**LAB CYCLE 1**

**Experiment No : 1**

**Date :** 30/09/2024

**Aim :** Write a program that prompts the user to enter his first name and last name and then displays

a message “Greetings!!! First name Last name”.

**Pseudocode :**

1. Read the first name

2. Read the last name

3. Print("greetings!!!", fname, lname)

**Method :**

|  |  |  |
| --- | --- | --- |
| **Function** | **Description** | **Syntax** |
| input() | Allows user input (Returns a string value) | input(prompt) |
| print() | Prints the specified message to the screen | print(object(s)) |

**Source Code :**

a=input("Enter your first name:")

b=input("Enter your last name:")

print("greetings!!!",a,b)

**Output :**

Enter your first name: John

Enter your last name: Doe

Greetings!!! John Doe

**Result :** The program is successfully executed and the output is verified.

**Experiment No : 2**

**Date :** 30/09/2024

**Aim :** Write a program to demonstrate different number data types in python.

**Pseudocode :**

1. Set x = 2, y = 3.3, z = 3 + 2j

2. Print "Type of x =", type of x

3. Print "Type of y =", type of y

4. Print "Type of z =", type of z

5. Print "integer:", x, "float:", y, "complex number:", z

**Method :**

|  |  |  |
| --- | --- | --- |
| **Function** | **Description** | **Syntax** |
| type() | Returns the type of the  specified object | type(object) |

**Source Code :**

x,y,z=2,3.3,3+2j

print("Type of x=",type(x))

print("Type of y=",type(y))

print("Type of z=",type(z))

print("integer:",x)

print("float:",y)

print("complex number:",z)

**Output :**

Type of x = <class 'int'>

Type of y = <class 'float'>

Type of z = <class 'complex'>

integer: 2

float: 3.3

complex number: (3+2j)

**Result :** The program is successfully executed and the output is verified.

**Experiment No : 3**

**Date :** 30/09/2024

**Aim :** Write a program to calculate the area of a circle by reading inputs from the user.

**Pseudocode :**

1. Read r

2. Set area = 3.14 \* r \* r

3. Print "Area of the circle =", area

**Source Code :**

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

area = 3.14 \* r \* r

print("Area of the circle =", area)

**Output :**

Enter the radius of the circle: 5

Area of the circle = 78.5

**Result :** The program is successfully executed and the output is verified.

**Experiment No : 4**

**Date :** 30/09/2024

**Aim :** Write a program to calculate the salary of an employee given his basic pay (to be entered by

the user) . HRA = 10 percent of the basic pay, TA = 5 percent of the basic pay.

**Pseudocode :**

1. Read BP

2. Set HRA = 0.10 \* BP

3. Set TA = 0.5 \* BP

4. Set salary = BP + HRA + TA

5. Print "Salary of the employee is:", salary

**Source Code :**

BP = float(input("Enter the basic pay of the employee: "))

HRA = 0.10 \* BP

TA = 0.5 \* BP

salary = BP + HRA + TA

print("Salary of the employee is:", salary)

**Output :**

Enter the basic pay of the employee: 25000

Salary of the employee is: 37500.0

**Result :** The program is successfully executed and the output is verified.

**Experiment No : 5**

**Date :** 30/09/2024

**Aim :** Write a Python program to perform arithmetic operations on two integer numbers.

**Pseudocode :**

1. Read x

2. Read y

3. Print "Sum:", x, "+", y, "=", x + y

4. Print "Difference:", x, "-", y, "=", x - y

5. Print "Multiplication:", x, "\*", y, "=", x \* y

6. Print "Division:", x, "/", y, "=", x / y

7. Print "Remainder:", x, "%", y, "=", x % y

**Source Code :**

x = int(input("Enter the first number: "))

y = int(input("Enter the second number: "))

print("Sum:", x, "+", y, "=", x + y)

print("Difference:", x, "-", y, "=", x - y)

print("Multiplication:", x, "\*", y, "=", x \* y)

print("Division:", x, "/", y, "=", x / y)

print("Remainder:", x, "%", y, "=", x % y)

**Output :**

Enter the first number: 10

Enter the second number: 5

Sum: 10 + 5 = 15

Difference: 10 - 5 = 5

Multiplication: 10 \* 5 = 50

Division: 10 / 5 = 2.0

Remainder: 10 % 5 = 0

**Result :** The program is successfully executed and the output is verified.

**Experiment No : 6**

**Date :** 30/09/2024

**Aim :** Write a Python program to get a string which is n (non-negative integer) copies of a given

string.

**Pseudocode :**

1. Read str

2. Read n

3. Print str \* n

**Source Code :**

str = input("Enter a string: ")

n = int(input("Enter the number of repetitions: "))

print(str \* n)

**Output :**

Enter a string: Hello

Enter the number of repetitions: 3

HelloHelloHello

**Result :** The program is successfully executed and the output is verified.

**Experiment No : 7**

**Date :** 30/09/2024

**Aim :** Program to accept an integer n and compute n+nn+nnn.

[Hint : n = 5, then compute 5 + 55 + 555]

**Pseudocode :**

1. Read n

2. Print n, n\*2, n\*3

3. Print "Sum:", n + n\*2 + n\*3

**Source Code :**

n = input("Enter the value of n: ")

print(n, "+", n\*2, "+", n\*3)

sum = int(n) + int(n\*2) + int(n\*3)

print("Sum:", sum)

**Output :**

Enter the value of n: 5

5 + 55 + 555

Sum: 615

**Result :** The program is successfully executed and the output is verified.

**Experiment No : 8**

**Date :** 30/09/2024

**Aim :** Find biggest of 3 numbers entered.

**Pseudocode :**

1. Read the 1st number (a)

2. Read the 2nd number (b)

3. Read the 3rd number (c)

4. Compare the numbers:

a. If a > b and a > c, largest = a

b. Else if b > a and b > c, largest = b

c. Else, largest = c

5. Print the largest number

**Source Code :**

a = int(input("Enter the 1st number: "))

b = int(input("Enter the 2nd number: "))

c = int(input("Enter the 3rd number: "))

if a > b and a > c:

largest = a

elif b > a and b > c:

largest = b

else:

largest = c

print("The largest number is:", largest)

**Output :**

Enter the 1st number: 12

Enter the 2nd number: 25

Enter the 3rd number: 8

The largest number is: 25

**Result :** The program is successfully executed and the output is verified.

**Experiment No : 9**

**Date :** 30/09/2024

**Aim** : Program to determine whether a year is a leap year or not.

**Pseudocode :**

1. Read year

2. If year is divisible by 4:

a. If year is not divisible by 100 or year is divisible by 400:

Print "year is a leap year"

3. Else:

a. Print "year is not a leap year"

**Source Code :**

year = int(input("Enter a year: "))

if year % 4 == 0 and (year % 100 != 0 or year % 400 == 0):

print(year, "is a leap year")

else:

print(year, "is not a leap year")

**Output :**

Enter a year: 2024

2024 is a leap year

**Result :** The program is successfully executed and the output is verified.

**Experiment No : 10**

**Date :** 30/09/2024

**Aim :** Write a Python program to determine the rate of entry-ticket in a trade fair

based on age as follows :

|  |  |
| --- | --- |
| **Age** | **Rate** |
| <10 | 7 |
| >10 and <60 | 10 |
| >=60 | 5 |

**Pseudocode :**

1. Read age

2. If age is 60 or greater:

a. Print "Ticket rate = 5"

3. Else if age is less than 60 but 10 or greater:

a. Print "Ticket rate = 10"

4. Else:

a. Print "Ticket rate = 7

**Source Code :**

age = int(input("Enter the age: "))

if age >= 60:

print("Ticket rate = 5")

elif age < 60 and age >= 10:

print("Ticket rate = 10")

else:

print("Ticket rate = 7")

**Output :**

Enter the age: 65

Ticket rate = 5

Enter the age: 25

Ticket rate = 10

Enter the age: 5

Ticket rate = 7

**Result :** The program is successfully executed and the output is verified.

**Experiment No : 11**

**Date** : 30/09/2024

**Aim :** Write a Python program to solve a quadratic equation.

**Pseudocode :**

1. Read the coefficients a, b, and c

2. Calculate the discriminant (d) as d = b^2 - 4ac

3. If d < 0:

a. Calculate two complex solutions using the formula:

sol1 = (-b - sqrt(d)) / (2a)

sol2 = (-b + sqrt(d)) / (2a)

b. Print the complex solutions

4. If d == 0:

a. Calculate the single solution as x = -b / (2a)

b. Print the single solution

5. If d > 0:

a. Calculate two real solutions using the formula:

sol1 = (-b - sqrt(d)) / (2a)

sol2 = (-b + sqrt(d)) / (2a)

b. Print the real solutions

**Method :**

|  |  |  |
| --- | --- | --- |
| **Function** | **Description** | **Syntax** |
| cmath.sqrt() | Computes the square root for real numbers | cmath.sqrt(value) |
| math.sqrt() | Displays the output to the user | math.sqrt(value) |

**Source Code :**

import math, cmath

print("Quadratic equation solver: ax^2 + bx + c = 0")

a = float(input("Enter coefficient of x^2 (a): "))

b = float(input("Enter coefficient of x (b): "))

c = float(input("Enter constant value (c): "))

d = (b\*\*2) - (4\*a\*c)

if d < 0:

sol1 = (-b - cmath.sqrt(d)) / (2 \* a)

sol2 = (-b + cmath.sqrt(d)) / (2 \* a)

print(f"The solution has complex solutions: {sol1}, {sol2}")

elif d == 0:

x = -b / (2 \* a)

print(f"Equation has one solution: {x}")

else:

sol1 = (-b - math.sqrt(d)) / (2 \* a)

sol2 = (-b + math.sqrt(d)) / (2 \* a)

print(f"Equation has two solutions: {sol1}, {sol2}")

**Output :**

Enter coefficient of x^2 (a): 1

Enter coefficient of x (b): -3

Enter constant value (c): 2

Equation has two solutions: 1.0, 2.0

**Result :** The program is successfully executed and the output is verified.

**LAB CYCLE 2**

**Experiment No : 1**

**Date** : 21/10/2024

**Aim :** Create a string from the given string where the first and last character are exchanged.

**Pseudocode :**

1. Read string

2. Set newstring = string[-1] + string[1:-1] + string[0]

3. Print newstring

**Source Code :**

string = input("Enter a string: ")

newstring = string[-1] + string[1:-1] + string[0]

print(newstring)

**Output :**

Enter a string: hello

oellh

**Result :** The program is successfully executed and the output is verified.

**Experiment No : 2**

**Date :** 21/10/2024

**Aim :** Get a string from an input string where all occurrences of the first character are replaced with

‘$’, except the first character**.**

**Pseudocode :**

1. Read string

2. Set first\_char = string[0]

3. Set new\_string = first\_char + string[1:].replace(first\_char, "$")

4. Print "Modified string:", new\_string

**Method :**

|  |  |  |
| --- | --- | --- |
| **Function** | **Description** | **Syntax** |
| replace() | Replaces all occurrences of a character with another | string.replace(old, new) |

**Source Code :**

string = input("Enter a string: ")

first\_char = string[0]

new\_string = first\_char + string[1:].replace(first\_char, "$")

print("Modified string:", new\_string)

**Output :**

Enter a string: restart

Modified string: resta$t

**Result :** The program is successfully executed and the output is verified.

**Experiment No : 3**

**Date :** 21/10/2024

**Aim :** Create a single string separated with space from two strings by swapping the character at

position 1.

**Pseudocode :**

1. Read string1

2. Read string2

3. Set swap\_str1 = string1[0] + string2[1] + string1[2:]

4. Set swap\_str2 = string2[0] + string1[1] + string2[2:]

5. Set string3 = swap\_str1 + " " + swap\_str2

6. Print string3

**Source Code :**

string1 = input("Enter your string 1: ")

string2 = input("Enter your string 2: ")

swap\_str1 = string1[0] + string2[1] + string1[2:]

swap\_str2 = string2[0] + string1[1] + string2[2:]

string3 = swap\_str1 + " " + swap\_str2

print(string3)

**Output :**

Enter your string 1: hello

Enter your string 2: world

hallo werld

**Result :** The program is successfully executed and the output is verified.

**Experiment No : 4**

**Date :** 21/10/2024

**Aim :** Count the number of characters (character frequency) in a string.

**Pseudocode :**

1. Read n and convert it to lowercase

2. Initialize an empty dictionary s

3. For each character i in n:

a. If i is in s, increment s[i] by 1

b. Else, set s[i] = 1

6. Print s

**Method :**

|  |  |  |
| --- | --- | --- |
| **Function** | **Description** | **Syntax** |
| lower() | Converts all characters in a string to lowercase | string.lower() |

**Source Code :**

n = input("Enter the string: ").lower()

s = {}

for i in n:

if i in s:

s[i] += 1

else:

s[i] = 1

print(s)

**Output :**

Enter the string: Hello

{'h': 1, 'e': 1, 'l': 2, 'o': 1}

**Result :** The program is successfully executed and the output is verified.

**Experiment No : 5**

**Date :** 21/10/2024

**Aim :** Add ‘ing’ at the end of a given string. If it already ends with ‘ing’, then add

‘ly’.

**Pseudocode :**

1. Read s

2. If s ends with "ing":

Print s + "ly"

4. Else:

Print s + "ing"

**Method :**

|  |  |  |
| --- | --- | --- |
| **Function** | **Description** | **Syntax** |
| endswith() | Checks if the string ends with a specific substring | string.endswith("substring") |

**Source Code :**

s = input("Enter a string: ")

if s.endswith("ing"):

print(s + "ly")

else:

print(s + "ing")

**Output :**

Enter a string: playing

playingly

**Result :** The program is successfully executed and the output is verified.

**Experiment No : 6**

**Date :** 21/10/2024

**Aim :** Store a list of first names. Count the occurrences of ‘a’ within the list.

**Pseudocode :**

1. Initialize an empty list names

2. Read n (limit)

3. For i from 0 to n-1:

a. Read el (name)

b. Append el to names

4. Print "Occurrences of 'a' in names:"

5. For each name in names:

a. Print name and count of 'a' in name (convert name to lowercase and count 'a')

**Method :**

|  |  |  |
| --- | --- | --- |
| **Function** | **Description** | **Syntax** |
| append() | Adds an item to the end of the list | list.append(item) |
| lower() | Converts a string to lowercase | string.lower() |
| count() | Counts occurrences of a substring | string.count('substring') |

**Source Code :**

names = []

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

for i in range(n):

el = input(f"Enter name{i+1}: ")

names.append(el)

print("Occurrences of ‘a’ in names:")

for name in names:

print(f"{name}: {name.lower().count('a')}")

**Output :**

Enter limit: 3

Enter name1: Alice

Enter name2: Bob

Enter name3: Anna

Occurrences of ‘a’ in names:

Alice: 1

Bob: 0

Anna: 2

**Result :** The program is successfully executed and the output is verified.

**Experiment No : 7**

**Date :** 21/10/2024

**Aim :** Write a python program to read two lists color-list1 and color-list2. Print out all colors from

color-list1 not contained in color-list2.

**Pseudocode :**

1. Read lst1

2. Read lst2

3. Convert lst1 and lst2 to sets

4. Print set(lst1) - set(lst2)

**Method :**

|  |  |  |
| --- | --- | --- |
| **Function** | **Description** | **Syntax** |
| split() | Splits a string into a list | string.split() |
| set() | Converts a list to a set | set(list) |
| list() | Converts a set back to a list | list(set) |

**Source Code :**

lst1 = input("Enter list1 elements by space separated: ").split()

lst2 = input("Enter list2 elements by space separated: ").split()

print("Output list:")

print(list(set(lst1) - set(lst2)))

**Output :**

Enter list1 elements by space separated: apple banana cherry

Enter list2 elements by space separated: banana orange

Output list: ['apple', 'cherry']

**Result :** The program is successfully executed and the output is verified.

**Experiment No : 8**

**Date :** 21/10/2024

**Aim :** Create a list of colors from comma-separated color names entered by the user. Display first

and last colors.

**Pseudocode :**

1. Read colors as a comma-separated string

2. Split the string into a list of colors

3. Print the list of colors

4. Print the first color (colors[0])

5. Print the last color (colors[-1])

**Method :**

|  |  |  |
| --- | --- | --- |
| **Function** | **Description** | **Syntax** |
| split() | Splits a string into a list based on a separator | string.split(",") |

**Source Code :**

colors = input("Enter colors (comma separated): ").split(",")

colors = [color for color in colors]

print(colors)

print("First color:", colors[0])

print("Last color:", colors[-1])

**Output :**

Enter colors (comma separated): red,green,blue,yellow

['red', 'green', 'blue', 'yellow']

First color: red

Last color: yellow

**Result :** The program is successfully executed and the output is verified.

**Experiment No : 9**

**Date :** 21/10/2024

**Aim :** Write a program to prompt the user for a list of integers. For all values greater than 100,store

‘over’ instead.

**Pseudocode :**

1. Read input and split it into a list of integers

2. For each element in the list:

If element > 100, replace it with "over"

4. Print the modified list

**Method :**

|  |  |  |
| --- | --- | --- |
| **Function** | **Description** | **Syntax** |
| split() | Splits a string into a list based on a separator | string.split() |

**Source Code :**

lst = [int(num) for num in input("Enter list elements (space separated): ").split()]

for i in range(len(lst)):

if lst[i] > 100:

lst[i] = "over"

print(lst)

**Output :**

Enter list elements (space separated): 50 120 85 200 75

[50, 'over', 85, 'over', 75]

**Result :** The program is successfully executed and the output is verified.

**Experiment No : 10**

**Date :** 21/10/2024

**Aim** : From a list of integers, create a list after removing even numbers.

**Pseudocode :**

1. Read input and split it into a list of integers

2. Create a new list odd\_lst containing only odd numbers from lst

3. Print odd\_lst

**Method :**

|  |  |  |
| --- | --- | --- |
| **Function** | **Description** | **Syntax** |
| split() | Splits a string into a list based on a separator | string.split() |

**Source Code :**

lst = [int(num) for num in input("Enter a list of numbers (space separated): ").split()]

odd\_lst = [odd for odd in lst if odd % 2 != 0]

print(odd\_lst)

**Output :**

Enter a list of numbers (space separated): 10 15 20 25 30

[15, 25]

**Result :** The program is successfully executed and the output is verified.

**Experiment No : 11**

**Date :** 21/10/2024

**Aim** : Accept a list of words and return the length of the longest word.

**Pseudocode :**

1. Read input and split it into a list of words

2. Find the maximum length of words in the list

3. Create a new list lg\_word containing all words with the maximum length

4. Print the largest word(s) and their size

**Method :**

|  |  |  |
| --- | --- | --- |
| **Function** | **Description** | **Syntax** |
| split() | Splits a string into a list based on a separator | string.split() |
| max() | Finds the maximum value in a list | max(list) |
| len() | Returns the length of an object | len(object) |

**Source Code :**

lst = input("Enter a list of words (space separated): ").split()

max\_length = max(len(word) for word in lst)

lg\_word = [word for word in lst if len(word) == max\_length]

print(f"Largest word(s): {lg\_word}, size: {max\_length}")

**Output :**

Enter a list of words (space separated): apple banana orange pineapple

Largest word(s): ['pineapple'], size: 9

**Result :** The program is successfully executed and the output is verified.

**Experiment No : 12**

**Date :** 21/10/2024

**Aim :** Write a program to prompt the user to enter two lists of integers and check

(a) Whether lists are of the same length.

(b) Whether the list sums to the same value.

(c) Whether any value occurs in both Lists.

**Pseudocode :**

1. Read lst1 as a list of space-separated integers

2. Read lst2 as a list of space-separated integers

3. Set length = (length of lst1 == length of lst2)

4. Set lsum = (sum of lst1 == sum of lst2)

5. Set common = intersection of lst1 and lst2

6. If length:

Print "Lists lengths are the same"

Else:

Print "Lists lengths are not the same"

9. Print "Lists common elements:", common

10. If lsum:

Print "List sums are the same"

Else:

Print "List sums are not the same"

**Method :**

|  |  |  |
| --- | --- | --- |
| **Function** | **Description** | **Syntax** |
| split() | Splits a string into a list based on a separator | string.split() |
| len() | Returns the length of an object | len(object) |
| sum() | Returns the sum of all elements in a list | sum(list) |
| set() | Converts a list to a set | set(list) set(list) |

**Source Code :**

lst1 = [int(num) for num in input("Enter first list (space separated): ").split()]

lst2 = [int(num) for num in input("Enter second list (space separated): ").split()]

length = len(lst1) == len(lst2)

lsum = sum(lst1) == sum(lst2)

common = set(lst1) & set(lst2)

if length:

print("Lists lengths are the same")

else:

print("Lists lengths are not the same")

print(f"Lists common elements: {common}")

if lsum:

print("List sums are the same")

else:

print("List sums are not the same")

**Output :**

Enter first list (space separated): 1 2 3

Enter second list (space separated): 4 5 6

Lists lengths are the same

Lists common elements: set()

List sums are not the same

**Result :** The program is successfully executed and the output is verified.

**Experiment No : 13**

**Date :** 21/10/2024

**Aim :** Write a Python program to count the occurrences of each word in a line of text.

Hint: use split() function and dictionary

**Pseudocode :**

1. Read a string, convert it to lowercase, and split it into words (sentence)

2. Initialize an empty dictionary freq\_dict

3. For each word in sentence:

4. If word exists in freq\_dict:

a. Increment freq\_dict[word] by 1

6. Else:

a. Set freq\_dict[word] = 1

8. Print "Word occurrence:"

9. For each key, value in freq\_dict:

a. Print key and value

**Method :**

|  |  |  |
| --- | --- | --- |
| **Function** | **Description** | **Syntax** |
| lower() | Converts a string to lowercase | string.lower() |
| split() | Splits a string into a list based on a separator | string.split() |
| dict() | Creates an empty dictionary | dict() |
| items() | Returns the key-value pairs of a dictionary | dict.items() |

**Source Code :**

sentence = [word for word in input("Enter a string: ").lower().split()]

freq\_dict = {}

for word in sentence:

if word in freq\_dict:

freq\_dict[word] += 1

else:

freq\_dict[word] = 1

print("Word occurrence:")

for key, value in freq\_dict.items():

print(f"{key}: {value}")

**Output :**

Enter a string: Hello hello world world world

Word occurrence:

hello: 2

world: 3

**Result :** The program is successfully executed and the output is verified.

**Experiment No : 14**

**Date :** 21/10/2024

**Aim :** List comprehensions:

(a) Generate positive list of numbers from a given list of integers

(b) Square of N numbers

(c) Form a list of vowels selected from a given word

(d) Form a list ordinal value of each element of a word (Hint: use ord() to get ordinal

values)

**Pseudocode :**

1. Initialize numbers and find positive\_numbers from numbers

2. Print positive\_numbers

3. Set N = 6 and calculate squares of numbers from 1 to N

4. Print squares

5. Set word = "comprehension" and find vowels in word

6. Print vowels

7. Set word = "hello" and find ordinal\_values of characters in word

8. Print ordinal\_values

**Method :**

|  |  |  |
| --- | --- | --- |
| **Function** | **Description** | **Syntax** |
| ord() | Returns the ASCII value of a character | ord(char) |
| range() | Generates a range of numbers | range(start, stop) |

**Source Code :**

numbers = [-10, 15, 5, 7, -26, 18, 0]

positive\_numbers = [num for num in numbers if num > 0]

print(f"Positive numbers in {numbers}:", positive\_numbers)

N = 6

squares = [num \*\* 2 for num in range(1, N + 1)]

print("Squares of first 6 numbers:", squares)

word = "comprehension"

vowels = [char for char in word if char in 'aeiou']

print(f"Vowels in the word: {word}", vowels)

word = "hello"

ordinal\_values = [ord(char) for char in word]

print(f"Ordinal values in the word: {word}", ordinal\_values)

**Output :**

Positive numbers in [-10, 15, 5, 7, -26, 18, 0]: [15, 5, 7, 18]

Squares of first 6 numbers: [1, 4, 9, 16, 25, 36]

Vowels in the word: comprehension ['o', 'e', 'e', 'i', 'o']

Ordinal values in the word: hello [104, 101, 108, 108, 111]

**Result :** The program is successfully executed and the output is verified.

**Experiment No : 15**

**Date :** 21/10/2024

**Aim :** Sort dictionary in ascending and descending order.

**Pseudocode :**

1. Initialize my\_dict with key-value pairs

2. Sort my\_dict by keys in ascending order and store in keys\_asc

3. Print "Sorted by keys (ascending):", keys\_asc

4. Sort my\_dict by keys in descending order and store in keys\_desc

5. Print "Sorted by keys (descending):", keys\_desc

6. Sort my\_dict by values in ascending order and store in values\_asc

7. Print "Sorted by values (ascending):", values\_asc

8. Sort my\_dict by values in descending order and store in values\_desc

9. Print "Sorted by values (descending):", values\_desc

**Method :**

|  |  |  |
| --- | --- | --- |
| **Function** | **Description** | **Syntax** |
| |  | | --- | | sorted() |  |  | | --- | |  | | Sorts the elements of an iterable | sorted(iterable, key=key\_func, reverse=bool) |
| |  | | --- | | items() |  |  | | --- | |  | | Returns key-value pairs of a dictionary as tuples | dict.items() |
| lambda | Creates an anonymous function | lambda arguments: expression |
| dict() | Creates a dictionary from a sequence of key-value pairs | dict(sequence) |

**Source Code :**

my\_dict = {'apple': 6, 'banana': 3, 'kiwi': 4, 'orange': 10}

keys\_asc = dict(sorted(my\_dict.items()))

print("Sorted by keys (ascending):", keys\_asc)

keys\_desc = dict(sorted(my\_dict.items(), reverse=True))

print("Sorted by keys (descending):", keys\_desc)

values\_asc = dict(sorted(my\_dict.items(), key=lambda item: item[1]))

print("Sorted by values (ascending):", values\_asc)

values\_desc = dict(sorted(my\_dict.items(), key=lambda item: item[1], reverse=True))

print("Sorted by values (descending):", values\_desc)

**Output :**

Sorted by keys (ascending): {'apple': 6, 'banana': 3, 'kiwi': 4, 'orange': 10}

Sorted by keys (descending): {'orange': 10, 'kiwi': 4, 'banana': 3, 'apple': 6}

Sorted by values (ascending): {'banana': 3, 'kiwi': 4, 'apple': 6, 'orange': 10}

Sorted by values (descending): {'orange': 10, 'apple': 6, 'kiwi': 4, 'banana': 3}

**Result :** The program is successfully executed and the output is verified.

**Experiment No : 16**

**Date :** 21/10/2024

**Aim :** Merge two dictionaries**.**

**Pseudocode :**

1. Initialize dict1 with key-value pairs

2. Initialize dict2 with key-value pairs

3. Print dict1

4. Print dict2

5. Merge dict2 into dict1 using the update() method

6. Print "Merged:", dict1

**Method :**

|  |  |  |
| --- | --- | --- |
| **Function** | **Description** | **Syntax** |
| update() | Merges two dictionaries. If keys are the same, the values from the second dictionary overwrite those in the first. | dict1.update(dict2) |

**Source Code :**

dict1 = {'banana': 3, 'mango': 5}

dict2 = {'orange': 2, 'pineapple': 4}

print(dict1)

print(dict2)

dict1.update(dict2)

print(f"Merged: {dict1}")

**Output :**

{'banana': 3, 'mango': 5}

{'orange': 2, 'pineapple': 4}

Merged: {'banana': 3, 'mango': 5, 'orange': 2, 'pineapple': 4}

**Result :** The program is successfully executed and the output is verified.

**LAB CYCLE 3**

**Experiment No : 1**

**Date :** 28/10/2024

**Aim :** Write a program to find the factorial of a number.

**Pseudocode :**

1. Define a function fact(n):

a. If n == 0 or n == 1, return 1.

b. If n < 0, return -1 (error for negative input).

c. Otherwise, return n \* fact(n - 1).

2. Input an integer n from the user.

3. Call the function fact(n).

4. If the result is -1:

a. Print "Enter a positive number."

5. Otherwise:

a. Print the factorial value returned by fact(n).

**Method :**

|  |  |  |
| --- | --- | --- |
| **Function** | **Description** | **Syntax** |
| fact() | Recursively calculates the factorial of a number. Returns -1 for negative input, and factorial otherwise. | fact(n) |

**Source Code :**

def fact(n):

if n == 0 or n == 1:

return 1

elif n < 0:

return -1

else:

return n \* fact(n - 1)

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

if fact(n) == -1:

print("Enter a positive number")

else:

f = fact(n)

print("Factorial:", f)

**Output :**

Enter number: 5

Factorial: 120

**Result :** The program is successfully executed and the output is verified.

**Experiment No : 2**

**Date :** 28/10/2024

**Aim :** Generate Fibonacci series of N terms.

**Pseudocode :**

1. Input the number of terms (n) from the user.

2. Initialize variables:

a. n1 = 0 (first term)

b. n2 = 1 (second term)

c. next = n1 (to store the current term in the sequence)

d. count = 1 (term counter)

3. Print "Fibonacci series:".

4. Use a loop while count <= n:

a. Print the current term (next).

b. Increment the counter (count).

c. Update the terms:

i. n1, n2 = n2, next (swap to move forward in the series).

ii. next = n1 + n2 (compute the next term in the sequence).

**Source Code :**

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

n1 = 0

n2 = 1

next = n1

count = 1

print("Fibonacci series:")

while count <= n:

print(next)

count += 1

n1, n2 = n2, next

next = n1 + n2

**Output :**

Enter limit: 5

Fibonacci series:

0

1

1

2

3

**Result :** The program is successfully executed and the output is verified.

**Experiment No : 3**

**Date :** 28/10/2024

**Aim :** Write a program to find the sum of all items in a list. [Using for loop]

**Pseudocode :**

1. Initialize mylist = [20, 24, 32, 60, 50]

2. Set total = 0

3. For each element i in mylist:

a. Add mylist[i] to total

4. Print "List is =", mylist

5. Print "Total sum of elements =", total

**Source Code :**

mylist=[20,24,32,60,50]

total=0

for i in range (0,len(mylist)):

total=total+mylist[i]

print("list is :",mylist)

print("total sum of elements:",total)

**Output :**

List is: [20, 24, 32, 60, 50]

Total sum of elements: 186

**Result :** The program is successfully executed and the output is verified.

**Experiment No : 4**

**Date :** 28/10/2024

**Aim** : Generate a list of four digit numbers in a given range with all their digits even and the number is a perfect square.

**Pseudocode :**

1. Define function all\_digits\_even(number):

a. For each digit in the number:

i. If digit is odd:

Return False

d. Return True

2. Set start = 1000

3. Set end = 9999

4. Create an empty list result

5. For i from start to end:

a. Set root = integer square root of i

b. If root \* root equals i and all\_digits\_even(i) is true:

i. Add i to result

6. Print result

**Method :**

|  |  |  |
| --- | --- | --- |
| **Function** | **Description** | **Syntax** |
| math.isqrt() | Computes the integer square root of a number. | math.isqrt(number) |

**Source Code :**

import math

def all\_digits\_even(number):

for digit in str(number):

if int(digit) % 2 != 0: # If any digit is odd

return False

return True

start = 1000

end = 9999

result = []

for i in range(start, end + 1):

root = math.isqrt(i) # Calculate integer square root

if root \* root == i and all\_digits\_even(i): # Check if it's a perfect square and all digits are even

result.append(i)

print("Four-digit numbers with all even digits that are perfect squares:", result)

**Output :**

Four-digit numbers with all even digits that are perfect squares: [1600, 6400, 3600, 2500, ...]

**Result :** The program is successfully executed and the output is verified.

**Experiment No : 5**

**Date :** 28/10/2024

**Aim :** Write a program using a for loop to print the multiplication table of n, where n is entered by the user.

**Pseudocode :**

1. Get the number `n` from the user.

2. Print the message "Multiplication Table of n".

3. For each `i` from 1 to 10:

a. Print the result of `n \* i`.

**Source Code :**

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

print(f"Multiplication Table of {n}:")

for i in range(1, 11):

print(f"{n} x {i} = {n \* i}")

**Output :**

Enter a number: 5

Multiplication Table of 5:

5 x 1 = 5

5 x 2 = 10

5 x 3 = 15

5 x 4 = 20

5 x 5 = 25

5 x 6 = 30

5 x 7 = 35

5 x 8 = 40

5 x 9 = 45

5 x 10 = 50

**Result :** The program is successfully executed and the output is verified.

**Experiment No : 6**

**Date :** 28/10/2024

**Aim :** Write a program to display alternate prime numbers till N (obtain N from the user).

**Pseudocode :**

1. Input N (the upper limit for checking primes)

2. Set count = 0 (initialize the prime counter)

3. Print "Alternate prime numbers up to N:"

4. For each num from 2 to N:

a. Set is\_prime = True

b. For each i from 2 to sqrt(num):

i. If num % i == 0:

- Set is\_prime = False

- Break the inner loop

c. If is\_prime is True:

i. If count % 2 == 0:

- Print num

ii. Increment count by 1

**Source Code :**

N = int(input("Enter the value of N: "))

count = 0

print("Alternate prime numbers up to", N, "are:")

for num in range(2, N + 1):

is\_prime = True

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

if num % i == 0:

is\_prime = False

break

if is\_prime:

if count % 2 == 0:

print(num)

count += 1

**Output :**

Enter the value of N: 20

Alternate prime numbers up to 20 are:

2

5

11

17

**Result :** The program is successfully executed and the output is verified.

**Experiment No : 7**

**Date :** 28/10/2024

**Aim :** Write a program to compute and display the sum of all integers that are divisible by 6 but not by 4, and that lie below a user-given upper limit.

**Pseudocode :**

1. Input upper\_limit (the upper range for numbers to check)

2. Set total\_sum = 0 (initialize the sum variable)

3. For each num from 1 to upper\_limit - 1:

a. If num is divisible by 6 (num % 6 == 0) and not divisible by 4 (num % 4 != 0):

i. Add num to total\_sum

4. Print total\_sum as the sum of numbers divisible by 6 but not by 4

**Source Code :**

upper\_limit = int(input("Enter the upper limit: "))

total\_sum = 0

for num in range(1, upper\_limit):

if num % 6 == 0 and num % 4 != 0:

total\_sum += num

print("The sum of all integers below", upper\_limit, "that are divisible by 6 but not by 4 is:",

total\_sum)

**Output :**

Enter the upper limit: 50

The sum of all integers below 50 that are divisible by 6 but not by 4 is: 72

**Result :** The program is successfully executed and the output is verified.

**Experiment No : 8**

**Date:** 28/10/2024

**Aim :** Calculate the sum of the digits of each number within a specified range (from 1 to a user-defined upper limit). Print the sum only if it is prime.

**Pseudocode :**

1. Define function sum\_of\_digits(n):

a. Convert n to string

b. Convert each character back to integer and sum them

c. Return the sum of digits

2. Define function is\_prime(num):

a. If num < 2, return False

b. For i from 2 to sqrt(num) + 1:

i. If num % i == 0, return False

c. Return True (num is prime)

3. Define function sum\_digits\_in\_range(upper\_limit):

a. For each num from 1 to upper\_limit:

i. Calculate digit\_sum using sum\_of\_digits(num)

ii. If digit\_sum is prime:

- Print "Sum of digits of num is digit\_sum, which is a prime"

4. Input upper\_limit from user

5. Call sum\_digits\_in\_range(upper\_limit)

**Source Code :**

def sum\_of\_digits(n):

return sum(int(digit) for digit in str(n))

def is\_prime(num):

if num < 2:

return False

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

if num % i == 0:

return False

return True

def sum\_digits\_in\_range(upper\_limit):

for num in range(1, upper\_limit + 1):

digit\_sum = sum\_of\_digits(num)

if is\_prime(digit\_sum):

print(f"Sum of digits of {num} is {digit\_sum}, which is a prime.")

limit = int(input("Enter an upper limit: "))

sum\_digits\_in\_range(limit)

**Output :**

Enter an upper limit: 20

Sum of digits of 2 is 2, which is a prime.

Sum of digits of 3 is 3, which is a prime.

Sum of digits of 5 is 5, which is a prime.

Sum of digits of 7 is 7, which is a prime.

Sum of digits of 11 is 2, which is a prime.

Sum of digits of 12 is 3, which is a prime.

Sum of digits of 14 is 5, which is a prime.

Sum of digits of 16 is 7, which is a prime.

Sum of digits of 20 is 2, which is a prime.

**Result :** The program is successfully executed and the output is verified.

**Experiment No : 9**

**Date :** 11/11/2024

**Aim :** A number is input through the keyboard. Write a program to determine if it’s palindromic**.**

**Pseudocode :**

1. Input number (as a string)

2. Set is\_palindrome = True

3. Get the length of the number

4. For i from 0 to length // 2 - 1:

a. If number[i] != number[length - 1 - i]:

i. Set is\_palindrome = False

ii. Break the loop

5. If is\_palindrome is True:

a. Print "number is a palindromic number."

6. Else:

a. Print "number is not a palindromic number."

**Source Code :**

number = input("Enter a number: ")

is\_palindrome = True

length = len(number)

for i in range(length // 2):

if number[i] != number[length - 1 - i]:

is\_palindrome = False

break

if is\_palindrome:

print(f"{number} is a palindromic number.")

else:

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

**Output :**

Enter a number: 12321

12321 is a palindromic number.

**Result :** The program is successfully executed and the output is verified.

**Experiment No : 10**

**Date :** 11/11/2024

**Aim :** Write a program to generate all factors of a number. [use while loop]

**Pseudocode :**

1. Input number

2. Initialize factor = 1

3. Print "Factors of number are:"

4. While factor <= number:

a. If number % factor == 0:

i. Print factor

b. Increment factor by 1

**Source Code :**

number = int(input("Enter a number to find its factors: "))

factor = 1

print(f"Factors of {number} are:")

while factor <= number:

if number % factor == 0:

print(factor)

factor += 1

**Output :**

Enter a number to find its factors: 12

Factors of 12 are:

1

2

3

4

6

12

**Result :** The program is successfully executed and the output is verified.

**Experiment No : 11**

**Date :** 11/11/2024

**Aim** : Write a program to find whether the given number is an Armstrong number or not. [use while loop]

**Pseudocode :**

1. Input number

2. Set original\_number = number

3. Set sum\_of\_powers = 0

4. Calculate num\_digits = length of the number (convert to string and find length)

5. While number > 0:

a. Extract the last digit: digit = number % 10

b. Add digit raised to the power of num\_digits to sum\_of\_powers

c. Remove the last digit: number = number // 10

6. If original\_number == sum\_of\_powers:

a. Print "original\_number is an Armstrong number."

7. Else:

a. Print "original\_number is not an Armstrong number."

**Source Code :**

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

original\_number = number

sum\_of\_powers = 0

num\_digits = len(str(number))

while number > 0:

digit = number % 10

sum\_of\_powers += digit \*\* num\_digits

number //= 10

if original\_number == sum\_of\_powers:

print(f"{original\_number} is an Armstrong number.")

else:

print(f"{original\_number} is not an Armstrong number.")

**Output :**

Enter a number: 153

153 is an Armstrong number.

**Result :** The program is successfully executed and the output is verified.

**Experiment No : 12**

**Date** 11/11/2024

**Aim :** Display the given pyramid with the step number accepted from the user. Eg: N=4

1

2 4

3 6 9

4 8 12 16

**Pseudocode :**

1. Input N (number of steps)

2. For i from 1 to N:

a. Initialize an empty list row

b. For j from 1 to i:

i. Calculate value = i \* j

ii. Append value as a string to row

c. Join row elements with a space and print the result

**Method :**

|  |  |  |
| --- | --- | --- |
| **Function** | **Description** | **Syntax** |
| join() | Joins elements of a list into a single string with a separator. | " ".join(list) |

**Source Code :**

N = int(input("Enter the number of steps: "))

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

row = []

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

value = j \* i

row.append(str(value))

print(" ".join(row))

**Output :**

Enter the number of steps: 5

1

2 4

3 6 9

4 8 12 16

5 10 15 20 25

**Result :** The program is successfully executed and the output is verified.

**Experiment No : 13**

**Date :** 11/11/2024

**Aim :** Construct following pattern using nested loop

\*

\* \*

\* \* \*

\* \* \* \*

\* \* \* \* \*

\* \* \* \*

\* \* \*

\* \*

\*

**Pseudocode :**

1. Set rows = 5

2. For i from 1 to rows:

a. For j from 0 to i - 1:

i. Print "\*", without a newline

b. Print a newline

3. For i from rows - 1 to 1:

a. For j from 0 to i - 1:

i. Print "\*", without a newline

b. Print a newline

**Method :**

|  |  |  |
| --- | --- | --- |
| **Function** | **Description** | **Syntax** |
| end=" " | Ensures the next print statement continues on the same line. | print(value, end=" ") |

**Source Code :**

rows = 5

# Top half of the diamond

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

for j in range(i):

print("\*", end=" ")

print()

# Bottom half of the diamond

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

for j in range(i):

print("\*", end=" ")

print()

**Output :**

\*

\* \*

\* \* \*

\* \* \* \*

\* \* \* \* \*

\* \* \* \*

\* \* \*

\* \*

\*

**Result :** The program is successfully executed and the output is verified.

**LAB CYCLE 4**

**Experiment No : 1**

**Date :** 11/11/2024

**Aim :** Write a program to print the Fibonacci series using recursion.

**Pseudocode :**

1. Define a function `fibonacci(n)`:

a. If `n` is less than or equal to 1:

i. Return `n`.

b. Else:

i. Return `fibonacci(n - 1) + fibonacci(n - 2)`.

2. Take input from the user and store it in variable `num\_terms`.

3. If `num\_terms` is less than or equal to 0:

a. Print "Please enter a positive integer."

4. Else:

a. Print "Fibonacci sequence:".

b. For each `i` from 0 to `num\_terms - 1`:

i. Print `fibonacci(i)`.

**Source Code :**

def fibonacci(n):

if n <= 1:

return n

else:

return fibonacci(n - 1) + fibonacci(n - 2)

num\_terms = int(input("Enter the number of terms: "))

if num\_terms <= 0:

print("Please enter a positive integer")

else:

print("Fibonacci sequence:")

for i in range(num\_terms):

print(fibonacci(i))

**Output :**

Enter the number of terms: 5

Fibonacci sequence:

0

1

1

2

3

**Result :** The program is successfully executed and the output is verified.

**Experiment No : 2**

**Date :** 11/11/2024

**Aim :** Write the to implement a menu-driven calculator. Use separate functions for the different

operations.

**Pseudocode :**

1. Define a function `add(x, y)`:

a. Return `x + y`.

2. Define a function `subtract(x, y)`:

a. Return `x - y`.

3. Define a function `multiply(x, y)`:

a. Return `x \* y`.

4. Define a function `divide(x, y)`:

a. If `y` equals 0:

i. Return "Error! Division by zero."

b. Else:

i. Return `x / y`.

5. Loop until the user exits:

a. Display the menu of operations:

i. "1. Addition"

ii. "2. Subtraction"

iii. "3. Multiplication"

iv. "4. Division"

v. "5. Exit".

b. Take input from the user and store it in `choice`.

c. If `choice` equals '5':

i. Print "Exiting..".

ii. Break the loop.

d. If `choice` is in ('1', '2', '3', '4'):

i. Take input for `num1` and `num2`.

ii. Perform the operation based on the value of `choice`:

- If `choice` is '1', call `add(num1, num2)`.

- If `choice` is '2', call `subtract(num1, num2)`.

- If `choice` is '3', call `multiply(num1, num2)`.

- If `choice` is '4', call `divide(num1, num2)`.

iii. Print the result.

e. Else:

i. Print "Invalid choice!".

**Source Code :**

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."

return x / y

while True:

print("\nSelect an operation:")

print("1. Addition")

print("2. Subtraction")

print("3. Multiplication")

print("4. Division")

print("5. Exit")

choice = input("Enter your choice: ")

if choice == '5':

print("Exiting..")

break

if choice in ('1', '2', '3', '4'):

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

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

if choice == '1':

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

elif choice == '2':

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

elif choice == '3':

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

elif choice == '4':

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

else:

print("Invalid choice!")

**Output :**

Select an operation:

1. Addition

2. Subtraction

3. Multiplication

4. Division

5. Exit

Enter your choice: 1

Enter first number: 12

Enter second number: 8

Result: 12.0 + 8.0 = 20.0

Enter your choice: 5

Exiting..

**Result :** The program is successfully executed and the output is verified.

**Experiment No : 3**

**Date :** 11/11/2024

**Aim :** Write a program to print the nth prime number. [Use function to check whether a number is

prime or not]

**Pseudocode :**

1. Define a function `is\_prime(num)`:

a. If `num` is less than or equal to 1:

i. Return False.

b. For each integer `i` from 2 to the square root of `num` (inclusive):

i. If `num` is divisible by `i`:

- Return False.

c. Return True.

2. Define a function `nth\_prime(n)`:

a. Initialize `count = 0` and `num = 2`.

b. Loop indefinitely:

i. If `is\_prime(num)` is True:

- Increment `count` by 1.

- If `count` equals `n`:

- Return `num`.

ii. Increment `num` by 1.

3. Take input from the user and store it in `n`.

4. If `n` is less than or equal to 0:

a. Print "Please enter a positive integer greater than 0."

5. Else:

a. Call `nth\_prime(n)` and store the result in a variable.

b. Print the result as "The N-th prime number is: result."

**Source Code :**

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

def nth\_prime(n):

count = 0

num = 2

while True:

if is\_prime(num):

count += 1

if count == n:

return num

num += 1

n = int(input("Enter the position of the prime number you want to find (n): "))

if n <= 0:

print("Please enter a positive integer greater than 0.")

else:

print(f"The {n}-th prime number is: {nth\_prime(n)}")

**Output :**

Enter the position of the prime number you want to find (n): 5

The 5-th prime number is: 11

**Result :** The program is successfully executed and the output is verified.

**Experiment No : 4**

**Date :** 11/11/2024

**Aim** : Write lambda functions to find the area of square, rectangle and triangle.

**Pseudocode :**

1. Define lambda functions:

a. area\_square = side \* side

b. area\_rectangle = length \* width

c. area\_triangle = 0.5 \* base \* height

2. While True:

a. Display menu:

i. "1. Square"

ii. "2. Rectangle"

iii. "3. Triangle"

iv. "4. Exit"

b. Input choice

c. If choice == '1':

i. Input side

ii. Calculate and print area using area\_square

d. Else If choice == '2':

i. Input length and width

ii. Calculate and print area using area\_rectangle

e. Else If choice == '3':

i. Input base and height

ii. Calculate and print area using area\_triangle

f. Else If choice == '4':

i. Print "Exiting..." and break loop

g. Else:

i. Print "Invalid choice!"

**Method :**

|  |  |  |
| --- | --- | --- |
| **Function** | **Description** | **Syntax** |
| lambda | Creates anonymous functions. | lambda arguments: expression |

**Source Code :**

area\_square = lambda side: side \* side

area\_rectangle = lambda length, width: length \* width

area\_triangle = lambda base, height: 0.5 \* base \* height

while True:

print("\nChoose a shape to calculate the area:")

print("1. Square")

print("2. Rectangle")

print("3. Triangle")

print("4. Exit")

choice = input("Enter your choice: ")

if choice == '1':

side = float(input("Enter the side length of the square: "))

print(f"The area of the square is: {area\_square(side)}")

elif choice == '2':

length = float(input("Enter the length of the rectangle: "))

width = float(input("Enter the width of the rectangle: "))

print(f"The area of the rectangle is: {area\_rectangle(length, width)}")

elif choice == '3':

base = float(input("Enter the base length of the triangle: "))

height = float(input("Enter the height of the triangle: "))

print(f"The area of the triangle is: {area\_triangle(base, height)}")

elif choice == '4':

print("Exiting...")

break

else:

print("Invalid choice!")

**Output :**

Choose a shape to calculate the area:

1. Square

2. Rectangle

3. Triangle

4. Exit

Enter your choice: 1

Enter the side length of the square: 5

The area of the square is: 25.0

**Result :** The program is successfully executed and the output is verified.

**Experiment No : 5**

**Date :** 11/11/2024

**Aim :** Write a program to display powers of 2 using anonymous function. [ Hint use map and lambda

function)

**Pseudocode :**

1. Function calculate\_powers\_of\_2(numbers):

a. Return [2^x for x in numbers]

2. Function main:

a. Input space-separated numbers

b. Convert input to list of integers

c. Call calculate\_powers\_of\_2(numbers) and print the result

d. Handle invalid input with an error message

3. If executed directly:

a. Call main

**Method :**

|  |  |  |
| --- | --- | --- |
| **Function** | **Description** | **Syntax** |
| split() | Splits the string into a list based on spaces. | string.split() |
| list() | Converts an iterable into a list. | list(iterable) |
| map() | Applies a function to every element of an iterable. | map(function, iterable) |

**Source Code :**

lst = [int(num) for num in input("Enter list elements (space-separated): ").split()]

powers\_of\_2 = list(map(lambda x: 2 \*\* x, lst))

print(f"\nPowers of 2: {powers\_of\_2}")

**Output :**

Enter list elements (space-separated): 1 2 3 4 5

Powers of 2: [2, 4, 8, 16, 32]

**Result :** The program is successfully executed and the output is verified.

**Experiment No : 6**

**Date :** 11/11/2024

**Aim :** Write a program to display multiples of 3 using anonymous function. [ Hint use filter and

lambda function].

**Pseudocode :**

1. Read a list of numbers from the user:

a. Use input() to take the list of numbers separated by spaces.

b. Split the input string and convert each element to an integer using map().

2. Use filter() to filter out numbers divisible by 3:

a. Define a lambda function `lambda x: x % 3 == 0` to check divisibility by 3.

b. Apply the lambda function using filter() to get multiples of 3.

3. Print the multiples of 3.

**Method :**

|  |  |  |
| --- | --- | --- |
| **Function** | **Description** | **Syntax** |
| filter() | Filters elements of an iterable based on a condition. | filter(function, iterable) |

**Source Code :**

user\_input = input("Enter a list of numbers separated by spaces: ")

numbers = list(map(int, user\_input.split()))

multiples\_of\_3 = list(filter(lambda x: x % 3 == 0, numbers))

print("Multiples of 3:", multiples\_of\_3)

**Output :**

Enter a list of numbers separated by spaces: 3 2 4 6 8 9 12 23

Multiples of 3: [3, 6, 9, 12]

**Result :** The program is successfully executed and the output is verified.

**Experiment No : 7**

**Date :** 11/11/2024

**Aim :** Write a program to sum the series 1/1! + 4/2! + 27/3! + ….. + nth term. [ Hint Use a function

to find the factorial of a number].

**Pseudocode :**

1. Define factorial(num):

a. If num is 0 or 1, return 1.

b. Otherwise, initialize result to 1.

c. For each number i from 2 to num:

i. Multiply result by i.

d. Return result as the factorial of num.

2. Define sum\_series(n):

a. Initialize total to 0.

b. For each number i from 1 to n:

i. Calculate (i^i) / (i!).

ii. Add the term to total.

c. Return total.

3. Read the value of n (number of terms) from the user.

4. Call the sum\_series(n) function to calculate the sum of the series.

5. Print the sum of the series up to n terms.

**Method :**

|  |  |  |
| --- | --- | --- |
| **Function** | **Description** | **Syntax** |
| factorial() | Computes the factorial of a number. | factorial(num) |
| sum\_series() | Calculates the sum of the series iii!\frac{i^i}{i!}i!ii​ up to n terms. | sum\_series(n) |

**Source Code :**

def factorial(num):

if num == 0 or num == 1:

return 1

else:

result = 1

for i in range(2, num + 1):

result \*= i

return result

def sum\_series(n):

total = 0

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

total += (i \*\* i) / factorial(i) # n^n / n!

return total

n = int(input("Enter the value of n: "))

result = sum\_series(n)

print(f"The sum of the series up to {n} terms is: {result}")

**Output :**

Enter the value of n: 4

The sum of the series up to 4 terms is: 18.166666666666664

**Result :** The program is successfully executed and the output is verified.

**Experiment No : 8**

**Date :** 11/11/2024

**Aim :** Write a function called compare which takes two strings S1 and S2 and an

integer n as arguments. The function should return True if the first n characters

of both the strings are the same else the function should return False.

**Pseudocode :**

1. Define function `compare(S1, S2, n)`:

a. Return True if the first `n` characters of `S1` are equal to the first `n` characters of `S2`.

b. Otherwise, return False.

2. Read the first string `S1` from the user.

3. Read the second string `S2` from the user.

4. Read the number `n` from the user to specify how many characters to compare.

5. Call the `compare(S1, S2, n)` function with `S1`, `S2`, and `n` as arguments.

6. Print the result of the comparison (True or False).

**Method :**

|  |  |  |
| --- | --- | --- |
| **Function** | **Description** | **Syntax** |
| compare() | Compares the first n characters of two strings. | compare(S1, S2, n) |

**Source Code :**

def compare(S1, S2, n):

return S1[:n] == S2[:n]

S1 = input("Enter the first string: ")

S2 = input("Enter the second string: ")

n = int(input("Enter the number of characters to compare: "))

result = compare(S1, S2, n)

print("Result:", result)

**Output :**

Enter the first string: hello

Enter the second string: helios

Enter the number of characters to compare: 3

Result: True

**Result :** The program is successfully executed and the output is verified.

**Experiment No : 9**

**Date :** 11/11/2024

**Aim :** Write a program to add variable length integer arguments passed to the function. [Also demo

the use of docstrings].

**Pseudocode :**

1. Define add\_integers(\*args):

a. Return the sum of all the integers in args using the sum() function.

2. Call add\_integers with 1, 2, 3 as arguments.

a. Print the result.

3. Call add\_integers with 5, 10, 15, 20 as arguments.

a. Print the result.

**Method :**

|  |  |  |
| --- | --- | --- |
| **Function** | **Description** | **Syntax** |
| add\_integers(\*args) | Accepts a variable number of integers and returns their sum. | add\_integers(\*args) |

**Source Code :**

def add\_integers(\*args):

"""

This function takes a variable number of integer arguments and returns their sum.

Parameters:

\*args (int): The integers to be summed up.

Returns:

int: The sum of all the integer arguments.

Example:

>>> add\_integers(1, 2, 3)

6

>>> add\_integers(5, 10, 15, 20)

50

"""

return sum(args)

result = add\_integers(1, 2, 3)

print("Sum of numbers:", result)

result = add\_integers(5, 10, 15, 20)

print("Sum of numbers:", result)

**Output :**

Sum of numbers: 6

Sum of numbers: 50

**Result :** The program is successfully executed and the output is verified.

**Experiment No : 10**

**Date :** 11/11/2024

**Aim :** Write a program using functions to implement these formulae for permutations and

combinations. The Number of permutations of n objects taken r at a time: p(n, r) = n!/(n − r)!.

The Number of combinations of n objects taken r at a time is: c(n, r) = n!/(r! ∗ (n − r)!).

**Pseudocode :**

1. Define function `permutations(n, r)`:

a. Calculate the factorial of `n` and `n - r`.

b. Return the result of `n! / (n - r)!`.

2. Define function `combinations(n, r)`:

a. Calculate the factorial of `n`, `r`, and `n - r`.

b. Return the result of `n! / (r! \* (n - r)!)`.

3. Get input values `n` (total objects) and `r` (objects taken at a time).

4. Call the `permutations(n, r)` function and print the result.

5. Call the `combinations(n, r)` function and print the result.

**Method :**

|  |  |  |
| --- | --- | --- |
| **Function** | **Description** | **Syntax** |
| math.factorial(x) | Returns the factorial of the number x. | math.factorial(x) |
| permutations(n, r) | Calculates the number of permutations of n objects taken r at a time. | permutations(n, r) |
| combinations(n, r) | Calculates the number of combinations of n objects taken r at a time. | combinations(n, r) |

**Source Code :**

import math

def permutations(n, r):

return math.factorial(n) // math.factorial(n - r)

def combinations(n, r):

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

n = int(input("Enter total number of objects (n): "))

r = int(input("Enter number of objects taken at a time (r): "))

print(f"P({n}, {r}) = {permutations(n, r)}")

print(f"C({n}, {r}) = {combinations(n, r)}")

**Output :**

Enter total number of objects (n): 5

Enter number of objects taken at a time (r): 3

P(5, 3) = 60

C(5, 3) = 10

**Result :** The program is successfully executed and the output is verified.

**Experiment No : 2**

**Date :** 30/09/2024

**Aim :**

**Pseudocode :**

**Method :**

|  |  |  |
| --- | --- | --- |
| **Function** | **Description** | **Syntax** |
|  |  |  |
|  |  |  |

**Source Code :**

**Output :**

**Result :** The program is successfully executed and the output is verified.