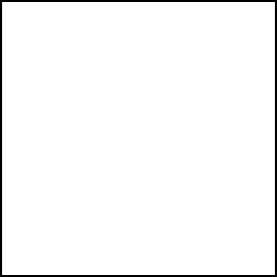
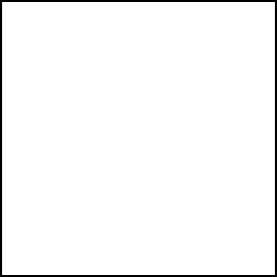
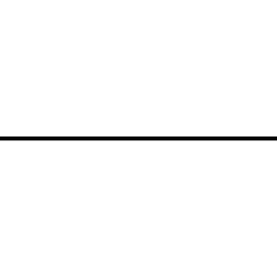
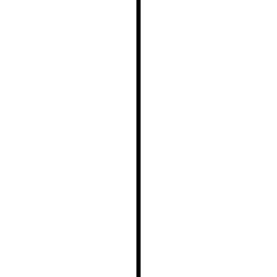
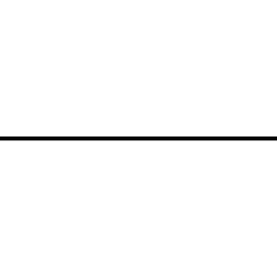
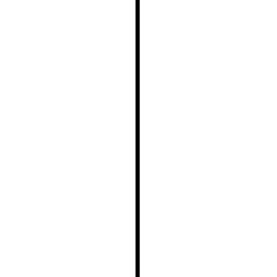
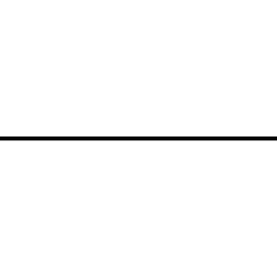
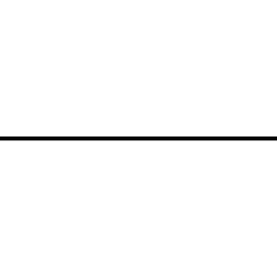
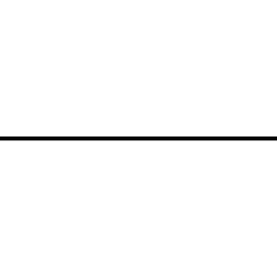
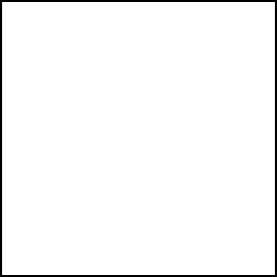
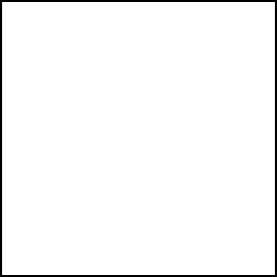
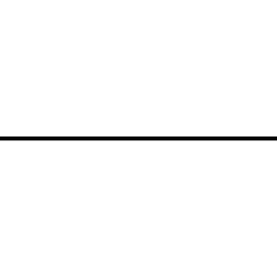
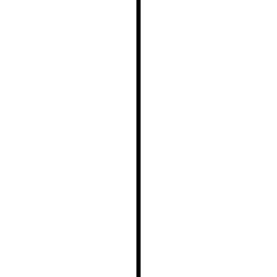
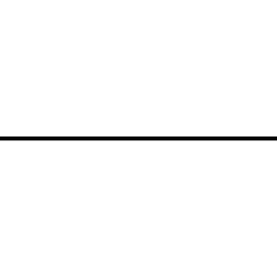
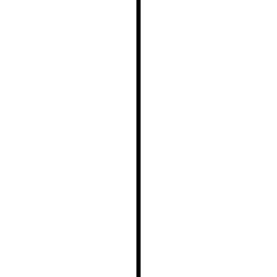
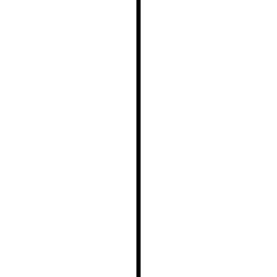
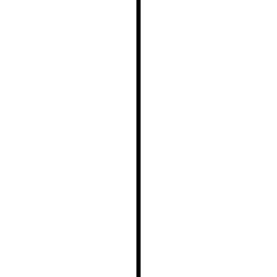
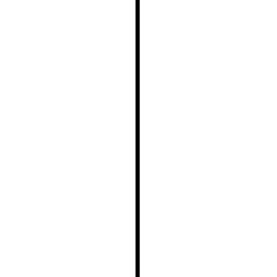
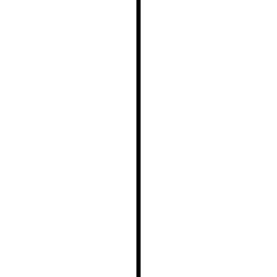
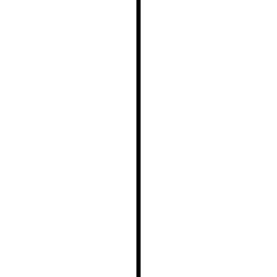
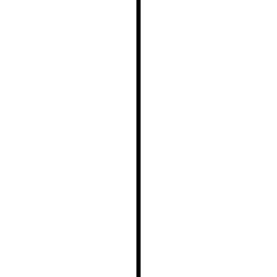
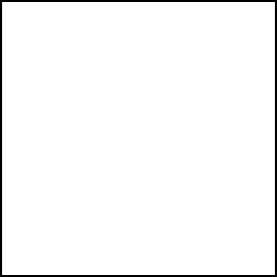
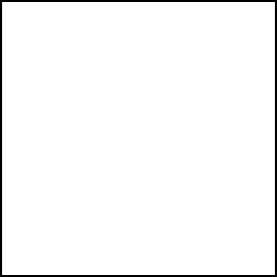
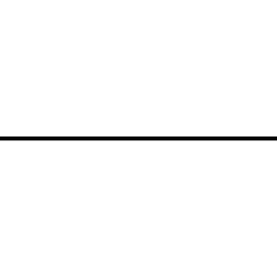
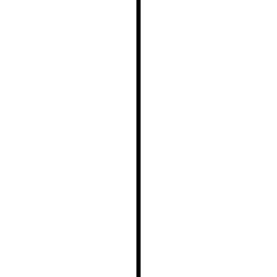
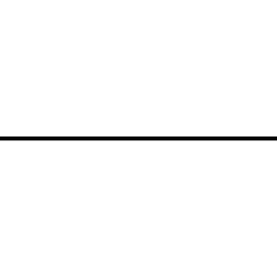
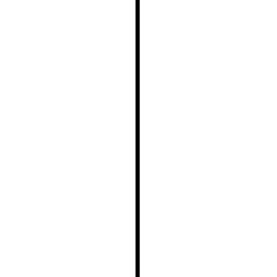
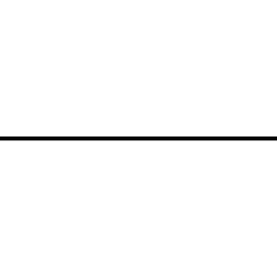
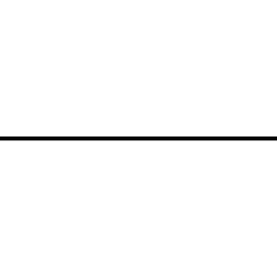
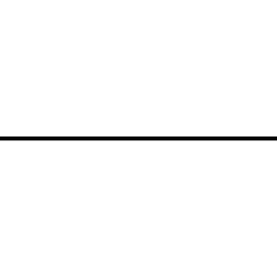
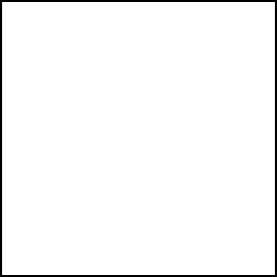
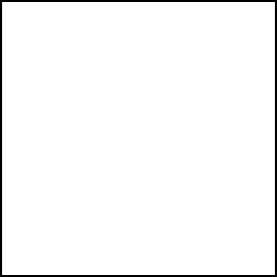
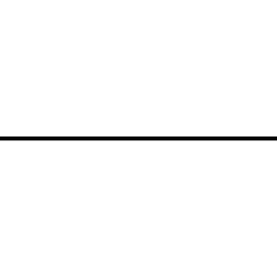
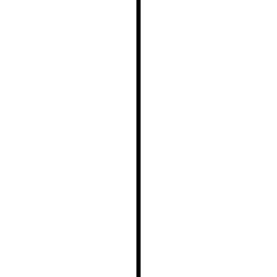
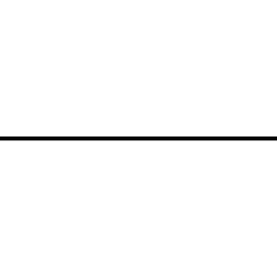
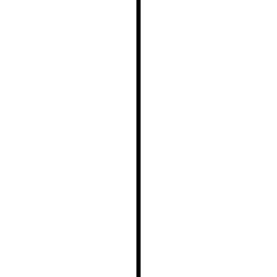
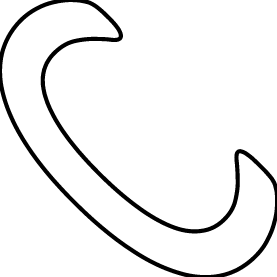
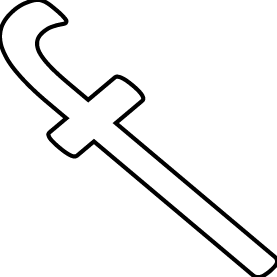
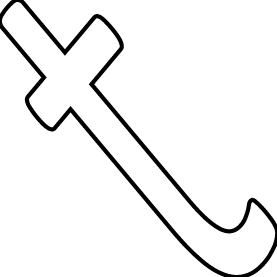
 **B M S College of Engineering Basavangudi, Bangalore**

**(Autonomous Institution under VTU, Belgaum**

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGG.

Applied Python Programming Lab Manual



**SEMESTER: IV**

**COURSE CODE: 23EC4AEAPL**

**PREPARED BY:  Dr. Lalitha S, Approved by**

**ASSISTANT. PROF, DEPT. OF EC ENGG.**

**BMSCE, BANGALORE-19 HOD,ECE**

**BMSCE, BANGALORE-19**

**ACKNOWLEDGEMENTS**

**Special thanks to the below faculties who have contributed towards bringing this Manual for students reference.**

**Dr.Feroz Morab, Assistant Professor**

**Dr. Karthikeya G.S, Assistant Professor**

**Dr.Hemavathi D, Assistant Professor**

**Dr. Shivkumar K, Assistant Professor**

**Mrs.Pooja A P, Assistant Professor**

**Mr. Eesha D, Assistant Professor**

**Yashaswini, Assistant Instructor**

**Course Syllabus**

| **CourseTitle** | **MATHEMATICSCONCEPTSUSINGPYTHON** | | | | |
| --- | --- | --- | --- | --- | --- |
| **CourseCode** | **22EC4AEMCP** | **Credits** | **1** | **L–T–P** | **0:0:1** |
| **CIE** | **50Marks(100%weightage)** | | **SEE** | **50Marks(100%weightage)** | |

**CourseOutcomes**:

Attheendofthecourse,studentswillhavetheabilityto:

| CO1 | **Understand**Pythonlibraries,OOPConceptsinPythonProgramming |
| --- | --- |
| CO2 | **Apply**differentmathematicalconcepts:ProbabilityandStatistics,Laplace,Fourierand *z*-Transformsusing pythonIDE platform (Jupyternotebook, pycharm, *etc*.) |
| CO3 | **Implement**real-timeapplicationsinsignalanalysisandcontrolsystems |

**ListofExperiments**

• **Basics of Python and Python Modules**

1. Program to find the best of two test average marks out of three test’s marks

accepted from the user.

2. Program to generate a Fibonacci sequence up to specified length.

3. Develop a program to check whether a given number/character is

Palindrome or not.

4. Develop a program to convert Decimal to binary, Octal and Hexa-decimal

and vice-versa using functions and Loops.

55

**• OOPS Concepts in Python Programming: Classes, Objects and Inheritance**

5. 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.

6. By using the concept of inheritance, write a program to find the area of

triangle, circle and rectangle.

**• Application to Field Theory**

7. Demonstration of electric field lines due to a point charge

8. Standing waves animation

**• Application to signals and systems and control systems**

9. Develop a Program for Sine Wave Generation.

10. Program to display pole – zero plot for a given transfer function.

11. Program to solve a given 2nd order difference equation using Z transform.

12. Program to solve a given 2nd order differential equation using

Laplace transform.

13. Program to display Bode plot for a given second order system.

14. Program to display Nyquist plot for a given second order system.

**ReferenceBooks**:

1. “Python Cookbook”, David Beazley and Brian K. Jones, 3rd Edition, 2013, O’Reilly MediaInc.
2. “Python:TheCompleteReference”,MartinC.Brown,4thEdition,2018,McGraw-Hill.

**INSTRUCTIONS**

1. Students must and should maintain adedicated observation book for lab, which will be continuous evaluated on a weekly basis.
2. The programs executed in the lab should be documented and the record must be submitted.
3. The Activity programs should be practiced by students and a printed report of the same should be submitted.

**Note:Uniqueness in data, variables and programming logic are expected from each Individual. Else it will be treated as Malpractice.**

**EVALUATION PATTERN FOR CIE– 50 Marks**

* **Continuous Internal evaluation on a weekly basis -25 Marks**

| Programs execution with expected output | 10 Marks |
| --- | --- |
| Documenting executed programs in observation book | 5 Marks |
| Hand written record [list of lab programs] and printed copy of report [Activity programs] | 5 + 5 Marks |
| Total | **25 Marks** |

* **Internal Lab test during end of semester – 25 Marks**

| Program write up | 10 Marks |
| --- | --- |
| Programs execution | 10 Marks |
| Viva on Python programming concepts | 5 Marks |
| Total | **25 Marks** |

**EVALUATION PATTERN FOR SEE**

| Program write up | 8 Marks |
| --- | --- |
| Programs execution[ two programs] | 35 Marks |
| Viva on Python programming concepts | 7 Marks |
| Total | **50 Marks** |

**ACTIVITY PROGRAMS**

1. Develop a program to convert Octal to Hexadecimal and Hexadecimal to Octal.
2. Write a python program to implement insertion sort and merge sort using lists.
3. Generating a Random Number using inbuilt functions randrange (), shuffle, uniform.
4. Program to find and print all permutations of a given sequence [ Integers/String] using list and functions
5. Program to find and print all Combinations of a given sequence [ Integers/String] using list and functions
6. Develop a Program for Cos Wave Generation.
7. Program to solve a given 1st order difference equation using Z transform.
8. Program to solve a given 1st order differential equation using Laplace transform

9. Program to calculate mean, median, mode, standard deviation and variance.

10. Program To Generate Random Numbers:

* 1. From a given list of specified Random Numbers
  2. Random floating-point number between 0 and 1
  3. Random integer between a given range (e.g., 1 and 100)

11. Program to print- all permutations for a given length of sequence

* 1. Using List b. Using Library functions

12. Program to print all permutations of coin tossing for a given number of

flips.

13. Program to print all combinations of the dice using recursion and memori

zation.

**INSTRUCTIONS**

1. Students must and should maintain an observation book which will be evaluated on a continuous basis.
2. The programs executed in the lab should be documented and the record should be submitted.
3. The Activity programs should be practiced by students and a printed report of the same should be submitted.

Note: Uniqueness in data and variables are expected from each student.

1. **Basics of Python and PythonModules.**
2. 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 m3 <= 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. Program to generate a Fibonacci sequence up to a specified length.

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. a) Develop a program to check whether a given number is Palindrome or not.

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

Enter a value : 12321

Palindrome

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")

3.b) Develop a program to check whether a given character is Palindrome or not.

val = input("Enter a char: ")

char\_val = str(val)

# Check if the string is a palindrome

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

print("Palindrome")

else:

print("Not Palindrome")

**Output:**

Enter a char: malayalam

Palindrome

Enter a char: convert

Not Palindrome

1. Develop a program to convert

a)Binary to Decimal,

b)Decimal to binary

c) Decimal to octal

d)Octal to decimal

e)Decimal to hexadecimal

f)Hexadecimal to decimalusing functions and Loops.

a). Binary to Decimal

def bin2dec(n):

if n<=1:

return n

else:

return ((n%10)+2\*bin2dec(n//10))

num=int(input("Enter binary number : "))

dec=bin2dec(num)

print("Decimal Equivalent Number : ",dec)

**Output:** Enter the binary number: 1010

Decimal equivalent number:10

b). Decimal to Binary

def dec2bin(n):

if n<=1:

return str(n)

else:

return (str((n%2))+str(dec2bin(n//2)))

num=int(input("Enter a deciaml number : "))

bin1=int(dec2bin(num)[::-1])

print("Decial Binary Equivalent Number : ",bin1)

**Output:**Enter a deciamlnumber : 34

Decial Binary Equivalent Number :100010

c). Decimal to Octal

def dec2oct(n):

if n<8:

return str(n)

else:

return (str(n%8)+str(dec2oct(n//8)))

num=int(input("Enter any Decimal Number : "))

oct1=int(dec2oct(num)[::-1])

print("Equivalent Octal Number : ",oct1)

**Output:**Enter any Decimal Number: 18

Equivalent Octal Number : 22

d). Octal to Decimal

def oct2dec(n):

if n<8:

return n

else:

return ((n%10)+8\*oct2dec(n//10))

num=int(input("Enter any Octal Number : "))

dec=oct2dec(num)

print("Equivalent Decimal Number : ",dec)

**Output**: Enter any Octal Number : 16

Equivalent Decimal Number : 14

e). Hexadecimal to Decimal

def hex2dec(m):

n=m[::-1]

l=len(n)

k=0

sum1=0

for i in range(l):

if ord(n[i])>=65 and ord(n[i])<=70:

sum1+=((ord(n[i])-65+10)\*16\*\*(k))

else:

sum1+=(int(n[i])\*16\*\*(k))

k+=1

return sum1

num=input("Enter any Hexadecimal Number : ")

dec=hex2dec(num)

print("Equivalent Decimal Number : ",dec)

**Output:**Enter any Hexadecimal Number : 64

Equivalent Decimal Number : 100

f). Decimal to Hexadecimal

def dec2hex(n):

if n<16:

if n%16>9:

return (chr(55+n))

else:

return str(n)

else:

if n%16>9:

return (chr(55+(n%16))+dec2hex(n//16))

else:

return (str(n%16)+dec2hex(n//16))

num=int(input("Enter any Decimal Number : "))

hex1=(dec2hex(num)[::-1])

print("Equivalent Hexadecimal Number : ",hex1)

**Output:**Enter any Decimal Number : 26

Equivalent Hexadecimal Number : 1A

**II OOPSConceptsinPythonProgramming:Classes,Objects**

**andInheritance.**

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

defcalculate\_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)

defcalculate\_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)

defcalculate\_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")

**Output:**

Resistance = 2200.0 ohms

Inductance = 0.0022 henries

Capacitance = 0.000144 farads

1. By using the concept of inheritance compose a program to find the area of triangle, circle and rectangle.

import math

class Shape:

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

self.area = 0

self.name = ""

defshowArea(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

defcalcArea(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

defcalcArea(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

defcalcArea(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

**III Application to Field Theory.**

1. Develop a program to demonstratethe electricfield linesdue toa pointcharge

import numpy as np

import matplotlib.pyplot as plt

# Constants

k = 8.99e9 # Coulomb's constant, N m²/C²

q = 1e-9 # Charge, C

# Create a grid of points

x = np.linspace(-10, 10, 400)

y = np.linspace(-10, 10, 400)

X, Y = np.meshgrid(x, y)

# Calculate electric field components

defelectric\_field(q, r0, x, y):

"""Return the electric field vector E = (Ex, Ey) due to a point charge q at r0."""

den = np.hypot(x - r0[0], y - r0[1])\*\*3 # Calculate the denominator

Ex = k \* q \* (x - r0[0]) / den

Ey = k \* q \* (y - r0[1]) / den

return Ex, Ey

# Position of the point charge

r0 = np.array([0.0, 0.0])

# Calculate the electric field components

Ex, Ey = electric\_field(q, r0, X, Y)

# Plotting

fig, ax = plt.subplots(figsize=(8, 8))

color = np.log(np.hypot(Ex, Ey))

# Plot electric field lines

ax.streamplot(X, Y, Ex, Ey, color=color, linewidth=1, cmap='inferno', density=2)

# Add point charge to the plot

ax.plot(r0[0], r0[1], 'ro')

# Labels and title

ax.set\_xlabel('x')

ax.set\_ylabel('y')

ax.set\_title('Electric Field Lines due to a Point Charge')

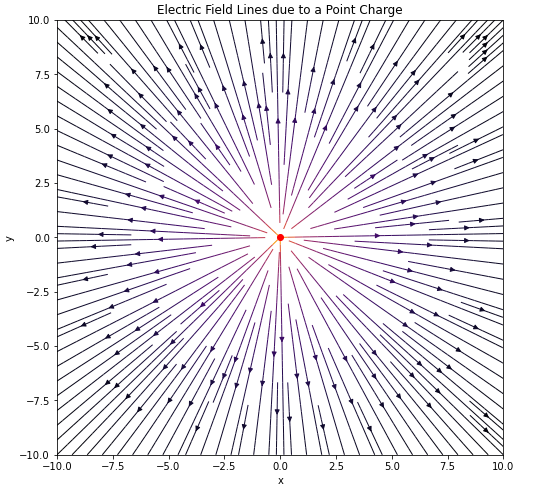
# Set limits

ax.set\_xlim(-10, 10)

ax.set\_ylim(-10, 10)

plt.show()

**Output:**

****

1. Develop a program on Standing waves animation

import numpy as np

import matplotlib.pyplot as plt

import matplotlib.animation as animation

# Parameters

L = 10 # Length of the string

k = 2 \* np.pi / L # Wave number

omega = 2 \* np.pi # Angular frequency

A = 1 # Amplitude

x = np.linspace(0, L, 1000) # Position along the string

# Create a figure and axis

fig, ax = plt.subplots()

line, = ax.plot(x, np.zeros\_like(x), lw=2)

# Set up plot limits and labels

ax.set\_xlim(0, L)

ax.set\_ylim(-2 \* A, 2 \* A)

ax.set\_xlabel('Position')

ax.set\_ylabel('Displacement')

ax.set\_title('Standing Waves')

# Function to update the frame

def update(frame):

t = frame / 30 # Time variable

y = 2 \* A \* np.sin(k \* x) \* np.cos(omega \* t) # Standing wave equation

line.set\_ydata(y)

return line,

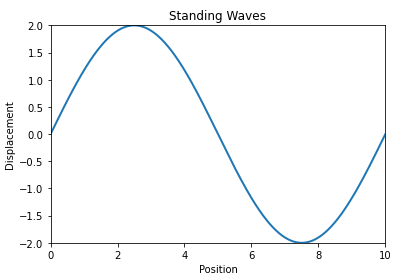
# Create the animation

ani = animation.FuncAnimation(fig, update, frames=300, interval=20, blit=True)

# Display the animation

plt.show()

**Output:**

****

**IV Application to signals and systems and controls systems.**

9. Develop a Program for Sine Wave Generation

import math

import numpy as np

import matplotlib.pyplot as plt

defgenerate\_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)')

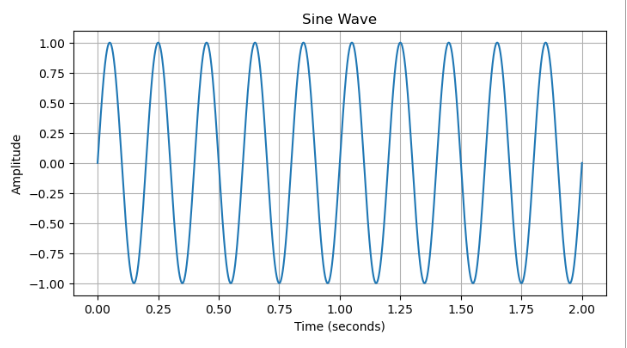
plt.ylabel('Amplitude')

plt.title('Sine Wave')

plt.grid(True)

plt.show()

**Output:**

****

10.Program to displaypole – zero plot for a given transfer function.

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')

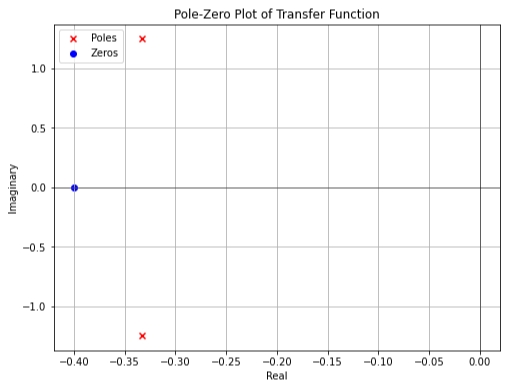
plt.title('Pole-Zero Plot of Transfer Function')

plt.legend()

plt.grid()

plt.show()

**Output:**



11.Program to solve a given 2nd order difference equationusing Z transform.

import numpy as np

import matplotlib.pyplot as plt

defunilateral\_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")

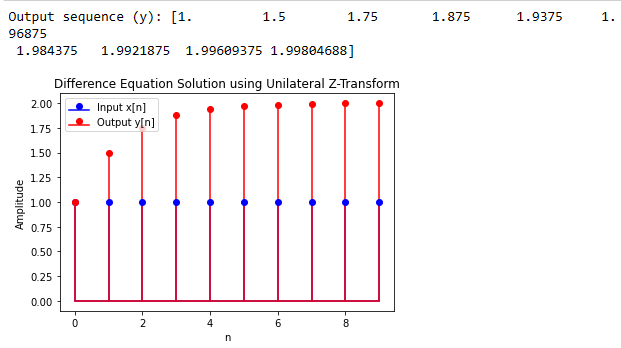
plt.ylabel("Amplitude")

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

plt.legend()

plt.show()

**Output:**



1. Program to solve a given 2ndorder differential equation using Laplace transform.

import sympy as sp

from scipy import signal

t,s=sp.symbols('t,s')

#y"+5y’+6y=0

#y(0)=0,y'(0)=2

#s^2L[y(s)]-y'(0)-y(0)+5sL[y(t)]-y(0)+6L[y(t)]

#s^2L[y(s)]-5sL[y(t)]+6L[y(t)]=y(0)+y’(0)

coeff=[1,5,6]

initial\_cond=[0,2] #y(0),y'(0)

l\_eq\_lhs=(coeff[0]\*(s\*\*2)+coeff[1]\*(s)+coeff[2])

eq\_rhs=initial\_cond[0]+initial\_cond[1]

l\_eq\_rhs=eq\_rhs/s

tf=l\_eq\_rhs/l\_eq\_lhs

l\_eq=sp.inverse\_laplace\_transform(tf,s,t)

print(l\_eq)

**Output:**

(exp(3\*t) - 3\*exp(t) + 2)\*exp(-3\*t)\*Heaviside(t)/3

**Note:**

The Heaviside step function, often denoted as H(t), is defined as:

H(t) = 0 for t < 0 h(t) for t>=0

Let’s break it down step by step:

1. Heaviside(t)/3: This term is 1/3 for t >=0 for t < 0.
2. –exp(-2\*t)*Heaviside(t):* This term is *–exp(-2t) for > = 0 and 0 for t<0.*
3. 2exp(-3t) *Heaviside(t)/3:* This term is *(2/3) \* exp(-3t) for t > = 0* and 0 for *t < 0.*

So, when t >=0, You can simplify the expression as follows:

(1/3) –exp(-2t) + (2/3) \* *exp(-3t)*

You can further simplify this expression by combining the constant terms:

(1/3) + (2/3) \* exp(-3t) *-exp(-2t)*

This is the simplified expression for your original expression when it is greater than or equal to 0.

13.Program to display Bode plot for a given second order system.

import numpy as np

import matplotlib.pyplot as plt

from 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)')

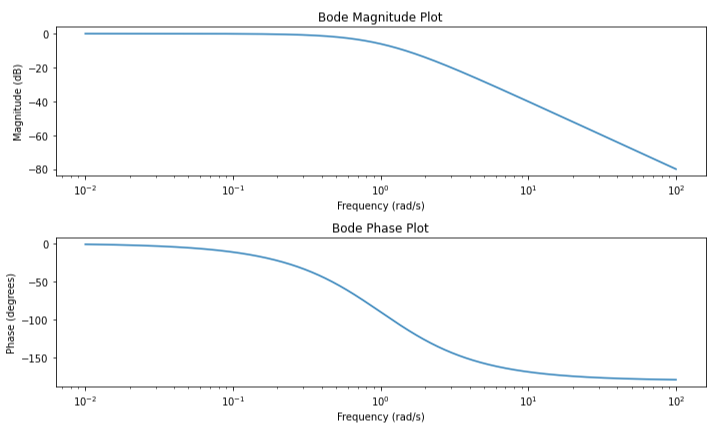
plt.ylabel('Phase (degrees)')

plt.title('Bode Phase Plot')

plt.tight\_layout()

plt.show()

**OUTPUT:**



14.Program to display 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')

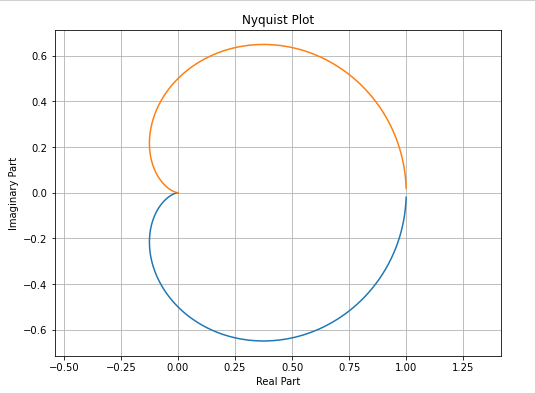
plt.title('Nyquist Plot')

plt.grid(True)

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

plt.show()

**Output:**



**ACTIVITY PROGRAMS**

1. Develop a program to convert Octal to Hexadecimal and Hexadecimal to Octal.

Octal to Hexadecimal:

def dec2hex(n):

if n<=1:

return str(n)

else:

if(n%16>9):

return ((chr(ord('A')+(n%16)-10))+str(dec2hex(n//16)))

else:

return (str(n%16)+str(dec2hex(n//16)))

def oct2dec(n):

if n<=1:

return n

else:

return ((n%10)+8\*oct2dec(n//10))

def oct2hex(n):

dec1=oct2dec(n)

hex1=dec2hex(dec1)

return hex1

num=int(input("Enter octal number : "))

print("Hexadecimal number = ",oct2hex(num)[::-1])

**Output:**

Enter octal Number : 345

Hexadecimal Number = E5

1. Write a python program to implement insertion sort and merge sort using lists.

#Insertion sort

definsertion\_sort(lst):

for i in range(1, len(lst)):

key = lst[i]

j = i - 1

while j >= 0 and key <lst[j]:

lst[j + 1] = lst[j]

j -= 1

lst[j + 1] = key

lst=eval(input("Enter the list to be sorted : "))

insertion\_sort(lst)

print("Sorted list :",lst)

Output:

Enter the list to be sorted : [33,22,1,56,78,93]

Sorted list : [1, 22, 33, 56, 78, 93]

#merge sort

defmerge\_sort(lst):

if len(lst) > 1:

mid = len(lst) // 2

left\_half = lst[:mid]

right\_half = lst[mid:]

merge\_sort(left\_half)

merge\_sort(right\_half)

i = j = k = 0

while i<len(left\_half) and j <len(right\_half):

if left\_half[i] <right\_half[j]:

lst[k] = left\_half[i]

i += 1

else:

lst[k] = right\_half[j]

j += 1

k += 1

while i<len(left\_half):

lst[k] = left\_half[i]

i += 1

k += 1

while j <len(right\_half):

lst[k] = right\_half[j]

j += 1

k += 1

lst=eval(input("Enter the list to be sorted : "))

merge\_sort(lst)

print(lst)

**Output:**

Enter the list to be sorted : [66,45,43,89,23,12]

Sorted List : [12, 23, 43, 45, 66, 89]

1. Generating a Random Number using inbuilt functions randrange (), shuffle, uniform.

#random

import random

a=random.randrange(1,100)

print("Random Number : ",a)

Sample Output:

Random Number ; 46

#shuffle

import random

l=["apple", "orange", "banana"]

print("Original List : ",l)

random.shuffle(l)

print("Shuffled List : ",l)

Sample Output:

Original List : ["apple", "orange", "banana"]

Shuffled List : ["banana", "orange", "apple"]

#uniform

import random

print(random.uniform(10,20))

**Sample Output:**

16.70368719166577

1. Program to find and print all permutations of a given sequence [Integers/String] using list and functions.

deftoString(List):

return ''.join(List)

defpermute(a, l, r):

if l == r:

print (toString(a))

else:

for i in range(l, r + 1):

a[l], a[i] = a[i], a[l]

permute(a, l + 1, r)

a[l], a[i] = a[i], a[l]

string = '123'

n = len(string)

a = list(string)

permute(a, 0, n-1)

**Output:**

123

132

213

231

321

312

1. Program to find and print all Combinations of a given sequence [ Integers/String] using list and functions

l=eval(input("Enter a list of integer : "))

r=int(input("Enter the number of terms : "))

n=len(l)

if r==1:

for i in range(n):

print(l[i])

elif r==n:

print(l)

else:

for i in range(n-r):

for j in range(n):

if i!=j:

print(l[i],l[j])

**Output:**

Enter a list of integer : [1,2,3,4]

Enter the number of terms : 2

1 2

1 3

1 4

2 1

2 3

2 4

1. Develop a Program for Cos Wave Generation.

import math

import numpy as np

import matplotlib.pyplot as plt

defgenerate\_cos\_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)

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

return time\_values, cos\_wave

frequency = 5

amplitude = 1.0

duration = 2

sampling\_rate = 44100

time\_values, cos\_wave = generate\_cos\_wave(frequency, amplitude, duration, sampling\_rate)

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

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

plt.xlabel('Time (seconds)')

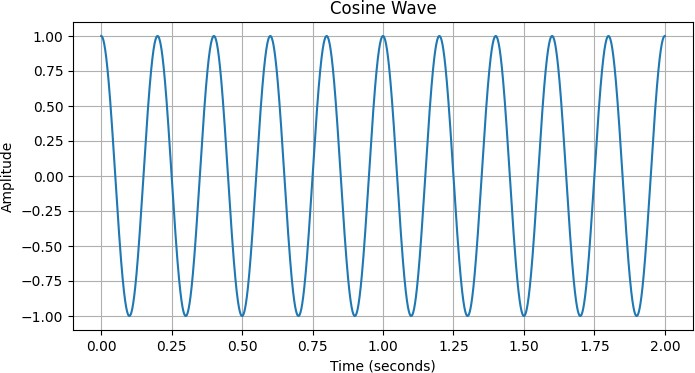
plt.ylabel('Amplitude')

plt.title('Cosine Wave')

plt.grid(True)

plt.show()

**Output:**



1. Program to solve a given 1st order difference equation using Z transform.

import numpy as np

import matplotlib.pyplot as plt

defdifference\_equation(a, x):

n = len(x)

y = np.zeros(n)

y[0] = x[0]

for i in range(1, n):

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

return y

a = 0.5

n\_samples = 10

x = np.ones(n\_samples)

y = difference\_equation(a, x)

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

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")

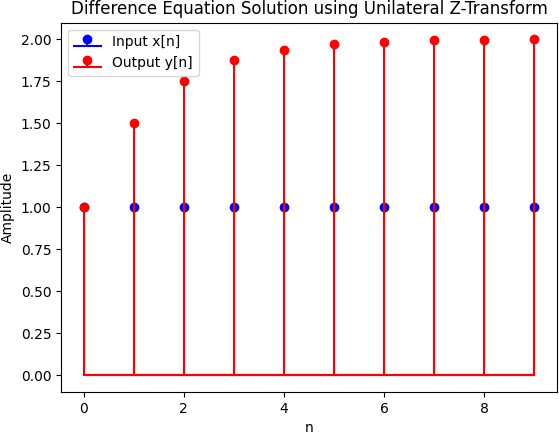
plt.ylabel("Amplitude")

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

plt.legend()

plt.show()

**Output:**



1. Program to solve a given 1st order differential equation using Laplace transform

import sympy as sp

t,s,Y=sp.symbols('t s Y')

ode=sp.Eq(sp.diff(Y,t)+3\*Y,3\*sp.exp(-4\*t))

l\_eq=sp.laplace\_transform(ode.lhs-ode.rhs,t,s,noconds=True)

y\_s=sp.solve(l\_eq,Y)[0]

solution=sp.inverse\_laplace\_transform(Y\_s,s,t,noconds=True)

print(solution)

**Output:**

DiracDelta(t) - 4\*exp(-4\*t)\*Heaviside(t)

1. 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)

**Output:**

Mean = 13.625

Median = 11

Mode = 30.453

Variance = 30.453

Standard Deviation = 5.51494

1. Program To Generate Random Numbers:
2. From a given list of specified Random Numbers

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

1. Random floating-point number between 0 and 1

import random

random\_float = random.random()

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

**Output:**

Random float: 0.9747897223038477

1. Random integer between a given range (e.g., 1 and 100)

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

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

**Output:**

Random integer: 97

11.Program to print- all permutations for a given length of sequence.

1. Using List b. Using Library functions
2. Using List

from itertools import permutations

defcalculate\_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

defcalculate\_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)]

b. Using Library functions.

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)

1. Program to print all permutations of coin tossing for a given number of flips.

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!

1. Program to print all combinations of the dice using recursion and memorization.

**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.

defdice\_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)

1. Program to solve a given 1st order differential equation using Laplace transform

import sympy as sp

t,s,Y=sp.symbols('t s Y')

ode=sp.Eq(sp.diff(Y,t)+3\*Y,3\*sp.exp(-4\*t))

l\_eq=sp.laplace\_transform(ode.lhs-ode.rhs,t,s,noconds=True)

y\_s=sp.solve(l\_eq,Y)[0]

solution=sp.inverse\_laplace\_transform(Y\_s,s,t,noconds=True)

print(solution)

**Output:**

DiracDelta(t) - 4\*exp(-4\*t)\*Heaviside(t)