITY:**

* 1. A Python program to print- all permutations of given length
  2. A Python program to print all combinations of given length
  3. A Python program to print all combinations of given length with unsorted input.
  4. Permutation of string:
  5. The code for finding permutations of fixed length is given below:
  6. Find the permutation by defining the length of the permutation.
  7. Let’s find the combination of a sequence using python.

## Combination of String

## **Combination with Replacement**

### Combination of Numeric Set

## [Permutations of Numeric data](https://www.digitalocean.com/community/tutorials/permutation-and-combinatios-in-python#permutations-of-numeric-data)

* 1. Combinations for letters in a word

**5. Sine Wave Generation**

**5.1**

import math

import numpy as np

import matplotlib.pyplot as plt

def generate\_sine\_wave(freq, amplitude, duration, sampling\_rate):

num\_samples = int(duration \* sampling\_rate)

time\_period = 1.0 / sampling\_rate

time\_values = np.arange(0, duration, time\_period)

sine\_wave = amplitude \* np.sin(2 \* np.pi \* freq \* time\_values)

return time\_values, sine\_wave

# Example usage

frequency = 5 # Frequency of the sine wave in Hz

amplitude = 1.0 # Amplitude of the sine wave

duration = 2 # Duration of the sine wave in seconds

sampling\_rate = 44100 # Sampling rate (number of samples per second)

time\_values, sine\_wave = generate\_sine\_wave(frequency, amplitude, duration, sampling\_rate)

# Plot the sine wave

plt.figure(figsize=(8, 4))

plt.plot(time\_values, sine\_wave)

plt.xlabel('Time (seconds)')

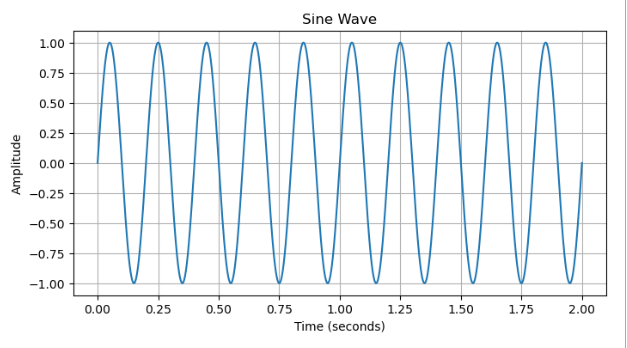
plt.ylabel('Amplitude')

plt.title('Sine Wave')

plt.grid(True)

plt.show()

**Output:**

****

**Program to solve difference equations using unilateral z transform**

import numpy as np

import matplotlib.pyplot as plt

def unilateral\_z\_transform\_difference\_equation(a, x):

n = len(x)

y = np.zeros(n)

y[0] = x[0] # Initial condition

for i in range(1, n):

y[i] = a \* y[i-1] + x[i]

return y

# Define the difference equation parameters

a = 0.5

# Generate an input sequence

n\_samples = 10

x = np.ones(n\_samples) # Impulse sequence

# Solve the difference equation using the unilateral Z-transform

y = unilateral\_z\_transform\_difference\_equation(a, x)

# Print the output sequence

print("Output sequence (y):", y)

# Plot the input and output sequences

plt.stem(range(n\_samples), x, basefmt="b-", linefmt="b-", markerfmt="bo", label="Input x[n]")

plt.stem(range(n\_samples), y, basefmt="r-", linefmt="r-", markerfmt="ro", label="Output y[n]")

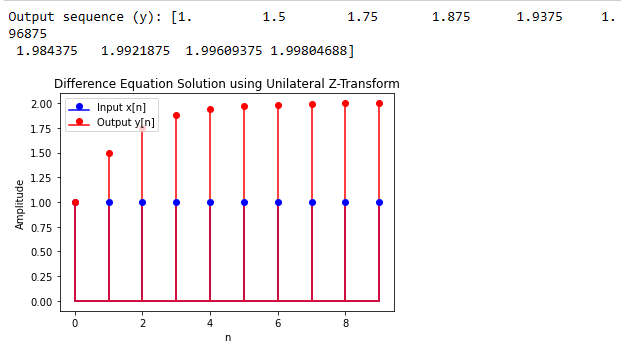
plt.xlabel("n")

plt.ylabel("Amplitude")

plt.title("Difference Equation Solution using Unilateral Z-Transform")

plt.legend()

plt.show()



**pole zero plot of transfer function using python**

import numpy as np

import matplotlib.pyplot as plt

from scipy import signal

numerator\_coeffs = [5, 2] # This represents: 5s + 2

denominator\_coeffs = [3, 2, 5] # This represents: 3s^2 + 2s + 5

transfer\_function = signal.TransferFunction(numerator\_coeffs, denominator\_coeffs)

poles = transfer\_function.poles

zeros = transfer\_function.zeros

plt.figure(figsize=(8, 6))

plt.scatter(np.real(poles), np.imag(poles), marker='x', color='red', label='Poles')

plt.scatter(np.real(zeros), np.imag(zeros), marker='o', color='blue', label='Zeros')

plt.axhline(0, color='black', linewidth=0.5)

plt.axvline(0, color='black', linewidth=0.5)

plt.xlabel('Real')

plt.ylabel('Imaginary')

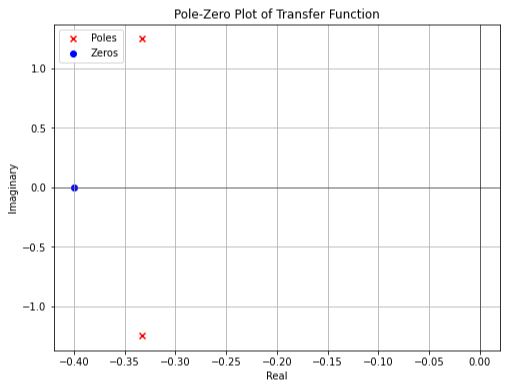
plt.title('Pole-Zero Plot of Transfer Function')

plt.legend()

plt.grid()

plt.show()

**Output:**



**Program to show bode plot for a given second order system.**

import numpy as np

import matplotlib.pyplot as plt

from scipy import signal

# Define the transfer function

numerator = [1]

denominator = [1, 2, 1] # Example: Second-order system (s^2 + 2s + 1)

system = signal.TransferFunction(numerator, denominator)

# Frequency range for Bode plot

omega = np.logspace(-2, 2, 1000)

# Calculate frequency response

\_, mag, phase = signal.bode(system, omega)

# Plot Bode magnitude plot

plt.figure(figsize=(10, 6))

plt.subplot(2, 1, 1)

plt.semilogx(omega, mag)

plt.xlabel('Frequency (rad/s)')

plt.ylabel('Magnitude (dB)')

plt.title('Bode Magnitude Plot')

# Plot Bode phase plot

plt.subplot(2, 1, 2)

plt.semilogx(omega, phase)

plt.xlabel('Frequency (rad/s)')

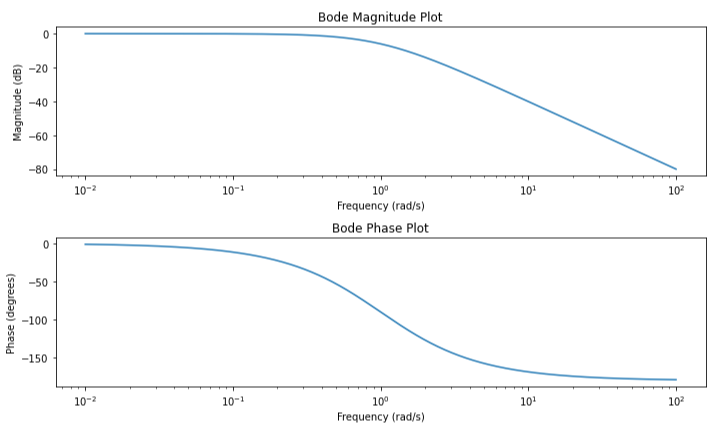
plt.ylabel('Phase (degrees)')

plt.title('Bode Phase Plot')

plt.tight\_layout()

plt.show()

**OUTPUT:**



**Program to show nyquist plot for a given second order system.**

import numpy as np

import matplotlib.pyplot as plt

from scipy import signal

# Define the transfer function

numerator = [1]

denominator = [1, 2, 1] # Example: Second-order system (s^2 + 2s + 1)

system = signal.TransferFunction(numerator, denominator)

# Frequency range for Nyquist plot

omega = np.logspace(-2, 2, 1000)

# Calculate frequency response

\_, h = signal.freqresp(system, omega)

# Extract real and imaginary parts

real\_part = np.real(h)

imag\_part = np.imag(h)

# Plot Nyquist plot

plt.figure(figsize=(8, 6))

plt.plot(real\_part, imag\_part)

plt.plot(real\_part, -imag\_part) # Plot the mirrored half

plt.xlabel('Real Part')

plt.ylabel('Imaginary Part')

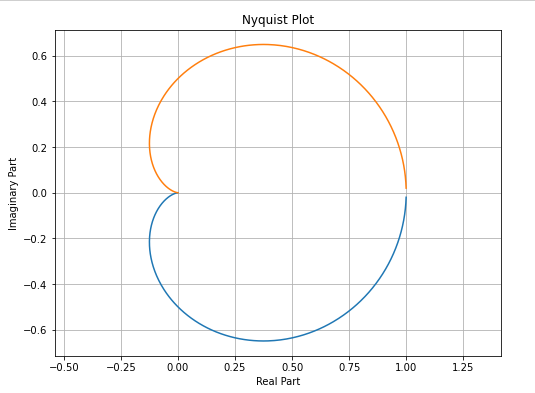
plt.title('Nyquist Plot')

plt.grid(True)

plt.axis('equal') # Equal aspect ratio

plt.show()

**Output:**



**Python program for second order linear constant coefficient differential equation solution**

import numpy as np

from scipy.integrate import solve\_ivp

import matplotlib.pyplot as plt

# Define the second-order ODE as a system of first-order ODEs

def second\_order\_ode(t, z):

y, y\_prime = z

y\_double\_prime = -2 \* y\_prime - 3 \* y

return [y\_prime, y\_double\_prime]

# Initial conditions: y(0) = 1, y'(0) = 0

initial\_conditions = [1, 0]

# Time span for integration

t\_span = (0, 10)

# Solve the ODE

solution = solve\_ivp(second\_order\_ode, t\_span, initial\_conditions, t\_eval=np.linspace(0, 10, 100))

# Extract results

t = solution.t

y = solution.y[0]

# Plot the solution

plt.figure(figsize=(8, 6))

plt.plot(t, y, label='Solution')

plt.xlabel('Time')

plt.ylabel('y')

plt.title('Solution of Second-Order ODE: y\'\' = -2y\' - 3y')

plt.legend()

plt.grid(True)

plt.show()

