LAB 1 AND LAB 2

# Q1} write a program to check if number is even or odd:

def check\_even\_odd(number):

if number % 2 == 0:

return "Even"

else:

return "Odd"

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

result = check\_even\_odd(num)

print(f"The number {num} is {result}.")

#Q2} write a program to assign grade to student based on present:

def assign\_grade(attendance\_percentage):

if attendance\_percentage >= 90:

return "Grade A"

elif attendance\_percentage >= 80:

return "Grade B"

elif attendance\_percentage >= 70:

return "Grade C"

else:

return "Grade D"

total\_classes = int(input("Enter total number of classes: "))

attended\_classes = int(input("Enter number of classes attended: "))

attendance\_percentage = (attended\_classes / total\_classes) \* 100

print(f"Attendance Percentage: {attendance\_percentage:.2f}%")

grade = assign\_grade(attendance\_percentage)

print(f"The assigned grade is: {grade}")

#Q3] write program to find area of circle:

import math # To access the value of pi

def calculate\_area\_of\_circle(radius):

area = math.pi \* (radius \*\* 2) # Area formula

return area

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

area = calculate\_area\_of\_circle(radius)

print(f"The area of the circle with radius {radius} is: {area:.2f}")

#Q4] write a program to check is given number is arstrong number:

def is\_armstrong\_number(number):

digits = str(number)

num\_digits = len(digits)

sum\_of\_powers = sum(int(digit) \*\* num\_digits for digit in digits)

return sum\_of\_powers == number

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

if is\_armstrong\_number(num):

print(f"{num} is an Armstrong number.")

else:

print(f"{num} is not an Armstrong number.")

#Q5] write a program to check if number is perfect number:

def is\_perfect\_number(num):

if num <= 1:

return False

divisors\_sum = 0

for i in range(1, num):

if num % i == 0:

divisors\_sum += i

return divisors\_sum == num

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

if is\_perfect\_number(num)

#Q6] write a program to reverse a number:

def reverse\_number(num):

reversed\_num = 0

while num > 0:

last\_digit = num % 10

reversed\_num = reversed\_num \* 10 + last\_digit

num = num // 10

return reversed\_num

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

reversed\_num = reverse\_number(num)

print(f"The reversed number is: {reversed\_num}")

#Q7]write a program for a console based calculator:

def add(x, y):

return x + y

def subtract(x, y):

return x - y

def multiply(x, y):

return x \* y

def divide(x, y):

if y == 0:

return "Error! Division by zero."

else:

return x / y

def calculator():

while True:

print("\nSelect operation:")

print("1. Add")

print("2. Subtract")

print("3. Multiply")

print("4. Divide")

print("5. Exit")

choice = input("Enter choice (1/2/3/4/5): ")

if choice == '5':

print("Exiting calculator.")

break

if choice not in ['1', '2', '3', '4']:

print("Invalid input. Please try again.")

continue

try:

num1 = float(input("Enter first number: "))

num2 = float(input("Enter second number: "))

except ValueError:

print("Invalid number input. Please enter numeric values.")

continue

if choice == '1':

print(f"{num1} + {num2} = {add(num1, num2)}")

elif choice == '2':

print(f"{num1} - {num2} = {subtract(num1, num2)}")

elif choice == '3':

print(f"{num1} \* {num2} = {multiply(num1, num2)}")

elif choice == '4':

print(f"{num1} / {num2} = {divide(num1, num2)}")

calculator()

#Q8] write a code to check whether number is a palindrome or not:

def is\_palindrome(num):

num\_str = str(num)

return num\_str == num\_str[::-1]

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

if is\_palindrome(num):

print(f"{num} is a palindrome.")

else:

print(f"{num} is not a palindrome.")

#Q9} write a number to check whether number is prime or not:

def is\_prime(num):

if num <= 1:

return False

for i in range(2, int(num \*\* 0.5) + 1):

if num % i == 0:

return False

return True

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

if is\_prime(num):

print(f"{num} is a prime number.")

else:

print(f"{num} is not a prime number.")

#Q10] write a program to generate fibonacci series

def fibonacci(n):

fib\_series = []

# First two terms are always 0 and 1

a, b = 0, 1

for \_ in range(n):

fib\_series.append(a)

a, b = b, a + b # Update values for the next term

return fib\_series

LAB 3

#q1] wap to check if a given number is a disaium number

def is\_disarium\_number(n):

# Convert the number to a string to easily access digits

num\_str = str(n)

length = len(num\_str)

sum\_of\_powers = sum(int(digit) \*\* (index + 1) for index, digit in enumerate(num\_str))

return sum\_of\_powers == n

num = int(input("Enter a number to check if it's a Disarium number: "))

if is\_disarium\_number(num):

print(f"{num} is a Disarium number.")

else:

print(f"{num} is not a Disarium number.")

#q2] check if a number is a harshad number

def is\_harshad\_number(n):

# Calculate the sum of digits of the number

sum\_of\_digits = sum(int(digit) for digit in str(n))

return n % sum\_of\_digits == 0

num = int(input("Enter a number to check if it's a Harshad number: "))

if is\_harshad\_number(num):

print(f"{num} is a Harshad number.")

else:

print(f"{num} is not a Harshad number.")

#q3] wap to compute value of x^n

def power(x, n):

return x \*\* n

x = float(input("Enter the base number (x): "))

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

result = power(x, n)

print(f"The value of {x}^{n} is {result}.")

#q4]# Function to compute the sum of digits of a number

def sum\_of\_digits(n):

# Initialize the sum to 0

total\_sum = 0

while n > 0:

total\_sum += n % 10 # Add the last digit to the sum

n = n // 10 # Remove the last digit by integer division

return total\_sum

num = int(input("Enter a number to compute the sum of its digits: "))

result = sum\_of\_digits(num)

print(f"The sum of digits in {num} is {result}.")

LAB 4

#q1]

#a]Write a program to print the following pattern:-

Pattern #1: Simple Number Triangle Pattern

Pattern:

1

2 2

3 3 3

4 4 4 4

5 5 5 5 5

def print\_number\_triangle(rows):

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

# Print the number 'i' repeated 'i' times

print(" ".join([str(i)] \* i))

rows = 5

print\_number\_triangle(rows)

#b]Pattern #2: Inverted Pyramid of Numbers

Pattern:

1 1 1 1 1

2 2 2 2

3 3 3

4 4

5

def print\_inverted\_pyramid(rows):

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

# Print leading spaces

print(" " \* (i - 1), end="")

print(" ".join([str(i)] \* (rows - i + 1)))

rows = 5

print\_inverted\_pyramid(rows)

c] Pattern #4: Inverted Pyramid of Descending Numbers

Pattern:

5 5 5 5 5

4 4 4 4

3 3 3

2 2

1

def print\_inverted\_pyramid\_descending(rows):

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

print(" " \* (rows - i), end="")

# Print the number 'i' repeated 'i' times

print(" ".join([str(i)] \* i))

rows = 5

print\_inverted\_pyramid\_descending(rows)

d]Pattern #5: Inverted Pyramid of the Same Digit

Pattern:

5 5 5 5 5

5 5 5 5

5 5 5

5 5

5

def print\_inverted\_pyramid\_same\_digit(rows, digit):

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

# Print leading spaces

print(" " \* (rows - i), end="")

# Print the same digit repeated 'i' times

print(" ".join([str(digit)] \* i))

rows = 5

digit = 5

print\_inverted\_pyramid\_same\_digit(rows, digit)

#Q2] write a program to:

#a] separate the following string into comma (,) separated values. X= “ India.is.my.country”

X = "India.is.my.country"

split\_values = X.split(".")

comma\_separated\_values = ",".join(split\_values)

print(comma\_separated\_values)

#b]To remove a given character from a string. Y=”M.A.N.I.P.A.L”

Y = "M.A.N.I.P.A.L"

char\_to\_remove = "."

modified\_string = Y.replace(char\_to\_remove, "")

print("Original String:", Y)

print("Modified String:", modified\_string)

#c] Write a program to sort strings alphabetically in python.

strings = ["Banana", "Apple", "Orange", "Mango", "Pineapple"]

sorted\_strings = sorted(strings)

print("Sorted list of strings:")

for string in sorted\_strings:

print(string)

#d] Take an input from a user as a string then, reverse it.

user\_input = input("Enter a string to reverse: ")

reversed\_string = user\_input[::-1]

print("Reversed string:", reversed\_string)

#e] Write a program to check if a string contains only digits.

user\_input = input("Enter a string to check if it contains only digits: ")

if user\_input.isdigit():

print("The string contains only digits.")

else:

print("The string does not contain only digits.")

#f] Write a program to find the number of vowels in the string.

def count\_vowels(string):

vowels = "aeiouAEIOU" # Vowels in both lowercase and uppercase

count = 0 # Variable to store the count of vowels

for char in string:

if char in vowels:

count += 1 # Increment the count if the character is a vowel

return count

user\_input = input("Enter a string to count the number of vowels: ")

vowel\_count = count\_vowels(user\_input)

print("Number of vowels in the string:", vowel\_count)

LAB 5

#q1]Write a program to separate the following string into comma (,) separated values.

X= “ India.is.my.country”

X = "India.is.my.country"

result = X.replace(".", ",")

print(result)

#q2] Write a program to sort strings alphabetically in python.

strings = ["banana", "apple", "cherry", "date"]

sorted\_strings = sorted(strings)

print(sorted\_strings)

#q3] Write a program to remove a given character from a string.

Y=”M.A.N.I.P.A.L”

Y = "M.A.N.I.P.A.L"

char\_to\_remove = "."

modified\_string = Y.replace(char\_to\_remove, "")

print(modified\_string)

#q4] Write a program to take an input from a user as a string then, reverse it.

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

reversed\_string = user\_input[::-1]

print("Reversed string:", reversed\_string)

#q5] Write a program to check if a string contains only digits.

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

if user\_input.isdigit():

print("The string contains only digits.")

else:

print("The string does not contain only digits.")

#q6] Write a program to find the number of vowels in the string.

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

vowels = "aeiouAEIOU"

vowel\_count = 0

for char in user\_input:

if char in vowels:

vowel\_count += 1

print(f"Number of vowels in the string: {vowel\_count}")

#q7] Write a program to check if every word in a string begins with a capital letters or not.

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

words = user\_input.split()

all\_capital = all(word[0].isupper() for word in words)

if all\_capital:

print("Every word in the string begins with a capital letter.")

else:

print("Not every word in the string begins with a capital letter.")

LAB 6

#q1] Write a program to create a class called "Person" with a name and age attribute. Create two instances of the "Person" class, set their attributes using the constructor, and print their name and age.

class Person:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

person1 = Person("Alice", 30)

person2 = Person("Bob", 25)

print(f"Person 1: Name = {person1.name}, Age = {person1.age}")

print(f"Person 2: Name = {person2.name}, Age = {person2.age}")

#q2] Create a class named 'Student' with String variable 'name' and integer variable 'roll\_no'. Assign the value of roll no as '2' and that of name as "John" by creating an object of the class Student.

class Student:

def \_\_init\_\_(self, name, roll\_no):

self.name = name

self.roll\_no = roll\_no

student1 = Student("John", 2)

print(f"Student Name: {student1.name}")

print(f"Roll Number: {student1.roll\_no}")

#q3] Write a program to define a class to represent a bank account, with methods to deposit, withdraw, and check the balance.

class BankAccount:

def \_\_init\_\_(self, initial\_balance=0):

self.balance = initial\_balance

def deposit(self, amount):

if amount > 0:

self.balance += amount

print(f"Deposited: {amount}. Current balance: {self.balance}")

else:

print("Deposit amount must be greater than zero.")

def withdraw(self, amount):

if amount > 0:

if amount <= self.balance:

self.balance -= amount

print(f"Withdrew: {amount}. Current balance: {self.balance}")

else:

print("Insufficient funds!")

else:

print("Withdrawal amount must be greater than zero.")

def check\_balance(self):

print(f"Current balance: {self.balance}")

account = BankAccount(1000)

account.check\_balance() # Check the initial balance

account.deposit(500) # Deposit $500

account.withdraw(200) # Withdraw $200

account.check\_balance() # Check the balance after transactions

account.withdraw(1500) # Try to withdraw more than the balance

#q4]⁠Write a program to define a class Student with attributes like name and age, and create objects to represent different students.

class Student:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

def display\_info(self):

print(f"Student Name: {self.name}, Age: {self.age}")

student1 = Student("Alice", 20)

student2 = Student("Bob", 22)

student3 = Student("Charlie", 19)

student1.display\_info()

student2.display\_info()

student3.display\_info()

LAB 7

#q1] Question: Create a class Person with attributes name and age, and a method display() that prints the name and age. Then, create a subclass Student that inherits from Person and adds an attribute student\_id. Write a method show\_details() in Student to print all details, including student\_id in python

class Person:

def \_\_init\_\_(self, name, age):

self.name = name # Initialize name

self.age = age # Initialize age

def display(self):

print(f"Name: {self.name}")

print(f"Age: {self.age}")

class Student(Person):

def \_\_init\_\_(self, name, age, student\_id):

super().\_\_init\_\_(name, age) # Call the parent class constructor

self.student\_id = student\_id # Initialize student\_id

def show\_details(self):

self.display() # Call the parent class display method

print(f"Student ID: {self.student\_id}")

person = Person("Alice", 30)

student = Student("Bob", 20, "S12345")

print("Person Details:")

person.display()

print("\nStudent Details:")

student.show\_details()

#q2] Question: Create a class Vehicle with a method info() that prints "This is a vehicle". Inherit Car from Vehicle and add a method car\_info() to print "This is a car". Further, create another subclass ElectricCar that inherits from Car and adds a method battery\_info() to print "This car has a battery." Demonstrate how multilevel inheritance works by calling all methods from an ElectricCar object.

in python

class Vehicle:

def info(self):

print("This is a vehicle")

class Car(Vehicle):

def car\_info(self):

print("This is a car")

class ElectricCar(Car):

def battery\_info(self):

print("This car has a battery.")

electric\_car = ElectricCar()

electric\_car.info()

electric\_car.car\_info()

electric\_car.battery\_info()

#q3] Create two classes Teacher and Author, each with their own description() method to describe the profession. Then, create a subclass TutorAuthor that inherits from both Teacher and Author. Use this subclass to call the description() method from each parent class. Use the super() function to resolve any conflicts if necessary.

class Teacher:

def description(self):

print("I am a teacher. I educate students.")

class Author:

def description(self):

print("I am an author. I write books.")

class TutorAuthor(Teacher, Author):

def description(self):

print("I am a tutor and an author.")

super().description() # Calling the method from Teacher class

super(Teacher, self).description() # Calling the method from Author class

tutor\_author = TutorAuthor()

tutor\_author.description()

#q4] Question: Create a class Animal with a method sound() that prints "Animals make sound". Then create two subclasses Dog and Cat, each with their own version of sound() method that prints "Dog barks" and "Cat meows" respectively. Demonstrate hierarchical inheritance by calling the sound() method from both Dog and Cat objects.

class Animal:

def sound(self):

print("Animals make sound")

class Dog(Animal):

def sound(self):

print("Dog barks")

class Cat(Animal):

def sound(self):

print("Cat meows")

dog = Dog()

cat = Cat()

print("Dog says:")

dog.sound() # Calls Dog's version of sound()

print("Cat says:")

cat.sound() # Calls Cat's version of sound()

LAB 8

#q1] Implement the bubble sort algo using classes and methods

class BubbleSort:

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

self.data = data # Initialize the list of data to be sorted

def sort(self):

n = len(self.data)

for i in range(n):

swapped = False

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

if self.data[j] > self.data[j + 1]:

self.data[j], self.data[j + 1] = self.data[j + 1], self.data[j]

swapped = True

if not swapped:

break

def display(self):

print("Sorted list:", self.data)

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

data = [64, 34, 25, 12, 22, 11, 90]

sorter = BubbleSort(data)

print("Original list:", sorter.data)

sorter.sort()

sorter.display()

#q2]Implement the insertion sort using Python

class InsertionSort:

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

self.data = data # Initialize the list of data to be sorted

def sort(self):

for i in range(1, len(self.data)): # Start from the second element

key = self.data[i] # The element to be inserted

j = i - 1 # Start comparing with the previous element

while j >= 0 and self.data[j] > key:

self.data[j + 1] = self.data[j]

j -= 1

self.data[j + 1] = key

def display(self):

print("Sorted list:", self.data)

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

data = [64, 34, 25, 12, 22, 11, 90]

sorter = InsertionSort(data)

print("Original list:", sorter.data)

sorter.sort()

sorter.display()

#q3]Implement the selection sort using Python

class SelectionSort:

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

self.data = data # Initialize the list of data to be sorted

def sort(self):

n = len(self.data)

for i in range(n):

min\_index = i

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

if self.data[j] < self.data[min\_index]:

min\_index = j

if min\_index != i:

self.data[i], self.data[min\_index] = self.data[min\_index], self.data[i]

def display(self):

print("Sorted list:", self.data)

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

data = [64, 34, 25, 12, 22, 11, 90]

sorter = SelectionSort(data)

print("Original list:", sorter.data)

sorter.sort()

sorter.display()

#q4] Implement the merge sort in python

class MergeSort:

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

self.data = data # Initialize the list of data to be sorted

def sort(self):

self.data = self.merge\_sort(self.data) # Call the merge\_sort function to sort the list

def merge\_sort(self, arr):

if len(arr) <= 1:

return arr # Base case: A list of 0 or 1 element is already sorted

mid = len(arr) // 2

left\_half = arr[:mid]

right\_half = arr[mid:]

left\_half = self.merge\_sort(left\_half)

right\_half = self.merge\_sort(right\_half)

return self.merge(left\_half, right\_half)

def merge(self, left, right):

sorted\_list = []

i = j = 0

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

if left[i] < right[j]:

sorted\_list.append(left[i])

i += 1

else:

sorted\_list.append(right[j])

j += 1

sorted\_list.extend(left[i:])

sorted\_list.extend(right[j:])

return sorted\_list

def display(self):

print("Sorted list:", self.data)

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

data = [64, 34, 25, 12, 22, 11, 90]

sorter = MergeSort(data)

print("Original list:", sorter.data)

sorter.sort()

sorter.display()

#q5]Implement the linear search algorithm using python

class LinearSearch:

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

self.data = data # Initialize the list of data

def search(self, target):

for index, value in enumerate(self.data):

if value == target:

return index

return -1

def display\_result(self, target):

index = self.search(target)

if index != -1:

print(f"Element {target} found at index {index}.")

else:

print(f"Element {target} not found in the list.")

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

data = [64, 34, 25, 12, 22, 11, 90]

searcher = LinearSearch(data)

target = 22

searcher.display\_result(target)

target = 100

searcher.display\_result(target)

#q5] Implement the binary search using Python

class BinarySearch:

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

self.data = data # Initialize the list of sorted data

def search(self, target):

low = 0

high = len(self.data) - 1

while low <= high:

mid = (low + high) // 2 # Find the middle index

if self.data[mid] == target: # Target found

return mid

elif self.data[mid] < target: # Target is in the right half

low = mid + 1

else: # Target is in the left half

high = mid - 1

return -1 # Return -1 if the target is not found

def display\_result(self, target):

index = self.search(target)

if index != -1:

print(f"Element {target} found at index {index}.")

else:

print(f"Element {target} not found in the list.")

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

# Create an object of BinarySearch with a sorted list

data = [11, 12, 22, 25, 34, 64, 90]

searcher = BinarySearch(data)