
Practical 1:

AIM: If the age of Ram, Sam, and Khan are input through the keyboard, write a python program to determine the eldest and youngest of the three.

```
# Taking age input from the user ram
= int(input("Enter Ram's age: ")) sam
= int(input("Enter Sam's age: ")) khan
= int(input("Enter Khan's age: "))
# Finding the eldest if ram > sam
and ram > khan: print("Ram is
the eldest.") elif sam > ram and
sam > khan:
    print("Sam is the eldest.")
else:
    print("Khan is the eldest.")
# Finding the youngest if ram < sam
and ram < khan: print("Ram is the
youngest.") elif sam < ram and sam
< khan:
    print("Sam is the youngest.")
else:
    print("Khan is the youngest.")
```

Enter Ram's age: 20

Enter Sam's age: 25

Enter Khan's age: 18 Sam

is the eldest.

Khan is the youngest.

Practical 2

AIM: Write a Python program to print first 10 prime number.

```
# Get how many prime numbers to print from user N =
int(input("Enter how many prime numbers to print: "))
count = 0 # Count of prime numbers found num = 2 #
Number to check if prime print(f"First {N} prime numbers
are:") while count < N:
    is_prime = True # Assume num is prime for
    i in range(2, num):
        if num % i == 0: is_prime =
        False break if is_prime:
        print(num, end=" ") count
        += 1 num += 1
```

Enter how many prime numbers to print: 10

First 10 prime numbers are:

2 3 5 7 11 13 17 19 23 29

Practical 3

AIM: Write a Python program to count the number of vowels (a, e, i, o, u) present in a given string.

```
# Program to count the number of vowels in a string
# Get input from user text =
input("Enter a string: ")
# Define vowels vowels
= "aeiouAEIOU" #
Initialize counter count
= 0
# Loop through each character in the string for
char in text:
    if char in vowels:
        count += 1
# Display result print("Number of
vowels:", count) Enter a string:
hello WORLD

Number of vowels: 3
```

Practical 4

AIM: Write a Python program that includes two functions to calculate the factorial of a number: one using an iterative approach (with loops) and another using recursion.

```
# Program to calculate factorial using both iterative and recursive approaches

# Iterative approach def
factorial_iterative(n):
    result = 1
    for i in
range(1, n + 1):
        result *= i
    return result # Recursive
approach def
factorial_recursive(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial_recursive(n - 1)

# Get user input num =
int(input("Enter a number: "))

# Display results print("Factorial using iterative approach:",
factorial_iterative(num)) print("Factorial using recursive approach:",
factorial_recursive(num))
```

Output:

Enter a number: 4

Factorial using iterative approach: 24

Factorial using recursive approach: 24

Practical 5

AIM: Write a Python function to check if a given number reads the same forwards and backwards (i.e., it is a palindrome number).

```
# Function to check if a number is a palindrome def
is_palindrome(number):

    # Convert number to string for easy comparison
    str_num = str(number)

    # Check if the string reads the same forwards and backwards
    if str_num == str_num[::-1]:

        return True

    else:

        return False

# Get input from user num =
int(input("Enter a number: "))

# Check and display result if
is_palindrome(num):

    print(num, "is a palindrome number.")

else:

    print(num, "is not a palindrome number.")
```

Enter a number: 535

535 is a palindrome number.

Practical 6

AIM: Write a Python function to check if a given number is an Armstrong number or not.

```
# Function to check if a number is an Armstrong number
def is_armstrong(number):
    # Convert the number to string to find its digits
    digits = str(number)
    power = len(digits) # Number of digits
    # Calculate the sum of each digit raised to the power of number of digits
    total = sum(int(digit) ** power for digit in digits) # Check if
    the total equals the original number          return total == number
    # Get input from user num =
    int(input("Enter a number: "))
    # Check and display result if
    is_armstrong(num):
        print(num, "is an Armstrong number.")
    else:
        print(num, "is not an Armstrong number.")
```

Enter a number: 274

274 is not an Armstrong number.

Enter a number: 153

153 is an Armstrong number.

Practical 7

AIM: Write a Python program to read data from a file and write data to a file using file handling techniques.

```
# Program to read from and write to a file in Python

# Write data to a file with
open("example.txt", "w") as file:

    file.write("Hello, this is a test file.\n")

    file.write("Python file handling is simple and powerful!\n")

file.write("This line is written using the write() method.\n") print("Data has been
written to 'example.txt' successfully.\n")

# Read data from the file
with open("example.txt", "r") as file:
    content = file.read() print("Reading data
from 'example.txt':")
print("-----")
print(content)

Data has been written to 'example.txt' successfully.
Reading data from 'example.txt':
-----

Hello, this is a test file.

Python file handling is simple and powerful!

This line is written using the write() method.
```

Practical 8

AIM: Write a Python program to handle division by zero errors using a try- except block.

```
# Program to handle division by zero using try-except
try:
    # Take input from user
    numerator = int(input("Enter the numerator: "))
    denominator = int(input("Enter the denominator: "))

    # Try performing division
    result = numerator / denominator

    print("Result:", result)
except ZeroDivisionError:
    # Handles the case where denominator is zero
    print("Error: Division by zero is not allowed.")
except ValueError:
    # Handles invalid input (like letters instead of numbers)
    print("Error: Please enter valid integers.")
```

Output: →**Valid Input**

Enter the numerator: 6

Enter the denominator: 3

Result: 2.0

→ Division by Zero

Enter the numerator: 5

Enter the denominator: 0

Error: Division by zero is not allowed.

Practical 9

AIM: Write a Python program that defines a class Person with attributes and methods, and then create a subclass Student that inherits from Person.

```
# Base class class
Person:
def __init__(self, name, age):
    self.name = name
    self.age = age
def display_info(self):
    print(f"Name:",self.name)
    print(f"Age:",self.age) #
Derived (child) class class
Student(Person):
    def __init__(self, name, age, student_id, course):
        # Call the parent class constructor using super()
        super().__init__(name, age)
```

```
self.student_id = student_id

self.course = course

    def display_student_info(self):
        # Display both Person and Student information
        self.display_info()

print(f"Student ID:",self.student_id)

print(f"Course:",self.course)

# Create an object of the Student class s1 =
Student("Dhruvi", 20, "S123", "Computer Science")

# Display student details print("Student
Details:") s1.display_student_info()
```

Output:

```
Student Details:
Name: Dhruvi
Age: 20
Student ID: S123
Course: Computer Science
```

Practical 10:

AIM:

- 1. Linear Search:** Write a Python program to perform a linear search to find an element in a list.
- 2. Bubble Sort:** Write a Python program to sort a list using the bubble sort algorithm.

1. Linear Search:

```
# Linear Search Program in Python def
linear_search(list, target):
    for i in range(len(list)):
        if list[i] == target:
            return i # Return index if element found
    return -1 # Return -1 if element not found
# Example usage numbers = [10, 25, 30, 45, 50]
search_item = int(input("Enter the element to search: "))
result = linear_search(numbers, search_item)
if result != -1:
    print(f"Element found at index :",result) else:
    print("Element not found in the list")
```

Output:

```
Enter the element to search: 45
Element found at index : 3
```

```
Enter the element to search: 3
Element not found in the list
```

2. Bubble Sort:

```
# Bubble Sort Program in Python
def bubble_sort(list):
    n = len(list)
    for i in range(n - 1):
        for j in range(0, n - i - 1):
            if list[j] > list[j + 1]:
                # Swap elements
                list[j], list[j + 1] = list[j + 1], list[j]
    return list
# Example usage
numbers = [64, 34, 25, 12, 22, 11, 90]
print("Original list:", numbers)

sorted_list = bubble_sort(numbers)
print("Sorted list:", sorted_list)
```

Output:

Original list: [64, 34, 25, 12, 22, 11, 90]

Sorted list: [11, 12, 22, 25, 34, 64, 90]

1:

Write a Python function to sum all the numbers in a list.

```
# Function to sum all numbers in a list
def sum_list(numbers):
    total = 0
    for num in numbers:
        total += num
    return total

# Example usage
my_list = [10, 20, 30, 40, 50]
print("List:", my_list)
print("Sum of all numbers:", sum_list(my_list))
```

Output:

List: [10, 20, 30, 40, 50]

Sum of all numbers: 150

2:

Write a Python program to Swapping of two numbers with and without using temporary variable.

1: Swapping Two Numbers Using a Temporary Variable

```
# Initial values a = 5 b = 10

print("Before swapping with temp: a =", a, ", b =", b)

temp = a # Store a's value temporarily a = b # Assign
b to a b = temp # Assign temp (original a) to b

print("After swapping with temp: a =", a, ", b =", b)
```

Output:

Before swapping with temp: a = 5 , b = 10

After swapping with temp: a = 10 , b = 5

2: Swapping Two Numbers Without Using a Temporary Variable

```
a = 5

b = 10

print("Before swapping without temp: a =", a, ", b =", b)

a = a + b # Add both numbers, store in a b = a - b #
Subtract b from new a, gives original a a = a - b #
Subtract new b from new a, gives original b print("After
swapping without temp: a =", a, ", b =", b)
```

Output:

Before swapping without temp: a = 5 , b = 10

After swapping without temp: a = 10 , b = 5

3:

Write a program to get the file size of a plain text file.

```
import os
# Specify the file path (replace with your file name/path)
file_path = "example.txt" # Check if the file exists if
os.path.isfile(file_path): # Get file size in bytes size =
os.path.getsize(file_path) print(f"The size of '{file_path}'
is {size} bytes.") else:
    print(f"The file '{file_path}' does not exist.")
```

Output:

The size of 'example.txt' is 2048 bytes.

Practical 14:

AIM: Write a program to take an input N and print Fibonacci sequence till N terms.

```
# Take input from user
N = int(input("Enter the number of terms for Fibonacci sequence: "))
# Initialize first two terms a,
b = 0, 1
# Check if N is valid if N <= 0: print("Please
enter a positive integer.")
elif N == 1:
    print("Fibonacci sequence up to 1 term:")
    print(a)
else:
    print(f"Fibonacci sequence up to {N} terms:")
    count = 0 while count < N:
        print(a, end=" ") #
        Calculate the next term
        a, b = b, a + b count += 1
```

Output:

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

Practical 15:

AIM: Write a program that creates a list of 10 random integers. Then create two lists by name `odd_list` and `even_list` that have all odd and even values of the list respectively.

```
numbers = [] # Create an empty list to store user inputs

# Take 10 integer inputs from the user and append them to the list

print("Enter 10 integers:") for i in range(10):

    num = int(input(f"Number {i+1}: ")) # Taking input one by one

    numbers.append(num) # Adding the input number to the list 'numbers'

# Create two empty lists to store odd and even numbers separately

odd_list = [] even_list = []

# Loop through the 'numbers' list to check each number

for num in numbers:

    if num % 2 == 0: # Check if the number is divisible by 2 (even)

        even_list.append(num) # Add the number to the 'even_list'

    else:

        odd_list.append(num) # Otherwise, add it to the 'odd_list'

# Print the original list of numbers entered by the user

print("Original list:", numbers) print("Odd numbers:", odd_list)    #

Print the list of odd numbers print("Even numbers:", even_list)    #

Print the list of even numbers
```

Output:

Enter 10 integers:

Number 1: 12

Number 2: 7

Number 3: 19

Number 4: 8

Number 5: 5

Number 6: 16

Number 7: 21

Number 8: 4

Number 9: 11

Number 10: 10

Original list: [12, 7, 19, 8, 5, 16, 21, 4, 11, 10]

Odd numbers: [7, 19, 5, 21, 11]

Even numbers: [12, 8, 16, 4, 10]

Practical 16:

AIM: Write a python program to find sum of natural numbers.

```
# Get input from the user: how many natural numbers to sum
n = int(input("Enter a positive integer: ")) sum = 0 # Initialize
sum variable
# Loop from 1 to n (inclusive) and add each number to sum for
i in range(1, n + 1):
    sum += i print("Sum of first", n, "natural
numbers is:", sum)
```

Output:

```
Enter a positive integer: 8
Sum of first 8 natural numbers is: 36
```

Practical 17:

AIM: Write a Python program to print a pattern.

```
  *
 * * *
* * * * *
* * * * * *
* * * * * * *
* * * * * * * *
```

1.

```
# Star Pyramid Pattern
```

```

rows = 5
for i in range(rows):
    # Print spaces
    for j in range(rows - i - 1):
        print(" ", end="")

    # Print stars
    for k in range(2 * i + 1):
        print("*",
end="")    print() # Move to next line

```

Output:

```

      *
     ***
    *****
   *********
  ***********

```

2.

```

  1
 1 2 3
1 2 3 4 5
1 2 3 4 5 6 7
1 2 3 4 5 6 7 8 9

```

```

# Number Pyramid Pattern
rows = 5
for i in range(1,
rows + 1):
    # Print spaces

    for j in range(rows - i):
        print(" ", end="")

    # Print numbers

```

```
for k in range(1, 2 * i):  
    print(k, end="")  
  
print() # Move to next line
```

Output:

```
1  
123  
12345  
1234567  
123456789
```