PYTHON PRACTICAL FILE

SUBJECT CODE : 223601105

EXAM ROLL NO : 24236761060

NAME : TANYA RAWAT

COURSE : 761 : MASTER OF OPERATIONAL RESEARCH

DEPARTMENT/COLLEGE : DEPARTMENT OF OPERATIONAL RESEARCH,NORTH CAMPUS

SECTION : A

1. Write a program to compute the roots of a quadratic equation.

CODE:

```
import math
print("The equation is of the form ax\u00b2 + bx + c")
a = float(input("Enter coefficient a: "))
b = float(input("Enter coefficient b: "))
c = float(input("Enter coefficient c: "))
discriminant = b**2 - 4*a*c

#if determinant is > 0 , the equation has two distinct real roots.
if discriminant > 0:
   root1 = (-b + math.sqrt(discriminant)) / (2 * a)
   root2 = (-b - math.sqrt(discriminant)) / (2 * a)
   print(f"The roots are real and different: {root1:.2f} and {root2:.2f}")

#If the discriminant is equal to 0, the equation has one real repeated root.
elif discriminant == 0:
   root = -b / (2 * a)
   print(f"The roots are real and equal: {root:.2f}")

#  If the discriminant is less than 0, the equation has two complex roots.
else:
   real_part = -b / (2 * a)
   imaginary_part = math.sqrt(-discriminant) / (2 * a)
   print(f"The roots are complex: {real_part:.2f} + {imaginary_part}i and
{real_part:.2f} - {imaginary_part}i")
```

OUTPUT:

```
The equation is of the form ax² + bx + c
Enter coefficient a: 1
Enter coefficient b: 2
Enter coefficient c: -15
The roots are real and different: 3.00 and -5.00
```

---

2. WAP to play Stone, Paper, Scissors with Computer.

CODE:

```
import random

print("Playing Stone , Paper and Scissor with Computer")
user_input = int(input("\n1 for Stone\n2 for Paper \n3 for
scissor.\nEnter your choice:")) #user input


comp_input = random.randint(1,3) # computer input
print("Your choice is :", user_input)
print("Computer's choice is :", comp_input)

#here we check who wins using conditional statements
if(user_input == comp_input):
   print("Game is tied!")
elif(user_input == 1 and comp_input ==3 or user_input == 2 and
comp_input == 1 or user_input == 3 and comp_input == 2):
   print("You won the game!")
else :
   print("You lost the game!")
```

OUTPUT:

```
Playing Stone , Paper and Scissor with Computer

1 for Stone
2 for Paper
3 for scissor.
Enter your choice:2
Your choice is : 2
Computer's choice is : 2
Game is tied!
```

3. Write a program for a BMI calculator with categorization of underweight, normal weight, and overweight.

CODE:

```python
def calculate_bmi(weight, height):  #function to calculate bmi
    bmi = weight / (height ** 2)
    return bmi

def categorize_bmi(bmi): #function to categorise the bmi in three
categories
    if bmi < 18.5:
        return "Underweight"
    elif 18.5 <= bmi < 24.9:
        return "Normal weight"
    else:
        return "Overweight"

weight = int(input("Enter your weight in kg: ")) #input user's weight
height = float(input("Enter your height in meters: ")) #input user's
height

bmi = calculate_bmi(weight, height) #calling the calculate_bmi function

print(f"Your BMI is: {bmi:.2f}")
print(f"Category: {categorize_bmi(bmi)}")
```

OUTPUT:

```
Enter your weight in kg: 56
Enter your height in meters: 1.651
Your BMI is: 20.54
Category: Normal weight
```

```
Enter your weight in kg: 45
Enter your height in meters: 1.7
Your BMI is: 15.57
Category: Underweight
```

```
Enter your weight in kg: 85
Enter your height in meters: 1.75
Your BMI is: 27.76
Category: Overweight
```
---

4. Write a program to demonstrate exception handling of a ZeroDivisionError.

CODE:

```python
try:
    numerator = int(input("Enter the numerator: "))
    denominator = int(input("Enter the denominator: "))

    result = numerator / denominator
    print(f"The result of {numerator} / {denominator} is: {result}")

except ZeroDivisionError:
    print("Error: Division by zero is not allowed.")

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

```
c 1.py"
Enter the numerator: 45
Enter the denominator: 150
The result of 45 / 150 is: 0.3
```

```
Enter the numerator: 59
Enter the denominator: 0
Error: Division by zero is not allowed.
```

---

5. Write a program to demonstrate OOP using a user-defined Cuboid class.

CODE:

```
class Cuboid:
    def __init__(self,l,b,h):
        self.l = l
        self.b=b
        self.h=h
    def lid_Area(self):
        return self.l*self.b
    def perimeter(self):
        return 4*(self.l+self.b+self.h)
    def volume(self):
        return self.l*self.b*self.h

length = int(input("Enter the length of the Cuboid: "))
breadth = int(input("Enter the breadth of the Cuboid: "))
height = int(input("Enter the height of the Cuboid: "))
Obj = Cuboid(length, breadth, height)
print("Lid Area of the Cuboid is: ", Obj.lid_Area())
print("Perimerter of the Cuboid is: ", Obj.perimeter())
print("Volume of the Cuboid is: ", Obj.volume())
```

OUTPUT:

```
Enter the length of the Cuboid: 5
Enter the breadth of the Cuboid: 6
Enter the height of the Cuboid: 3
Lid Area of the Cuboid is:  30
Perimerter of the Cuboid is:  56
Volume of the Cuboid is:  90
```

6. Write a program to demonstrate inheritance in OOP using user-defined Rectangle and Cuboid classes.

CODE:

```
class Rectangle:
    def __init__(self,l,b):
        self.l = l
        self.b=b
    def Area(self):
        return self.l*self.b
```

```
    def perimeter(self):
        return 2*(self.l+self.b)

class Cuboid(Rectangle):
    def __init__(self,l,b,h):
        self.h=h
        Rectangle.__init__(self,l,b)
    def perimeter(self):
        return 4*(self.l+self.b+self.h)
    def volume(self):
        return self.l*self.b*self.h

length = int(input("Enter the length of the Cuboid/Rectangle: "))
breadth = int(input("Enter the breadth of the Cuboid/Rectangle: "))
height = int(input("Enter the height of the Cuboid: "))
Obj = Cuboid(length, breadth, height)
obj2 = Rectangle(length, breadth)
print("Perimerter of the Rectangle is: ", obj2.perimeter())
print("Area of the Rectangle is: ", obj2.Area())
print("Perimerter of the Cuboid is: ", Obj.perimeter())
print("Volume of the Cuboid is: ", Obj.volume())
```

OUTPUT:

```
Enter the length of the Cuboid/Rectangle: 4
Enter the breadth of the Cuboid/Rectangle: 5
Enter the height of the Cuboid: 6
Perimerter of the Rectangle is:  18
Area of the Rectangle is:  20
Perimerter of the Cuboid is:  60
Volume of the Cuboid is:  120
```

7. Write a program to implement inheritance. Create a class Employee and inherit two classes Manager and Clerk from Employee.

CODE:

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

    def show_details(self):
```

```
        print(f"Name: {self.name}, Age: {self.age}, Salary:
{self.salary}")

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

    def show_details(self):
        super().show_details()
        print(f"Designation: Manager")

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

    def show_details(self):
        super().show_details()
        print(f"Designation: Clerk")


print("Manager Details.\n")
manager = Manager("Alice", 40, 80000)
clerk = Clerk("Bob", 25, 30000)

manager.show_details()

print("\nClerk Details.\n")

clerk.show_details()
```

OUTPUT:

```
Manager Details.

Name: Alice, Age: 40, Salary: 80000
Designation: Manager

Clerk Details.

Name: Bob, Age: 25, Salary: 30000
Designation: Clerk
```

8. Write a program to demonstrate the demarcation of class variables and instance variables in OOP using the Employee class.

CODE:

```
class Employee:
    emp_id = 100 #class variable
    def __init__(self,name,designation,salary):
        self.name = name
        self.designation = designation
        self.salary= salary
        Employee.emp_id+=1

    def show_Details(self):
        print("\nEmployee Name:",self.name,"\nEmployee Designation:"
                ,self.designation,"\nEmployee
Salary:",self.salary,"\nEmployee Id:",Employee.emp_id)

    def total(self):
        print("Total Number of Employees:",Employee.emp_id-100)

i='yes'
while i=='yes':
    name=input("\nEnter the employee name: ")
    salary=int(input("Enter the employee salary: "))
    designation=input("Enter the employee designation: ")
    obj=Employee(name,designation,salary)
    obj.show_Details()
    obj.total()
    i = input("\nEnter yes to continue: ")
```

OUTPUT:

```
Enter the employee name: Tanya
Enter the employee salary: 65798
Enter the employee designation: Engineer

Employee Name: Tanya
Employee Designation: Engineer
Employee Salary: 65798
Employee Id: 101
Total Number of Employees: 1

Enter yes to continue: yes

Enter the employee name: Anshika
Enter the employee salary: 546789
Enter the employee designation: Data Analyst

Employee Name: Anshika
Employee Designation: Data Analyst
Employee Salary: 546789
Employee Id: 102
Total Number of Employees: 2

Enter yes to continue: no
```

---

9. Write a program to determine EOQ using various inventory models.

CODE:

```
import math
#Class to organize all four models
class EOQModels:
   def __init__(self, demand, ordering_cost, holding_cost,
production_rate=None, shortage_cost=None):
       self.demand = demand  # Annual demand (D)
       self.ordering_cost = ordering_cost  # Ordering cost per order
(S)
       self.holding_cost = holding_cost  # Holding cost per unit per
year (H)
       self.production_rate = production_rate  # Production rate (P)
for EPQ
```

```
        self.shortage_cost = shortage_cost  # Shortage cost per unit (C)
for backorder models

    def eoq(self):
        """Calculate EOQ"""
        return math.sqrt((2 * self.demand * self.ordering_cost) /
self.holding_cost)

    def epq(self):
        """Calculate EPQ"""
        if self.production_rate is None or self.production_rate <=
self.demand:
            raise ValueError("Production rate must be greater than
demand for EPQ.")
        return math.sqrt((2 * self.demand * self.ordering_cost) /
(self.holding_cost * (1 - (self.demand / self.production_rate))))

    def eoq_with_backorders_planned_shortages(self):
        """Calculate EOQ with Backorders (Planned Shortages)"""
        if self.shortage_cost is None:
            raise ValueError("Shortage cost must be provided for this
model.")
        eoq = self.eoq()
        return eoq * (1 + (self.shortage_cost / self.holding_cost))

    def eoq_with_backorders_shortages_lost(self):
        """Calculate EOQ with Backorders (Shortages Lost)"""
        if self.shortage_cost is None:
            raise ValueError("Shortage cost must be provided for this
model.")
        eoq = self.eoq()
        return eoq * (1 + (self.shortage_cost / (self.holding_cost +
self.shortage_cost)))

# Taking input for various parameters that are to be used in
calculation
demand = float(input("Enter the annual demand (D): "))
ordering_cost = float(input("Enter the ordering cost per order (S): "))
holding_cost = float(input("Enter the holding cost per unit per year
(H): "))

# Creating object for the class EOQModels
eoq_model = EOQModels(demand, ordering_cost, holding_cost)
```

```
# Calculate EOQ
#Calculates the optimal order quantity that minimizes total inventory
costs based on annual demand,
# ordering costs, and holding costs.

print(f"\nOptimal Quantity using EOQ: {eoq_model.eoq():.2f}\n")

# Calculate EPQ
#Determines the optimal production quantity when production rates
exceed demand,
#taking into account the same parameters as the EOQ model.
try:
   production_rate = float(input("Enter the production rate (P) for
EPQ: "))
   eoq_model.production_rate = production_rate
   print(f"\nOptimal Quantity using EPQ: {eoq_model.epq():.2f}\n")
except ValueError as e:
   print(e)

# Calculate EOQ with Backorders (Planned Shortages)
#Computes the optimal order quantity considering costs associated with
backorders when
# demand exceeds supply, enhancing inventory planning.
try:
   shortage_cost = float(input("Enter the shortage cost per unit (C)
for Planned Shortages: "))
   eoq_model.shortage_cost = shortage_cost
   print(f"\nOptimal Quantity using EOQ with Backorders (Planned
Shortages): {eoq_model.eoq_with_backorders_planned_shortages():.2f}\n")
except ValueError as e:
   print(e)

# Calculate EOQ with Backorders (Shortages Lost)
#Evaluates the optimal order quantity while factoring in costs related
to lost sales due to stockouts,
#providing a more comprehensive inventory cost analysis.

try:
   print(f"\nOptimal Quantity using EOQ with Backorders (Shortages
Lost): {eoq_model.eoq_with_backorders_shortages_lost():.2f}\n")
except ValueError as e:
   print(e)
```

OUTPUT:

```
Enter the annual demand (D): 1000
Enter the ordering cost per order (S): 50
Enter the holding cost per unit per year (H): 2

Optimal Quantity using EOQ: 223.61

Enter the production rate (P) for EPQ: 1200

Optimal Quantity using EPQ: 547.72

Enter the shortage cost per unit (C) for Planned Shortages: 3

Optimal Quantity using EOQ with Backorders (Planned Shortages): 559.02


Optimal Quantity using EOQ with Backorders (Shortages Lost): 357.77
```

10. Write a program to determine different characteristics using various queuing models.

CODE:

```
#Class for oraganizing the queueing models.
class QueueModel:
   def __init__(self, arrival_rate, service_rate, num_servers=1,
capacity=float('inf')):
       self.arrival_rate = arrival_rate  # λ (lambda)
       self.service_rate = service_rate    # μ (mu)
       self.num_servers = num_servers       # c (number of servers)
       self.capacity = capacity             # K (system capacity)

   def mm1(self):
       rho = self.arrival_rate / self.service_rate  # Traffic intensity
       L = rho / (1 - rho)  # Average number of customers in the system
       W = 1 / (self.service_rate - self.arrival_rate)  # Average time
spent in the system
       Lq = (rho ** 2) / (1 - rho)  # Average number in the queue
       Wq = Lq / self.arrival_rate   # Average time spent in the queue
       return L, W, Lq, Wq

   def mm_infinity(self):
       L = self.arrival_rate / self.service_rate  # Average number of
customers in the system
```

```
        W = 1 / self.service_rate  # Average time spent in the system
        Lq = 0  # No queue in M/M/∞
        Wq = 0  # No waiting time in M/M/∞
        return L, W, Lq, Wq

    def mm_c(self):
        rho = self.arrival_rate / (self.service_rate * self.num_servers)
# Traffic intensity
        Lq = ((self.arrival_rate**2) / (self.service_rate**2)) /
(self.num_servers * (1 - rho))  # Average number in queue
        L = Lq + (self.arrival_rate / self.service_rate)  # Average
number in the system
        Wq = Lq / self.arrival_rate  # Average time spent in queue
        W = Wq + (1 / self.service_rate)  # Average time spent in the
system
        return L, W, Lq, Wq

    def mm_c_k(self):
        # M/M/c/K Queue characteristics
        rho = self.arrival_rate / (self.service_rate * self.num_servers)
# Traffic intensity

        # Calculate the probability of having n customers in the system
        P0 = (1 / sum((self.arrival_rate / self.service_rate)**n /
self.factorial(n) for n in range(self.num_servers))) * \
             (1 + (self.arrival_rate /
self.service_rate)**self.num_servers / (self.num_servers * (1 - rho)))

        Lq = ((self.arrival_rate**2) / (self.service_rate**2)) /
(self.num_servers * (1 - rho)) * P0
        L = Lq + (self.arrival_rate / self.service_rate)  # Average
number in the system
        Wq = Lq / self.arrival_rate  # Average time spent in queue
        W = Wq + (1 / self.service_rate)  # Average time spent in the
system
        return L, W, Lq, Wq

    def factorial(self, n):
        if n == 0:
            return 1
        result = 1
        for i in range(2, n + 1):
            result *= i
```

```
        return result


# User Input
arrival_rate = float(input("Enter the arrival rate (λ): "))
service_rate = float(input("Enter the service rate (μ): "))
model_type = input("Enter the queue model type :\n1 for M/M/1, \n2 for
M/M/∞, \n3 for M/M/c, \n4 for M/M/c/K\n Enter the option:")

if model_type == '1':
    queue = QueueModel(arrival_rate, service_rate)
    L, W, Lq, Wq = queue.mm1()
    print(f"Average number of customers in the system (M/M/1): {L:.2f}")
    print(f"Average time spent in the system (M/M/1): {W:.2f} time
units")
    print(f"Average number in the queue (M/M/1): {Lq:.2f}")
    print(f"Average time spent in the queue (M/M/1): {Wq:.2f} time
units")

elif model_type == '2':
    queue = QueueModel(arrival_rate, service_rate)
    L, W, Lq, Wq = queue.mm_infinity()
    print(f"Average number of customers in the system (M/M/∞): {L:.2f}")
    print(f"Average time spent in the system (M/M/∞): {W:.2f} time units")
    print(f"Average number in the queue (M/M/∞): {Lq:.2f}")
    print(f"Average time spent in the queue (M/M/∞): {Wq:.2f} time units")

elif model_type == '3':
    num_servers = int(input("Enter the number of servers (c): "))
    queue = QueueModel(arrival_rate, service_rate, num_servers)
    L, W, Lq, Wq = queue.mm_c()
    print(f"Average number of customers in the system (M/M/c): {L:.2f}")
    print(f"Average time spent in the system (M/M/c): {W:.2f} time
units")
    print(f"Average number in the queue (M/M/c): {Lq:.2f}")
    print(f"Average time spent in the queue (M/M/c): {Wq:.2f} time
units")

elif model_type == '4':
    num_servers = int(input("Enter the number of servers (c): "))
    capacity = int(input("Enter the system capacity (K): "))
```

```
    queue = QueueModel(arrival_rate, service_rate, num_servers,
capacity)
    L, W, Lq, Wq = queue.mm_c_k()
    print(f"Average number of customers in the system (M/M/c/K):
{L:.2f}")
    print(f"Average time spent in the system (M/M/c/K): {W:.2f} time
units")
    print(f"Average number in the queue (M/M/c/K): {Lq:.2f}")
    print(f"Average time spent in the queue (M/M/c/K): {Wq:.2f} time
units")

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

OUTPUT:

```
Enter the arrival rate (λ): 4
Enter the service rate (μ): 5
Enter the queue model type :
1 for M/M/1,
2 for M/M/∞,
3 for M/M/c,
4 for M/M/c/K.
 Enter the option:1
Average number of customers in the system (M/M/1): 4.00
Average time spent in the system (M/M/1): 1.00 time units
Average number in the queue (M/M/1): 3.20
Average time spent in the queue (M/M/1): 0.80 time units
```

```
Enter the arrival rate (λ): 5
Enter the service rate (μ): 10
Enter the queue model type :
1 for M/M/1,
2 for M/M/∞,
3 for M/M/c,
4 for M/M/c/K.
 Enter the option:4
Enter the number of servers (c): 2
Enter the system capacity (K): 1
Average number of customers in the system (M/M/c/K): 0.63
Average time spent in the system (M/M/c/K): 0.13 time units
Average number in the queue (M/M/c/K): 0.13
Average time spent in the queue (M/M/c/K): 0.03 time units
```

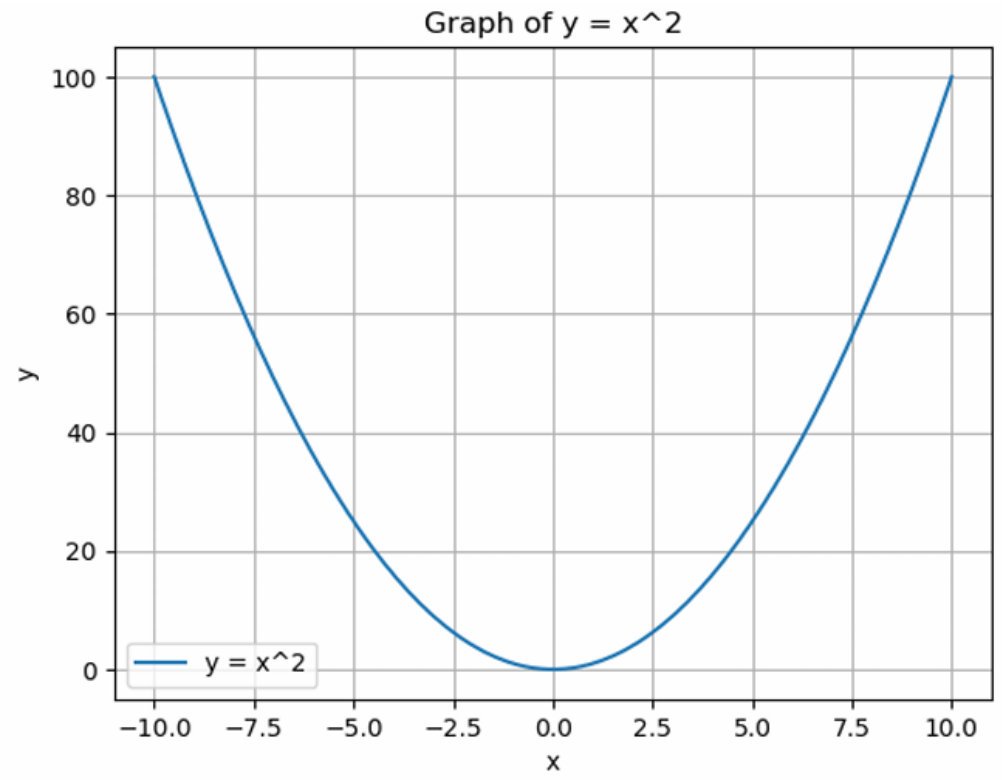
```
Enter the arrival rate (λ): 5
Enter the service rate (μ): 10
Enter the queue model type :
1 for M/M/1,
2 for M/M/∞,
3 for M/M/c,
4 for M/M/c/K.
 Enter the option:2
Average number of customers in the system (M/M/∞): 0.50
Average time spent in the system (M/M/∞): 0.10 time units
Average number in the queue (M/M/∞): 0.00
Average time spent in the queue (M/M/∞): 0.00 time units
```

11. Write a program to plot a graph for the function .

CODE

```
import matplotlib.pyplot as plt


def plot_parabola():
    x = list(range(-10, 11))
    y = [i**2 for i in x]


    plt.plot(x, y, label="y = x^2")
    plt.xlabel("x")
    plt.ylabel("y")
    plt.title("Graph of y = x^2")
    plt.legend()
    plt.grid()
    plt.show()
```

OUTPUT



12. Write a program to fit a Poisson distribution to given data.

CODE

```
from scipy.stats import poisson

# Sample data (replace this with your own data)

data = [2, 3, 4, 5, 5, 6, 7, 8, 8, 9, 10]

 # Estimate the mean (λ) for the Poisson distribution lambda_

est = np.mean(data)

# Generate Poisson probabilities

x = np.arange(0, max(data) + 1)

poisson_probs = poisson.pmf(x, lambda_est)

 # Plot histogram and Poisson fit

plt.hist(data, bins=np.arange(0, max(data) + 2) - 0.5, density=True, alpha=0.6, label='Data')

plt.plot(x, poisson_probs, 'o-', label=f'Poisson Fit (λ={lambda_est:.2f})')

# Add labels and legend

plt.xlabel("Value")

plt.ylabel("Probability")

plt.legend()
```

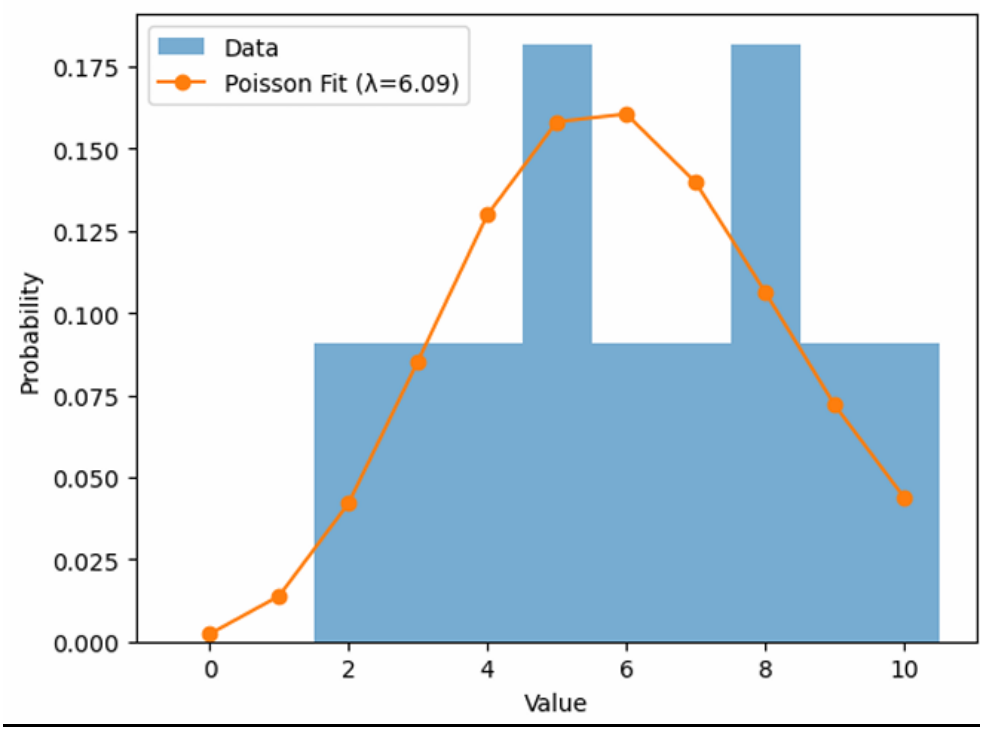
plt.show()

Example Usage

data = [2, 3, 2, 1, 0, 4, 3, 2, 1, 2]

fit_poisson(data)

OUTPUT



13. Write a Python function that calculates the Pearson correlation coefficient between two lists of numbers.

CODE

def pearson_correlation(x, y):

    if len(x) != len(y):

        raise ValueError("Input lists must have the same length.")

    n = len(x)

    mean_x = sum(x) / n

    mean_y = sum(y) / n

    numerator = sum((xi - mean_x) * (yi - mean_y) for xi, yi in zip(x, y))

```
    denominator = (
        (sum((xi - mean_x)**2 for xi in x) * sum((yi - mean_y)**2 for yi in y)) ** 0.5
    )
    if denominator == 0:
        return 0  # Avoid division by zero
    return numerator / denominator
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]
result = pearson_correlation(x, y)
print("Pearson Correlation Coefficient:", result
```

OUTPUT

```
Pearson Correlation Coefficient: 1.0
```

14. Write a Python function that calculates the Spearman correlation coefficient.

CODE

```
def spearman_correlation(x, y):
    if len(x) != len(y):
        raise ValueError("Input lists must have the same length.")

    # Rank the values in both lists
    def rank(values):
        sorted_values = sorted((val, idx) for idx, val in enumerate(values))
        ranks = [0] * len(values)
        for rank, (val, idx) in enumerate(sorted_values, start=1):
            ranks[idx] = rank
        return ranks

    rank_x = rank(x)
    rank_y = rank(y)

    # Calculate the Pearson correlation of the ranks
```

```
    n = len(x)
    mean_rank_x = sum(rank_x) / n
    mean_rank_y = sum(rank_y) / n
    numerator = sum((rx - mean_rank_x) * (ry - mean_rank_y) for rx, ry in zip(rank_x, rank_y))
    denominator = (
        (sum((rx - mean_rank_x)**2 for rx in rank_x) * sum((ry - mean_rank_y)**2 for ry in rank_y)) **
0.5
    )
    if denominator == 0:
        return 0  # Avoid division by zero
    return numerator / denominator
x = [10, 20, 30, 40, 50]
y = [1, 2, 3, 4, 5]
result = spearman_correlation(x, y)
print("Spearman Correlation Coefficient:", result)
```

OUTPUT

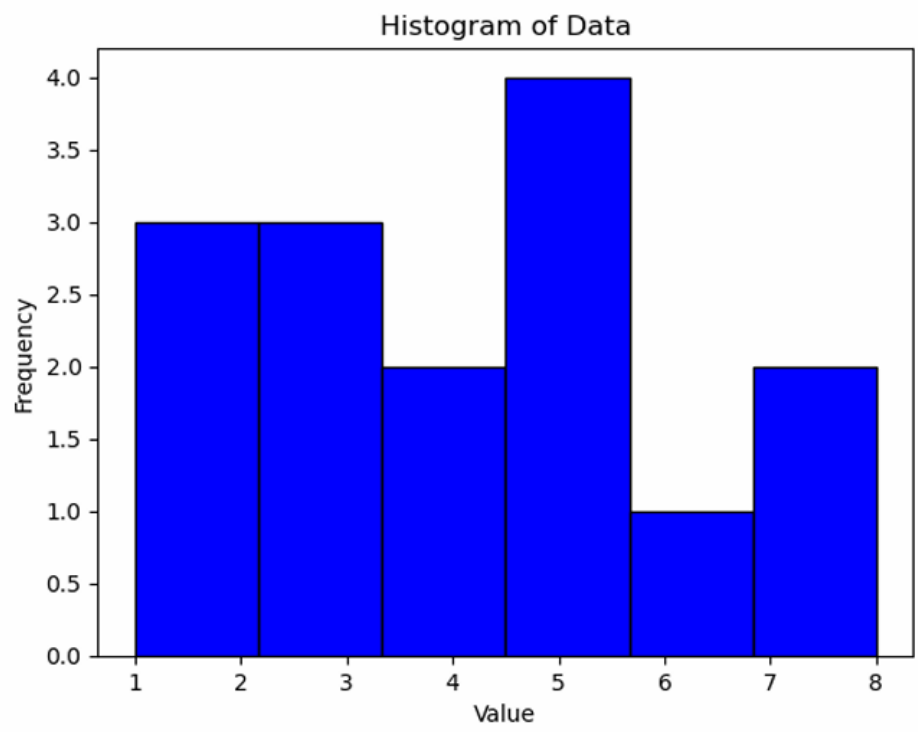
```
Spearman Correlation Coefficient: 1.0
```

15. Using Matplotlib, plot a histogram for a list of numbers.

CODE

```
import matplotlib.pyplot as plt
data = [1, 2, 2, 3, 3, 3, 4, 4, 5, 5, 5, 5, 6, 7, 8]
plt.hist(data, bins=6, color='blue', edgecolor='black')
plt.title("Histogram of Data")
plt.xlabel("Value")
plt.ylabel("Frequency")
plt.show(data = [1, 2, 2, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 5, 5]
plot_histogram(data)
```

OUTPUT



16. Write a Python function that calculates the Z-score for a list of numbers.

CODE

```
def calculate_z_scores(data):
    """
    Calculate the Z-scores for a list of numbers.

    Z-score = (x - mean) / standard deviation
    """
    if not data:
        raise ValueError("The input list cannot be empty.")

    mean = sum(data) / len(data)
    std_dev = (sum((x - mean) ** 2 for x in data) / len(data)) ** 0.5
    if std_dev == 0:
        raise ValueError("Standard deviation is zero, Z-scores cannot be calculated.")
    z_scores = [(x - mean) / std_dev for x in data]
    return z_scores
```

```
data = [10, 20, 30, 40, 50]
z_scores = calculate_z_scores(data)
print("Data:", data)
print("Z-scores:", z_scores
```

OUTPUT

```
Data: [10, 20, 30, 40, 50]
Z-scores: [-1.414213562373095, -0.7071067811865475, 0.0, 0.7071067811865475, 1.414213562373095]
```

17. Write a program to test the significance of two sample means.

CODE

```
from scipy.stats import ttest_ind
def test_significance_of_means(sample1, sample2, alpha=0.05):
    """
    Perform a two-sample t-test to test the significance of the difference between two sample means.

    Parameters:
    - sample1: List of numbers (first sample)
    - sample2: List of numbers (second sample)
    - alpha: Significance level (default is 0.05)
    Returns:
    - t_stat: T-statistic value
    - p_value: P-value of the test
    - conclusion: Whether to reject or fail to reject the null hypothesis
    """
    t_stat, p_value = ttest_ind(sample1, sample2)
    if p_value < alpha:
        conclusion = "Reject the null hypothesis: The means are significantly different."
    else:
        conclusion = "Fail to reject the null hypothesis: No significant difference between the means."
    return t_stat, p_value, conclusion
sample1 = [10, 12, 13, 15, 16]
sample2 = [14, 16, 18, 19, 20]
```

```
t_stat, p_value, conclusion = test_significance_of_means(sample1, sample2)
print("T-statistic:", t_stat)
print("P-value:", p_value)
print("Conclusion:", conclusion)
```

OUTPUT

```
T-statistic: -2.7693979882623045
P-value: 0.0243192757622274
Conclusion: Reject the null hypothesis: The means are significantly different.
```

18. Write a program to test the goodness of fit of a given dataset to a binomial distribution.

CODE

```
import numpy as np
from scipy.stats import binom, chisquare
def goodness_of_fit_binomial(data, n, p):
    """
    Perform a goodness-of-fit test to check if the data follows a binomial distribution.

    Parameters:
    - data: List of numbers representing the dataset
    - n: Number of trials (for binomial distribution)
    - p: Probability of success on a single trial (for binomial distribution)
    Returns:
    - chi2_stat: Chi-squared statistic
    - p_value: P-value of the test
    - conclusion: Whether the data fits the binomial distribution
    """
    # Get the observed frequencies for each outcome in the range [0, n]
    observed_freq, bins = np.histogram(data, bins=np.arange(-0.5, n + 1.5, 1))

    # Compute expected frequencies using the binomial distribution
    expected_freq = [binom.pmf(k, n, p) * len(data) for k in range(n + 1)]
    chi2_stat, p_value = chisquare(observed_freq, expected_freq)
```

```
    if p_value < 0.05:
        conclusion = "Reject the null hypothesis: The data does not fit the binomial distribution."
    else:
        conclusion = "Fail to reject the null hypothesis: The data fits the binomial distribution."
    return chi2_stat, p_value, conclusion
data = [2, 3, 3, 4, 5, 2, 6, 5, 3, 4, 2, 1, 3, 4]
n = 6
p = 0.5
chi2_stat, p_value, conclusion = goodness_of_fit_binomial(data, n, p)
print("Chi-squared Statistic:", chi2_stat)
print("P-value:", p_value)
print("Conclusion:", conclusion)
```

OUTPUT

```
Chi-squared Statistic: 3.5238095238095233
P-value: 0.7407992947842195
Conclusion: Fail to reject the null hypothesis: The data fits the binomial distribution.
```

19. Write a program to test the significance of two sample variances.

CODE

```
import numpy as np
from scipy.stats import f
def test_significance_of_variance(sample1, sample2, alpha=0.05):
    """
    Perform an F-test to test the significance of the difference between the variances of two samples.

    Parameters:
    - sample1: List of numbers (first sample)
    - sample2: List of numbers (second sample)
    - alpha: Significance level (default is 0.05)

    Returns:
    - f_stat: F-statistic
```

```
    - p_value: P-value of the test
    - conclusion: Whether to reject or fail to reject the null hypothesis
    """
    var1 = np.var(sample1, ddof=1)
    var2 = np.var(sample2, ddof=1)
    f_stat = var1 / var2 if var1 > var2 else var2 / var1
    df1 = len(sample1) - 1
    df2 = len(sample2) - 1
    p_value = 2 * min(f.cdf(f_stat, df1, df2), 1 - f.cdf(f_stat, df1, df2))
    if p_value < alpha:
        conclusion = "Reject the null hypothesis: The variances are significantly different."
    else:
        conclusion = "Fail to reject the null hypothesis: No significant difference between the variances."
    return f_stat, p_value, conclusion
 sample1 = [10, 12, 13, 15, 16]
sample2 = [14, 16, 18, 19, 20]
f_stat, p_value, conclusion = test_significance_of_variance(sample1, sample2)
 print("F-statistic:", f_stat)
 print("P-value:", p_value)
 print("Conclusion:", conclusion)
```

OUTPUT

```
F-statistic: 1.0175438596491229
P-value: 0.9869568504972466
Conclusion: Fail to reject the null hypothesis: No significant difference between the variances.
```

---

20. Write a program to implement linear regression in Python.

CODE

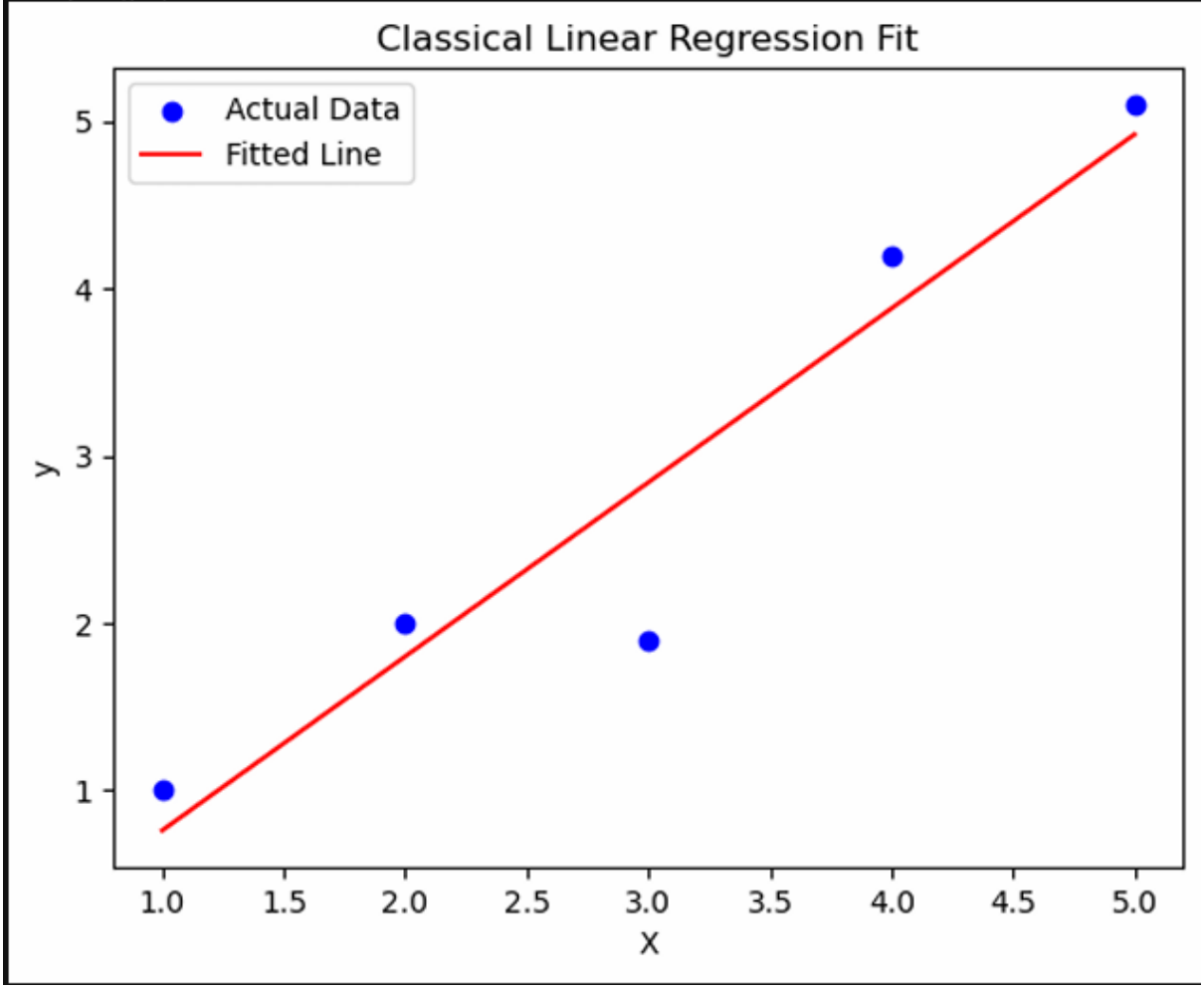
```
: import numpy as np
 import matplotlib.pyplot as plt
 class ClassicalLinearRegression:
    def __init__(self):
        self.beta_0 = 0
```

```
        self.beta_1 = 0

    def fit(self, X, y):
        """
        Fit the model using the classical Ordinary Least Squares method.

        X: Feature values (independent variable)
        y: Target values (dependent variable)
        """
        # Number of data points
        n = len(X)

        # Calculate the slope (β₁)
        numerator = n * np.sum(X * y) - np.sum(X) * np.sum(y)
        denominator = n * np.sum(X**2) - (np.sum(X))**2
        self.beta_1 = numerator / denominator

        # Calculate the intercept (β₀)
        self.beta_0 = (np.sum(y) - self.beta_1 * np.sum(X)) / n

    def predict(self, X):
        """
        Predict the target values based on the input X using the fitted model.

        X: Input feature values
        """
        return self.beta_0 + self.beta_1 * X

    def coefficients(self):
        """
        Return the coefficients (intercept and slope) of the model.
```

```
        """
        return self.beta_0, self.beta_1
# Example usage
X = np.array([1, 2, 3, 4, 5])  # Feature values (independent variable)
y = np.array([1, 2, 1.9, 4.2, 5.1])  # Target values (dependent variable)
model = ClassicalLinearRegression()
model.fit(X, y)
beta_0, beta_1 = model.coefficients()
print(f"Intercept (β₀): {beta_0}")
print(f"Slope (β₁): {beta_1}")
y_pred = model.predict(X)
plt.scatter(X, y, color='blue', label='Actual Data')
plt.plot(X, y_pred, color='red', label='Fitted Line')
plt.xlabel('X')
plt.ylabel('y')
plt.title('Classical Linear Regression Fit')
plt.legend()
plt.show()
```

OUTPUT

```
Intercept (β₀): -0.2799999999999979
Slope (β₁): 1.0399999999999994
```

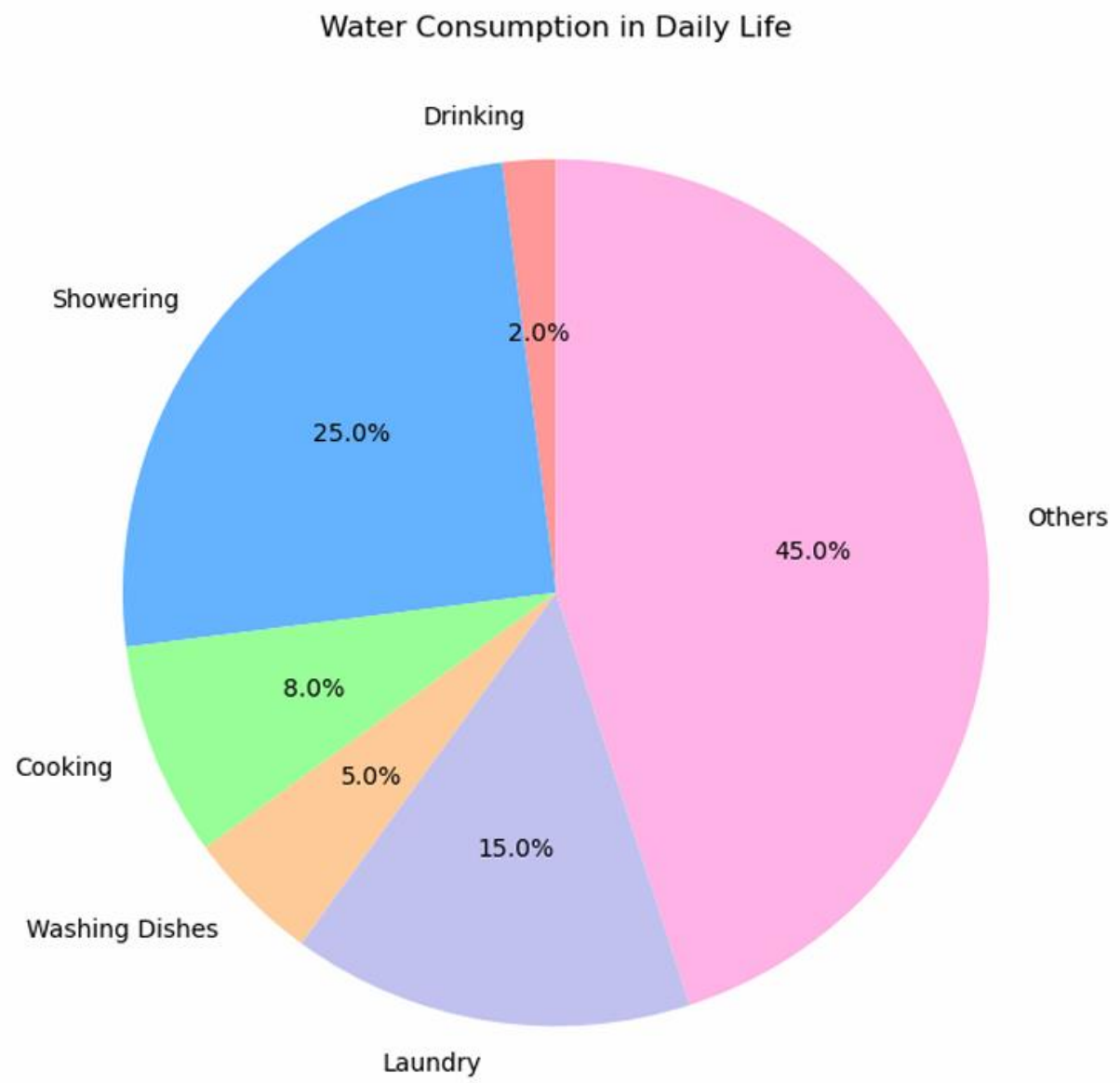


21. Write a program to plot a pie chart for the consumption of water in daily life.

CODE

```
import matplotlib.pyplot as plt

 categories = ['Drinking', 'Showering', 'Cooking', 'Washing Dishes', 'Laundry', 'Others']

 consumption = [2, 25, 8, 5, 15, 45]

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

 plt.pie(consumption, labels=categories, autopct='%1.1f%%', startangle=90,
colors=['#ff9999','#66b3ff','#99ff99','#ffcc99','#c2c2f0','#ffb3e6'])

 plt.title('Water Consumption in Daily Life')

 plt.show(
```

OUTPUT



---

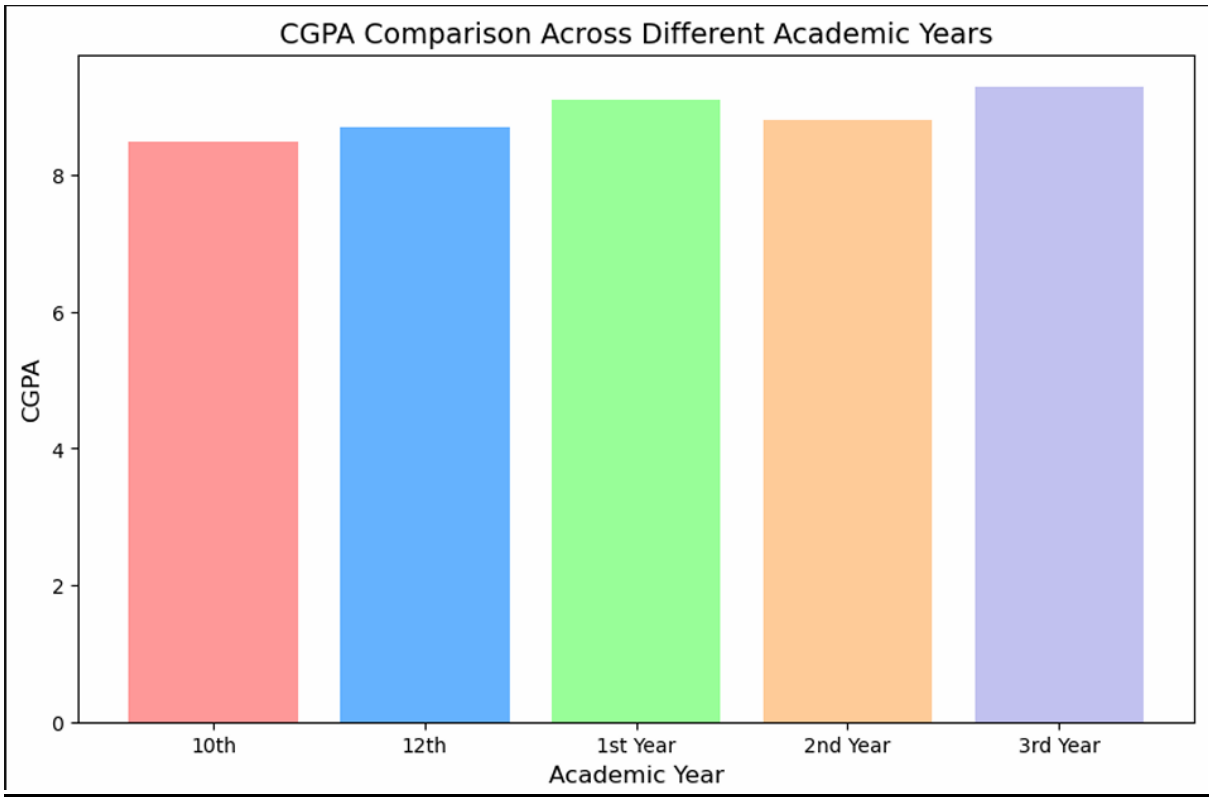
22. Write a program to plot a bar chart to display results for 10th, 12th, and years of CGPA.

CODE

```
import matplotlib.pyplot as plt
 years = ['10th', '12th', '1st Year', '2nd Year', '3rd Year']
 cgpa = [8.5, 8.7, 9.1, 8.8, 9.3]
plt.figure(figsize=(10, 6))
plt.bar(years, cgpa, color=['#ff9999', '#66b3ff', '#99ff99', '#ffcc99', '#c2c2f0'])
 plt.title('CGPA Comparison Across Different Academic Years', fontsize=14)
 plt.xlabel('Academic Year', fontsize=12)
```

```
plt.ylabel('CGPA', fontsize=12)
plt.show()
```

OUTPUT



23. Write a program to perform various statistical measures using Pandas.

CODE

```
import pandas as pd
data = {
   'Maths': [85, 90, 78, 92, 88, 76, 95, 89, 84, 91],
   'Science': [78, 85, 92, 88, 84, 79, 91, 87, 85, 93],
   'English': [80, 85, 88, 90, 85, 76, 91, 88, 79, 84]
}
df = pd.DataFrame(data)
print("Dataset:\n", df)
mean_values = df.mean()
print("\nMean Values:\n", mean_values)
median_values = df.median()
print("\nMedian Values:\n", median_values)
```

```
std_dev = df.std()
print("\nStandard Deviation:\n", std_dev)
variance = df.var()
print("\nVariance:\n", variance)
min_values = df.min()
max_values = df.max()
print("\nMinimum Values:\n", min_values)
print("\nMaximum Values:\n", max_values)
mode_values = df.mode().iloc[0]
print("\nMode Values:\n", mode_values)
correlation = df.corr()
print("\nCorrelation Matrix:\n", correlation)
covariance = df.cov()
print("\nCovariance Matrix:\n", covariance)
sum_values = df.sum()
print("\nSum of each column:\n", sum_values)
count_values = df.count()
print("\nCount of non-NA values in each column:\n", count_values
```

OUTPUT

```
Dataset:
    Maths  Science  English
0      85       78       80
1      90       85       85
2      78       92       88
3      92       88       90
4      88       84       85
5      76       79       76
6      95       91       91
7      89       87       88
8      84       85       79
9      91       93       84

Mean Values:
 Maths      86.8
Science    86.2
English    84.6
dtype: float64

Median Values:
 Maths      88.5
Science    86.0
English    85.0
dtype: float64

Standard Deviation:
 Maths      6.088240
Science    5.094660
English    4.948625
dtype: float64

Variance:
 Maths      37.066667
Science    25.955556
English    24.488889
dtype: float64
```

```
Minimum Values:
 Maths      76
Science    78
English    76
dtype: int64

Maximum Values:
 Maths      95
Science    93
English    91
dtype: int64

Mode Values:
 Maths      76.0
Science    85.0
English    85.0
Name: 0, dtype: float64

Correlation Matrix:
             Maths   Science   English
Maths    1.000000  0.420551  0.642435
Science  0.420551  1.000000  0.721891
English  0.642435  0.721891  1.000000

Covariance Matrix:
              Maths    Science    English
Maths    37.066667  13.044444  19.355556
Science  13.044444  25.955556  18.200000
English  19.355556  18.200000  24.488889

Sum of each column:
 Maths      868
Science    862
English    846
dtype: int64

Count of non-NA values in each column:
 Maths      10
Science    10
English    10
dtype: int64
```

24. Write a program to perform read and write operations with CSV files.

CODE

```
import pandas as pd

 data = {
```

```
    'Name': ['Alice', 'Bob', 'Charlie', 'David'],
    'Age': [23, 25, 30, 22],
    'City': ['New York', 'Los Angeles', 'Chicago', 'Houston']
}
df = pd.DataFrame(data)
df.to_csv('students.csv', index=False)
print("Data written to students.csv")
In [19]: df_read = pd.read_csv('students.csv')
print("\nData read from students.csv:")
print(df_read)
```

OUTPUT

```
Data read from students.csv:
      Name  Age         City
0    Alice   23     New York
1      Bob   25  Los Angeles
2  Charlie   30      Chicago
3    David   22      Houston
```

---

25. Write a program to compute values of Sin(x) using a Taylor series.

CODE

```
import math
def sin_taylor(x, terms=10):
    result = 0
    for n in range(terms):
        # Calculate the nth term of the Taylor series
        sign = (-1)**n
        term = sign * (x**(2*n + 1)) / math.factorial(2*n + 1)
        result += term
    return result
x = float(input("Enter the value of x (in radians): "))
terms = int(input("Enter the number of terms for the Taylor series: "))
approx_sin = sin_taylor(x, terms)
true_sin = math.sin(x)
```

```
print(f"Approximation of sin({x}) using Taylor series: {approx_sin}")
print(f"Actual value of sin({x}) using math.sin(x): {true_sin}")
```

OUTPUT

```
Enter the value of x (in radians): 45
Enter the number of terms for the Taylor series: 23
Approximation of sin(45.0) using Taylor series: 1.0249385564217157e+18
Actual value of sin(45.0) using math.sin(x): 0.8509035245341184
```

---

26. Write a program to display the following pattern:

5

45

345

2345

12345

CODE

```
def print_pattern():
    for i in range(1, 6):
        # Print leading spaces
        print(" " * (5 - i), end="")
        for j in range(6 - i, 6):
            print(j, end="")
        # Move to the next line after each row
        print()
print_pattern()
```

OUTPUT

```
    5
   45
  345
 2345
12345
```

---

27. Write a program to check if a number or string is a palindrome.

CODE

```
def is_palindrome(value):
    return str(value) == str(value)[::-1]
input_string = input("Enter a string: ")
if is_palindrome(input_string):
    print(f"'{input_string}' is a palindrome!")
else:
    print(f"'{input_string}' is not a palindrome.")
```

OUTPUT

```
Enter a string: 1221
'1221' is a palindrome!
```

28. Write a program to find the greatest of numbers using a loop.

CODE

```
def find_greatest(numbers):
    greatest = numbers[0]
    for num in numbers:
        if num > greatest:
            greatest = num
    return greatest
numbers = list(map(int, input("Enter numbers separated by spaces: ").split()))
greatest_number = find_greatest(numbers)
print(f"The greatest number is: {greatest_number}")
```

OUTPUT

```
Enter numbers separated by spaces: 5 65 25 34
The greatest number is: 65
```

---

29. Write a program to print the Fibonacci series.

CODE

```
def fibonacci(n):
    a, b = 0, 1
    for _ in range(n):
        print(a, end=" ")
```

```
    a, b = b, a + b
terms = int(input("Enter the number of terms for Fibonacci series: "))
fibonacci(terms)
```

OUTPUT

```
Enter the number of terms for Fibonacci series: 7
0 1 1 2 3 5 8
```

---

30. Write a program to find a factorial using recursion.

CODE

```
def factorial(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial(n - 1)
number = int(input("Enter a number to find its factorial: "))
print(f"The factorial of {number} is: {factorial(number)}")
```

OUTPUT

```
Enter a number to find its factorial: 6
The factorial of 6 is: 720
```

---

31. Write a program to check if a number is an Armstrong number.

CODE

```
def is_armstrong(number):
    num_str = str(number)
    num_digits = len(num_str)
    sum_of_powers = sum(int(digit) ** num_digits for digit in num_str)
    if sum_of_powers == number:
        return True
    else:
        return False
```

```
number = int(input("Enter a number to check if it's Armstrong: "))
if is_armstrong(number):
    print(f"{number} is an Armstrong number.")
else:
    print(f"{number} is not an Armstrong number.")
```

OUTPUT

```
Enter a number to check if it's Armstrong: 153
153 is an Armstrong number.
```

32. Write a menu-driven program to find the reverse of a number and the sum of its digits.

CODE

```
: def reverse_number(number):
    return int(str(number)[::-1])
def sum_of_digits(number):
    return sum(int(digit) for digit in str(number))
def main():
    while True:
        print("\nMenu:")
        print("1. Reverse of a number")
        print("2. Sum of digits of a number")
        print("3. Exit")

        choice = input("Enter your choice (1/2/3): ")
        if choice == '1':
            number = int(input("Enter a number to reverse: "))
            print(f"The reverse of {number} is: {reverse_number(number)}")

        elif choice == '2':
            number = int(input("Enter a number to find the sum of digits: "))
            print(f"The sum of digits of {number} is: {sum_of_digits(number)}")

        elif choice == '3':
```

```
        print("Exiting the program.")
        break

    else:
        print("Invalid choice. Please try again.")
main()
```

OUTPUT

```
Menu:
1. Reverse of a number
2. Sum of digits of a number
3. Exit
Enter your choice (1/2/3): 2
Enter a number to find the sum of digits: 57894
The sum of digits of 57894 is: 33

Menu:
1. Reverse of a number
2. Sum of digits of a number
3. Exit
Enter your choice (1/2/3): 3
Exiting the program.
```