1. Develop a Python program to manage a task list using lists and tuples, including adding, removing, updating, and sorting tasks.

tasks = []

def show\_menu():

print("\n1. Add Task\n2. Remove Task\n3. Update Task\n4. View Tasks\n5. Sort Tasks\n6. Exit")

while True:

show\_menu()

choice = int(input("Choose option: "))

if choice == 1:

task = input("Enter task: ")

tasks.append(task)

elif choice == 2:

task = input("Enter task to remove: ")

tasks.remove(task)

elif choice == 3:

i = int(input("Enter task index to update: "))

tasks[i] = input("Enter new task: ")

elif choice == 4:

print("Tasks:", tasks)

elif choice == 5:

tasks.sort()

print("Sorted Tasks:", tasks)

elif choice == 6:

break

1. Create a Python code to demonstrate the use of sets and perform set operations (union, intersection, difference) to manage student enrolments in multiple courses / appearing for multiple entrance exams like CET, JEE, NEET etc.

cet = {"Alice", "Bob", "Charlie"}

jee = {"Bob", "David"}

neet = {"Alice", "Eve"}

print("Union (All Students):", cet | jee | neet)

print("Common in CET and JEE:", cet & jee)

print("CET but not in NEET:", cet - neet)

1. Write a Python program to create, update, and manipulate a dictionary of student records, including their grades and attendance.

students = {

"John": {"grade": "A", "attendance": 95},

"Emma": {"grade": "B", "attendance": 88}

}

def show\_menu():

print("\n--- Student Record Keeper ---")

print("1. View All Students")

print("2. Add New Student")

print("3. Update Student Record")

print("4. Delete Student Record")

print("5. Exit")

while True:

show\_menu()

choice = input("Enter your choice (1-5): ")

if choice == '1':

print("\nStudent Records:")

for name, info in students.items():

print(f"{name} - Grade: {info['grade']}, Attendance: {info['attendance']}%")

elif choice == '2':

name = input("Enter student name: ")

grade = input("Enter grade: ")

attendance = int(input("Enter attendance (%): "))

students[name] = {"grade": grade, "attendance": attendance}

print(f"{name} added successfully.")

elif choice == '3':

name = input("Enter student name to update: ")

if name in students:

grade = input("Enter new grade: ")

attendance = int(input("Enter new attendance (%): "))

students[name]["grade"] = grade

students[name]["attendance"] = attendance

print(f"{name}'s record updated.")

else:

print("Student not found.")

elif choice == '4':

name = input("Enter student name to delete: ")

if name in students:

del students[name]

print(f"{name}'s record deleted.")

else:

print("Student not found.")

elif choice == '5':

print("Exiting program. Goodbye!")

break

else:

print("Invalid choice. Please enter a number from 1 to 5.")

1. Write a Python program to print a triangle and diamond pattern.

def print\_triangle(n):

print("Triangle Pattern:")

for i in range(1, n + 1):

print(' ' \* (n - i) + '\*' \* (2 \* i - 1))

def print\_diamond(n):

print("\nDiamond Pattern:")

# Top half

for i in range(1, n + 1):

print(' ' \* (n - i) + '\*' \* (2 \* i - 1))

# Bottom half

for i in range(n - 1, 0, -1):

print(' ' \* (n - i) + '\*' \* (2 \* i - 1))

# Change this value to adjust the size of the patterns

n = 5

print\_triangle(n)

print\_diamond(n)

.

1. Develop a Python program that reads a text file and prints words of specified lengths (e.g., three, four, five, etc.) found within the file.

with open("sample.txt", "r") as file:

words = file.read().split()

for word in words:

if len(word) in [3, 4, 5]:

print(word)

1. Write a Python code to take a file which contains city names on each line. Alphabetically sort the city names and write it in another file.

# File: sort\_cities.py

# Input and output file names

input\_file = "cities.txt"

output\_file = "sorted\_cities.txt"

try:

# Read city names from the input file

with open(input\_file, "r") as infile:

words = infile.read().split()

# Sort the city names alphabetically

cities.sort()

# Write sorted city names to the output file

with open(output\_file, "w") as outfile:

for city in cities:

outfile.write(city + "\n")

print(f"Cities sorted and written to '{output\_file}' successfully.")

except FileNotFoundError:

print(f"Error: The file '{input\_file}' was not found.")

except Exception as e:

print("An error occurred:", e)

1. Demonstrate the use of a Python debugger.

Muzhe nai pata

1. Design a system using classes for vehicles, rental agencies, and rental transactions. Implement methods to handle vehicle availability, rental periods, pricing, and customer bookings.

from datetime import datetime

from datetime import datetime

# Base class

class Vehicle:

def \_\_init\_\_(self, vehicle\_id, rent\_per\_day):

self.vehicle\_id = vehicle\_id

self.rent\_per\_day = rent\_per\_day

self.is\_available = True

def mark\_unavailable(self):

self.is\_available = False

def mark\_available(self):

self.is\_available = True

def \_\_str\_\_(self):

status = "Available" if self.is\_available else "Not Available"

return f"ID: {self.vehicle\_id}, Type: {self.\_\_class\_\_.\_\_name\_\_}, Rate: ₹{self.rent\_per\_day}/day, Status: {status}"

# Child classes

class Car(Vehicle):

def \_\_init\_\_(self, vehicle\_id, rent\_per\_day, seats):

super().\_\_init\_\_(vehicle\_id, rent\_per\_day)

self.seats = seats

class Bike(Vehicle):

def \_\_init\_\_(self, vehicle\_id, rent\_per\_day, has\_helmet):

super().\_\_init\_\_(vehicle\_id, rent\_per\_day)

self.has\_helmet = has\_helmet

class Scooter(Vehicle):

def \_\_init\_\_(self, vehicle\_id, rent\_per\_day, electric=False):

super().\_\_init\_\_(vehicle\_id, rent\_per\_day)

self.electric = electric

# Transaction class

class RentalTransaction:

def \_\_init\_\_(self, customer\_name, vehicle, start\_date, end\_date):

self.customer\_name = customer\_name

self.vehicle = vehicle

self.start\_date = datetime.strptime(start\_date, "%Y-%m-%d")

self.end\_date = datetime.strptime(end\_date, "%Y-%m-%d")

self.days = (self.end\_date - self.start\_date).days + 1

self.total\_price = self.days \* self.vehicle.rent\_per\_day

def display\_transaction(self):

print("\n--- Rental Transaction ---")

print(f"Customer: {self.customer\_name}")

print(f"Vehicle: {self.vehicle.\_\_class\_\_.\_\_name\_\_} (ID: {self.vehicle.vehicle\_id})")

print(f"Rental Period: {self.start\_date.date()} to {self.end\_date.date()} ({self.days} days)")

print(f"Total Cost: ₹{self.total\_price}")

# Rental Agency

class RentalAgency:

def \_\_init\_\_(self, agency\_name):

self.agency\_name = agency\_name

self.vehicles = []

self.transactions = []

def add\_vehicle(self, vehicle):

self.vehicles.append(vehicle)

def show\_available\_vehicles(self):

print(f"\n--- Available Vehicles in {self.agency\_name} ---")

for v in self.vehicles:

if v.is\_available:

print(v)

def rent\_vehicle(self, customer\_name, vehicle\_id, start\_date, end\_date):

for vehicle in self.vehicles:

if vehicle.vehicle\_id == vehicle\_id and vehicle.is\_available:

vehicle.mark\_unavailable()

transaction = RentalTransaction(customer\_name, vehicle, start\_date, end\_date)

self.transactions.append(transaction)

transaction.display\_transaction()

return

print("Vehicle not found or not available.")

def return\_vehicle(self, vehicle\_id):

for vehicle in self.vehicles:

if vehicle.vehicle\_id == vehicle\_id:

vehicle.mark\_available()

print(f"Vehicle {vehicle\_id} returned successfully.")

return

print("Vehicle not found.")

# Example usage

agency = RentalAgency("SuperDrive Rentals")

# Add vehicles

agency.add\_vehicle(Car("C101", 2000, 5))

agency.add\_vehicle(Bike("B201", 700, True))

agency.add\_vehicle(Scooter("S301", 500, electric=True))

# Show available

agency.show\_available\_vehicles()

# Rent a vehicle

agency.rent\_vehicle("Priya", "C101", "2025-04-06", "2025-04-08")

# Return it

agency.return\_vehicle("C101")#For easier code refer exp 6

1. Write a GUI program to create a student form containing Name, Age, Branch and Favourite Games. Display all the above contents in Text Box.

import tkinter as tk

# Function to run when Submit button is clicked

def submit():

# Collect selected games

games = []

if cricket.get(): games.append("Cricket")

if football.get(): games.append("Football")

if chess.get(): games.append("Chess")

if badminton.get(): games.append("Badminton")

# Print user inputs

print("Name:", name\_entry.get())

print("Branch:", branch\_choice.get())

print("Favorite Games:", ", ".join(games) if games else "None")

# Create main window

root = tk.Tk()

root.title("College Admission Form")

root.geometry("300x350")

# Name input

tk.Label(root, text="Name").pack()

name\_entry = tk.Entry(root)

name\_entry.pack()

# Branch selection (Radio Buttons)

tk.Label(root, text="Branch").pack()

branch\_choice = tk.StringVar(value="CSE") # default value

# List of branches

branches = ["CSE", "ECE", "IT", "MECH"]

for branch in branches:

tk.Radiobutton(root, text=branch, variable=branch\_choice, value=branch).pack(anchor="w")

# Favorite games (Checkboxes)

tk.Label(root, text="Favorite Games").pack()

cricket = tk.BooleanVar()

football = tk.BooleanVar()

chess = tk.BooleanVar()

badminton = tk.BooleanVar()

tk.Checkbutton(root, text="Cricket", variable=cricket).pack(anchor="w")

tk.Checkbutton(root, text="Football", variable=football).pack(anchor="w")

tk.Checkbutton(root, text="Chess", variable=chess).pack(anchor="w")

tk.Checkbutton(root, text="Badminton", variable=badminton).pack(anchor="w")

# Submit button

tk.Button(root, text="Submit", command=submit).pack(pady=10)

# Run the GUI

root.mainloop()

1. Write a Python script that prompts the user to enter a password. Use regular expressions to validate the password based on these criteria: At least 8 characters long, Contains at least one uppercase letter, one lowercase letter, one digit, and one special character.

import re

# Simple regex pattern fulfilling all the conditions

pattern = r'^(?=.\*[a-z])(?=.\*[A-Z])(?=.\*\d)(?=.\*[@#$%^&+=!]).{8,}$'

# Prompt user for password

password = input("Enter your password: ")

# Validate password

if re.match(pattern, password):

print("Password is valid.")

else:

print("Invalid password! Make sure it has at least 8 characters, including uppercase, lowercase, digit, and special character.")

1. Write a Python program to create a 1D, 2D, and 3D NumPy array. Perform basic operations like reshaping, slicing, and indexing. Calculate the dot product and cross product of two vectors.

import numpy as np

# 1D Array

array\_1d = np.array([1, 2, 3, 4, 5])

print("1D Array:", array\_1d)

# 2D Array

array\_2d = np.array([[1, 2, 3], [4, 5, 6]])

print("\n2D Array:\n", array\_2d)

# 3D Array

array\_3d = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])

print("\n3D Array:\n", array\_3d)

# Reshaping

reshaped = array\_1d.reshape((5, 1))

print("\nReshaped 1D Array to 2D (5x1):\n", reshaped)

# Slicing

slice\_2d = array\_2d[:, 1] # Second column from 2D array

print("\nSliced column from 2D array:", slice\_2d)

# Indexing

element = array\_3d[1][0][1] # Accessing element '6' from 3D array

print("\nIndexed element from 3D array:", element)

# Vectors for dot and cross product

v1 = np.array([1, 2, 3])

v2 = np.array([4, 5, 6])

# Dot Product

dot\_product = np.dot(v1, v2)

print("\nDot Product of v1 and v2:", dot\_product)

# Cross Product

cross\_product = np.cross(v1, v2)

print("Cross Product of v1 and v2:", cross\_product)

1. Develop a Python script to create two arrays of the same shape and perform element-wise addition, subtraction, multiplication, and division.

import numpy as np

# Create two arrays of the same shape

array1 = np.array([10, 20, 30, 40, 50])

array2 = np.array([2, 4, 6, 8, 10])

# Element-wise operations

addition = array1 + array2

subtraction = array1 - array2

multiplication = array1 \* array2

division = array1 / array2

# Display results

print("Array 1:", array1)

print("Array 2:", array2)

print("\nElement-wise Addition:", addition)

print("Element-wise Subtraction:", subtraction)

print("Element-wise Multiplication:", multiplication)

print("Element-wise Division:", division)

1. Using the Iris Data perform the following tasks:  
   i. Read the first 8 rows of the dataset.  
   ii. Display the column names of the Iris dataset.  
   iii. Fill any missing data with the mean value of the respective column.  
   iv. Remove rows that contain any missing values.

import pandas as pd

# Load the Iris dataset

# Option 1: Load from a local CSV file

# df = pd.read\_csv("iris.csv")

# Option 2: Load directly using seaborn (if installed)

import seaborn as sns

df = sns.load\_dataset("iris")

# i. Read the first 8 rows

print("First 8 rows of the dataset:\n", df.head(8))

# ii. Display the column names

print("\nColumn Names:", df.columns.tolist())

# iii. Fill any missing data with the mean value of the respective column

df\_filled = df.fillna(df.mean(numeric\_only=True))

print("\nData after filling missing values:\n", df\_filled.head())

# iv. Remove rows that contain any missing values

df\_dropped = df.dropna()

print("\nData after dropping rows with missing values:\n", df\_dropped.head())

1. Using the Cars Data perform the following tasks:  
   i. Create a scatter plot between the Age and Price of the cars to illustrate how the price decreases as the age of the car increases.  
   ii. Generate a histogram to show the frequency distribution of kilometres driven by the cars.  
   iii. Produce a bar plot to display the distribution of cars by fuel type.  
   iv. Create a pie chart to represent the percentage distribution of cars based on fuel types.

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Load your Cars Data (replace with your file if you're using one)

# Example: df = pd.read\_csv('cars.csv')

# For demonstration, here's a sample DataFrame:

data = {

'Age': [1, 2, 3, 4, 5, 6, 7],

'Price': [500000, 450000, 400000, 350000, 300000, 250000, 200000],

'Kilometers\_Driven': [10000, 20000, 30000, 25000, 40000, 35000, 50000],

'Fuel\_Type': ['Petrol', 'Diesel', 'Diesel', 'Petrol', 'CNG', 'Diesel', 'Petrol']

}

df = pd.DataFrame(data)

# i. Scatter plot: Age vs Price

plt.figure(figsize=(6, 4))

sns.scatterplot(x='Age', y='Price', data=df)

plt.title('Age vs Price of Cars')

plt.xlabel('Age (Years)')

plt.ylabel('Price (INR)')

plt.grid(True)

plt.tight\_layout()

plt.show()

# ii. Histogram: Kilometers Driven

plt.figure(figsize=(6, 4))

plt.hist(df['Kilometers\_Driven'], bins=5, color='skyblue', edgecolor='black')

plt.title('Distribution of Kilometers Driven')

plt.xlabel('Kilometers Driven')

plt.ylabel('Frequency')

plt.tight\_layout()

plt.show()

# iii. Bar plot: Count of cars by fuel type

plt.figure(figsize=(6, 4))

sns.countplot(x='Fuel\_Type', data=df, palette='Set2')

plt.title('Number of Cars by Fuel Type')

plt.xlabel('Fuel Type')

plt.ylabel('Count')

plt.tight\_layout()

plt.show()

# iv. Pie chart: Percentage of cars by fuel type

fuel\_counts = df['Fuel\_Type'].value\_counts()

plt.figure(figsize=(5, 5))

plt.pie(fuel\_counts, labels=fuel\_counts.index, autopct='%1.1f%%', colors=sns.color\_palette('Set2'))

plt.title('Fuel Type Distribution')

plt.tight\_layout()

plt.show()

1. Write a Python code to take a CSV file as input with coordinates of points in three dimensions. Find out the two closest points.

import pandas as pd

import numpy as np

# Function to calculate Euclidean distance

def calculate\_distance(point1, point2):

return np.linalg.norm(point1 - point2)

# Read the CSV file containing 3D coordinates

df = pd.read\_csv('points.csv')

# Initialize minimum distance and closest points

min\_distance = float('inf')

closest\_points = None

# Loop through all pairs of points

for i in range(len(df)):

for j in range(i + 1, len(df)):

point1 = df.iloc[i][['x', 'y', 'z']].values

point2 = df.iloc[j][['x', 'y', 'z']].values

distance = calculate\_distance(point1, point2)

if distance < min\_distance:

min\_distance = distance

closest\_points = (point1, point2)

# Output the closest points and their distance

print("Closest points:", closest\_points)

print("Euclidean Distance:", min\_distance)

1. Develop a Python program that simulates a banking system with a function to withdraw money. Raise custom exceptions for scenarios such as insufficient funds and invalid account numbers.

# Custom exception for insufficient funds

class InsufficientFundsError(Exception):

def \_\_init\_\_(self, message="Insufficient funds in the account"):

self.message = message

super().\_\_init\_\_(self.message)

# Custom exception for invalid account number

class InvalidAccountError(Exception):

def \_\_init\_\_(self, message="Invalid account number"):

self.message = message

super().\_\_init\_\_(self.message)

# Bank account class

class BankAccount:

def \_\_init\_\_(self, account\_number, balance=0):

self.account\_number = account\_number

self.balance = balance

def withdraw(self, amount):

if amount <= 0:

raise ValueError("Withdrawal amount must be positive.")

if amount > self.balance:

raise InsufficientFundsError(f"Cannot withdraw {amount}, insufficient balance.")

self.balance -= amount

print(f"Withdrawal successful! New balance: {self.balance}")

def deposit(self, amount):

if amount <= 0:

raise ValueError("Deposit amount must be positive.")

self.balance += amount

print(f"Deposit successful! New balance: {self.balance}")

# Function to simulate a transaction

def simulate\_transaction(account\_number, amount, action):

# Simulate a dictionary of accounts

accounts = {

1001: BankAccount(1001, 5000),

1002: BankAccount(1002, 1000)

}

# Check if account exists

if account\_number not in accounts:

raise InvalidAccountError(f"Account {account\_number} does not exist.")

account = accounts[account\_number]

if action == 'withdraw':

account.withdraw(amount)

elif action == 'deposit':

account.deposit(amount)

else:

print("Invalid action! Choose 'withdraw' or 'deposit'.")

# Main program to simulate transactions

try:

account\_number = int(input("Enter account number: "))

amount = float(input("Enter amount: "))

action = input("Enter action (withdraw/deposit): ").lower()

simulate\_transaction(account\_number, amount, action)

except InvalidAccountError as e:

print(e)

except InsufficientFundsError as e:

print(e)

except ValueError as e:

print(f"Error: {e}")

1. Develop a Python GUI application that performs various unit conversions such as currency (Rupees to Dollars), temperature (Celsius to Fahrenheit), and length (Inches to Feet).

import tkinter as tk

# Function for currency conversion

def convert\_currency():

rupees = float(rupees\_entry.get())

dollars = rupees \* 0.012 # Assuming 1 Rupee = 0.012 USD

result\_label.config(text=f"{rupees} Rupees = {dollars:.2f} Dollars")

# Function for temperature conversion

def convert\_temperature():

celsius = float(celsius\_entry.get())

fahrenheit = (celsius \* 9/5) + 32

result\_label.config(text=f"{celsius}°C = {fahrenheit:.2f}°F")

# Function for length conversion

def convert\_length():

inches = float(inches\_entry.get())

feet = inches / 12

result\_label.config(text=f"{inches} Inches = {feet:.2f} Feet")

# Create main window

root = tk.Tk()

root.title("Unit Conversion")

# Currency Conversion

tk.Label(root, text="Enter amount in Rupees:").pack()

rupees\_entry = tk.Entry(root)

rupees\_entry.pack()

tk.Button(root, text="Convert to Dollars", command=convert\_currency).pack()

# Temperature Conversion

tk.Label(root, text="Enter temperature in Celsius:").pack()

celsius\_entry = tk.Entry(root)

celsius\_entry.pack()

tk.Button(root, text="Convert to Fahrenheit", command=convert\_temperature).pack()

# Length Conversion

tk.Label(root, text="Enter length in Inches:").pack()

inches\_entry = tk.Entry(root)

inches\_entry.pack()

tk.Button(root, text="Convert to Feet", command=convert\_length).pack()

# Result display label

result\_label = tk.Label(root, text="Result will be shown here")

result\_label.pack()

root.mainloop()

1. Develop a Python GUI application that calculates the areas of different geometric figures such as circles, rectangles, and triangles. Allows users to input the necessary dimensions for various geometric figures and calculate their respective areas. The application should include input fields for the dimensions, buttons to perform the calculations, and labels to display the results.

import tkinter as tk

import math

# Function for circle area

def calculate\_circle\_area():

radius = float(radius\_entry.get())

area = math.pi \* (radius \*\* 2)

result\_label.config(text=f"Circle Area: {area:.2f} square units")

# Function for rectangle area

def calculate\_rectangle\_area():

length = float(length\_entry.get())

width = float(width\_entry.get())

area = length \* width

result\_label.config(text=f"Rectangle Area: {area:.2f} square units")

# Function for triangle area

def calculate\_triangle\_area():

base = float(base\_entry.get())

height = float(height\_entry.get())

area = 0.5 \* base \* height

result\_label.config(text=f"Triangle Area: {area:.2f} square units")

# Create main window

root = tk.Tk()

root.title("Geometric Figure Area Calculator")

# Circle area input and button

tk.Label(root, text="Enter radius for Circle:").pack()

radius\_entry = tk.Entry(root)

radius\_entry.pack()

tk.Button(root, text="Calculate Circle Area", command=calculate\_circle\_area).pack()

# Rectangle area input and button

tk.Label(root, text="Enter length and width for Rectangle:").pack()

length\_entry = tk.Entry(root)

length\_entry.pack()

width\_entry = tk.Entry(root)

width\_entry.pack()

tk.Button(root, text="Calculate Rectangle Area", command=calculate\_rectangle\_area).pack()

# Triangle area input and button

tk.Label(root, text="Enter base and height for Triangle:").pack()

base\_entry = tk.Entry(root)

base\_entry.pack()

height\_entry = tk.Entry(root)

height\_entry.pack()

tk.Button(root, text="Calculate Triangle Area", command=calculate\_triangle\_area).pack()

# Result display label

result\_label = tk.Label(root, text="Result will be shown here")

result\_label.pack()

root.mainloop()

1. Create a Python program to check whether the given input is a digit, lowercase character, uppercase character, or a special character using an 'if else-if' ladder.

# Function to check the type of input

def check\_input\_type():

# Take user input

user\_input = input("Enter a character: ")

# Check if the input is a digit

if user\_input.isdigit():

print(f"The input '{user\_input}' is a Digit.")

# Check if the input is a lowercase character

elif user\_input.islower():

print(f"The input '{user\_input}' is a Lowercase Character.")

# Check if the input is an uppercase character

elif user\_input.isupper():

print(f"The input '{user\_input}' is an Uppercase Character.")

# If it's not any of the above, consider it a special character

else:

print(f"The input '{user\_input}' is a Special Character.")

# Call the function

check\_input\_type()

1. Write a Python program to demonstrate multilevel inheritance to calculate the gross salary of an employee. The program should prompt the user for the basic salary (BS) in grandfather class and then compute the dearness allowance (DA) as 70% of BS, the travel allowance (TA) as 30% of BS, and the house rent allowance (HRA) as 10% of BS. Finally, it should calculate the gross salary as the sum of BS, DA, TA, and HRA and display the result.

# Grandfather class to input basic salary

class Grandfather:

def \_\_init\_\_(self, basic\_salary):

self.basic\_salary = basic\_salary

# Parent class to calculate allowances

class Parent(Grandfather):

def \_\_init\_\_(self, basic\_salary):

# Calling the constructor of the Grandfather class

super().\_\_init\_\_(basic\_salary)

# Calculate Dearness Allowance (DA), Travel Allowance (TA), and House Rent Allowance (HRA)

self.da = 0.70 \* self.basic\_salary # 70% of basic salary

self.ta = 0.30 \* self.basic\_salary # 30% of basic salary

self.hra = 0.10 \* self.basic\_salary # 10% of basic salary

# Child class to calculate the gross salary

class Child(Parent):

def \_\_init\_\_(self, basic\_salary):

# Calling the constructor of the Parent class

super().\_\_init\_\_(basic\_salary)

def calculate\_gross\_salary(self):

# Calculate the gross salary by summing up BS, DA, TA, and HRA

gross\_salary = self.basic\_salary + self.da + self.ta + self.hra

return gross\_salary

# Function to prompt the user for input and calculate gross salary

def calculate\_salary():

# Taking basic salary as input

basic\_salary = float(input("Enter the basic salary of the employee: "))

# Creating an object of the Child class

employee = Child(basic\_salary)

# Calculating and displaying the gross salary

gross\_salary = employee.calculate\_gross\_salary()

print(f"The gross salary of the employee is: {gross\_salary:.2f}")

# Calling the function to run the program

calculate\_salary()

1. Write a Python program to demonstrate multilevel inheritance for calculating the simple interest based on user input. The program should prompt the user to enter the principal amount, the rate of interest, and the time period in years. It should then compute the simple interest using the formula Simple Interest = (Principal × Rate × Time) / 100 and display the result.

# Grandfather class to input principal amount

class Grandfather:

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

self.principal = principal

# Parent class to input rate of interest

class Parent(Grandfather):

def \_\_init\_\_(self, principal, rate\_of\_interest):

# Calling the constructor of the Grandfather class

super().\_\_init\_\_(principal)

self.rate\_of\_interest = rate\_of\_interest

# Child class to input time period and calculate simple interest

class Child(Parent):

def \_\_init\_\_(self, principal, rate\_of\_interest, time\_period):

# Calling the constructor of the Parent class

super().\_\_init\_\_(principal, rate\_of\_interest)

self.time\_period = time\_period

# Method to calculate simple interest

def calculate\_simple\_interest(self):

simple\_interest = (self.principal \* self.rate\_of\_interest \* self.time\_period) / 100

return simple\_interest

# Function to prompt the user for input and calculate simple interest

def calculate\_interest():

# Taking user inputs for principal, rate of interest, and time period

principal = float(input("Enter the principal amount: "))

rate\_of\_interest = float(input("Enter the rate of interest (%): "))

time\_period = float(input("Enter the time period (in years): "))

# Creating an object of the Child class

interest\_calculator = Child(principal, rate\_of\_interest, time\_period)

# Calculating and displaying the simple interest

simple\_interest = interest\_calculator.calculate\_simple\_interest()

print(f"The simple interest is: {simple\_interest:.2f}")

# Calling the function to run the program

calculate\_interest()

1. Write a Python program to demonstrate List creation and 10 methods.

# Creating a list

my\_list = [1, 2, 3, 4, 5]

# 1. append() - Adds an element to the end of the list

my\_list.append(6)

print("append():", my\_list)

# 2. extend() - Adds all elements of an iterable to the end of the list

my\_list.extend([7, 8])

print("extend():", my\_list)

# 3. insert() - Adds an element at a specified position

my\_list.insert(2, 9)

print("insert():", my\_list)

# 4. remove() - Removes the first occurrence of an element

my\_list.remove(9)

print("remove():", my\_list)

# 5. pop() - Removes and returns the element at the specified position (default is last element)

my\_list.pop()

print("pop():", my\_list)

# 6. index() - Returns the index of the first occurrence of an element

print("index():", my\_list.index(3))

# 7. count() - Returns the count of occurrences of an element

print("count():", my\_list.count(4))

# 8. sort() - Sorts the list in ascending order

my\_list.sort()

print("sort():", my\_list)

# 9. reverse() - Reverses the list

my\_list.reverse()

print("reverse():", my\_list)

# 10. clear() - Removes all elements from the list

my\_list.clear()

print("clear():", my\_list)

1. Write a Python program to demonstrate Dictionary creation and 10 methods.

# Creating a dictionary

my\_dict = {"name": "John", "age": 25, "city": "New York"}

# 1. get() - Returns the value of a specified key

print("get():", my\_dict.get("age"))

# 2. keys() - Returns a view object that displays a list of all keys in the dictionary

print("keys():", my\_dict.keys())

# 3. values() - Returns a view object that displays a list of all values in the dictionary

print("values():", my\_dict.values())

# 4. items() - Returns a view object that displays a list of a dictionary's key-value tuple pairs

print("items():", my\_dict.items())

# 5. update() - Updates the dictionary with elements from another dictionary or iterable

my\_dict.update({"email": "john@example.com"})

print("update():", my\_dict)

# 6. pop() - Removes and returns the value of a specified key

print("pop():", my\_dict.pop("city"))

print("After pop:", my\_dict)

# 7. popitem() - Removes and returns the last inserted key-value pair

print("popitem():", my\_dict.popitem())

print("After popitem:", my\_dict)

# 8. clear() - Removes all items from the dictionary

my\_dict.clear()

print("clear():", my\_dict)

# 9. copy() - Returns a shallow copy of the dictionary

my\_dict = {"name": "John", "age": 25, "city": "New York"}

new\_dict = my\_dict.copy()

print("copy():", new\_dict)

# 10. setdefault() - Returns the value of a key if it exists; otherwise, sets the default value

print("setdefault():", my\_dict.setdefault("country", "USA"))

print("After setdefault:", my\_dict)

1. Write a Python program to demonstrate set creation and 10 methods.

# Creating a set

my\_set = {1, 2, 3, 4, 5}

# 1. add() - Adds an element to the set

my\_set.add(6)

print("add():", my\_set)

# 2. update() - Adds multiple elements to the set

my\_set.update([7, 8])

print("update():", my\_set)

# 3. remove() - Removes a specific element from the set

my\_set.remove(3)

print("remove():", my\_set)

# 4. discard() - Removes a specific element from the set (does not raise error if element does not exist)

my\_set.discard(9) # No error will be raised

print("discard():", my\_set)

# 5. pop() - Removes and returns an arbitrary element from the set

print("pop():", my\_set.pop())

print("After pop:", my\_set)

# 6. clear() - Removes all elements from the set

my\_set.clear()

print("clear():", my\_set)

# 7. copy() - Returns a shallow copy of the set

my\_set = {1, 2, 3, 4, 5}

new\_set = my\_set.copy()

print("copy():", new\_set)

# 8. union() - Returns a new set with all items from both sets

another\_set = {4, 5, 6, 7}

print("union():", my\_set.union(another\_set))

# 9. intersection() - Returns a set with common elements from both sets

print("intersection():", my\_set.intersection(another\_set))

# 10. difference() - Returns a set with elements in the first set but not in the second set

print("difference():", my\_set.difference(another\_set))

1. Write a Python program to plot multiple bar chart for placement data

import matplotlib.pyplot as plt

# Data

years = ['2020', '2021', '2022', '2023']

CSE = [50, 60, 65, 70]

IT = [20, 25, 30, 35]

EXTC = [15, 20, 25, 30]

AIDS = [10, 15, 20, 25]

# Stacked bar chart

plt.bar(years, CSE, label='CSE', color='skyblue')

plt.bar(years, IT, bottom=CSE, label='IT', color='salmon')

plt.bar(years, EXTC, bottom=[i+j for i,j in zip(CSE, IT)], label='EXTC', color='lightgreen')

plt.bar(years, AIDS, bottom=[i+j+k for i,j,k in zip(CSE, IT, EXTC)], label='AIDS', color='lightcoral')

# Adding labels and title

plt.xlabel('Years')

plt.ylabel('Placement Percentage')

plt.title('Placement Data - Stacked Bar Chart')

# Adding legend

plt.legend()

# Display the plot

plt.show()

1. Write a Python program to find the nCr by using function.

def factorial(n):

if n == 0 or n == 1:

return 1

return n \* factorial(n - 1)

def nCr(n, r):

if r > n:

return 0

return factorial(n) // (factorial(r) \* factorial(n - r))

def main():

try:

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

r = int(input("Enter r: "))

if n < 0 or r < 0:

print("n and r must be non-negative integers.")

else:

result = nCr(n, r)

print(f"nCr({n}, {r}) = {result}")

except ValueError:

print("Please enter valid integers.")

if \_\_name\_\_ == "\_\_main\_\_":

main()

1. Write a Python program to demonstrate the module and package.

Muzhe nai aata

1. Write a Python program to check whether the given number is an Armstrong number using function.
2. Write a Python script that prompts the user to enter mobile number and mail-id. Use regular expressions to validate them.

import re

def validate\_mobile(mobile):

# Simple regex: exactly 10 digits

pattern = r"^\d{10}$"

return bool(re.match(pattern, mobile))

def validate\_email(email):

# Simple regex: something@something.something

pattern = r"^[a-zA-Z0-9.\_]+@[a-zA-Z0-9]+\.[a-zA-Z]{2,}$"

return bool(re.match(pattern, email))

def main():

# Prompt for mobile number

mobile = input("Enter mobile number (10 digits): ")

if validate\_mobile(mobile):

print("Valid mobile number!")

else:

print("Invalid mobile number. Please enter exactly 10 digits.")

# Prompt for email ID

email = input("Enter email ID: ")

if validate\_email(email):

print("Valid email ID!")

else:

print("Invalid email ID. Use format: user@domain.com")

if \_\_name\_\_ == "\_\_main\_\_":

main()