**Python program day-3** Date:22/02/2023

*1.Write a python program to find the difference between the sum of the squares of the first two hundred natural numbers and the square of the sum.*

**Program:**

def sum\_of\_squares(n):

return sum(i\*\*2 for i in range(1, n+1))

def square\_of\_sum(n):

return sum(range(1, n+1))\*\*2

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

n =int(input("enter the n value:"))

sum\_squares = sum\_of\_squares(n)

square\_sum = square\_of\_sum(n)

difference = square\_sum - sum\_squares

print("The sum of the squares of the first twenty natural numbers is,", sum\_squares)

print("The square of the sum of the first twenty natural numbers is,", square\_sum)

print("Hence the difference between the sum of the squares and the square of the sum is", square\_sum, "-", sum\_squares, "=", difference, "Output:", difference

**Input:**

The sum of the squares of the first twenty natural numbers is,

12+22+32+.....+202 = 2870

The square of the sum of the first twenty natural numbers is,

(1 + 2 + ... + 10)2 = 44100

Hence the difference between the sum of the squares of the first twenty natural

numbers and the square of the sum is 44100 - 2870 = 41230

**Output:** 401323300<br>

*2. Given an integer n, return the number of strings of length n that consist only of vowels*

*(a, e, i, o, u) and are lexicographically sorted.*

*A string s is lexicographically sorted if for all valid i, s[i] is the same as or comes*

*before s[i+1] in the alphabet.*

**Program:**

def count\_sorted\_vowel\_strings(n: int) -> int:

vowels = "aeiou"

count = 0

def generate\_strings(curr\_string, length):

nonlocal count

if length == 0:

count += 1

return

for vowel in vowels:

if not curr\_string or curr\_string[-1] <= vowel:

generate\_strings(curr\_string + vowel, length - 1)

generate\_strings("", n)

return count

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

n = int(input("Enter the length of strings (n): "))

result = count\_sorted\_vowel\_strings(n)

print("The number of strings of length", n, "that consist only of vowels and are lexicographically sorted is:", result)

**Input:** n = 2

**Output:** 15

**Explanation:** The 15 sorted strings that consist of vowels only are

["aa","ae","ai","ao","au","ee","ei","eo","eu","ii","io","iu","oo","ou","uu"].

*3. Given two binary strings a and b, return their sum as a binary string.*

* *a and b consist only of '0' or '1' characters.*
* *Each string does not contain leading zeros except for the zero itself.*

**Program:**

def addBinary(a: str, b: str) -> str:

result = []

carry = 0

i, j = len(a) - 1, len(b) - 1

while i >= 0 or j >= 0 or carry:

digit\_a = int(a[i]) if i >= 0 else 0

digit\_b = int(b[j]) if j >= 0 else 0

sum\_bits = digit\_a + digit\_b + carry

carry = sum\_bits // 2

result.append(str(sum\_bits % 2))

i -= 1

j -= 1

return ''.join(result[::-1])

a = input("enter the value of a:")

b = input("enter the value of b:")

print(addBinary(a, b))

**Input**: a = "1010", b = "1011"

**Output:** "10101"

*4. A valid number can be split up into these components (in order):*

1. *A decimal number or an integer.*
2. *(Optional) An 'e' or 'E', followed by an integer.*

*A decimal number can be split up into these components (in order):*

1. *(Optional) A sign character (either '+' or '-').*
2. *One of the following formats:*
   1. *One or more digits, followed by a dot '.'.*
   2. *One or more digits, followed by a dot '.', followed by one or more digits.*
   3. *A dot '.', followed by one or more digits.*

*An integer can be split up into these components (in order):*

1. *(Optional) A sign character (either '+' or '-').*
2. *One or more digits.*

*For example, all the following are valid numbers: ["2", "0089", "-0.1", "+3.14", "4.", "-.9", "2e10", "-90E3", "3e+7", "+6e-1", "53.5e93", "-123.456e789"], while the following are not valid numbers: ["abc", "1a", "1e", "e3", "99e2.5", "--6", "-+3", "95a54e53"].*

*Given a string s, return true if s is a valid number.*

**Program:**

def isNumber(s: str) -> bool:

s = s.strip() # Remove leading and trailing whitespace

seen\_dot = False

seen\_e = False

seen\_digit = False

seen\_number = False

for i, char in enumerate(s):

if char.isdigit():

seen\_digit = True

seen\_number = True

elif char in {'+', '-'}:

if i > 0 and s[i - 1] not in {'e', 'E'}:

return False

elif char == '.':

if seen\_dot or seen\_e:

return False

seen\_dot = True

elif char in {'e', 'E'}:

if seen\_e or not seen\_digit:

return False

seen\_e = True

seen\_digit = False # Reset for the exponent part

else:

return False

return seen\_number

* **Input:** s = "0"

**Output:** true

* **Input:** s = "e"

**Output**: false

*5.A party has been organised on a cruise. The party is organised for a limited time (T). The number of guests entering (E[i]) and leaving (L[i]) the party at every hour is represented as items of the list. The task is to find the maximum number of guests present on the cruise at any given instance within T hours.*

**Program:**

def maxGuests(T, E, L):

events = []

max\_guests = 0

current\_guests = 0

for i in range(len(E)):

events.append((i, E[i]))

events.append((i, -L[i]))

events.sort()

for event in events:

current\_guests += event[1]

max\_guests = max(max\_guests, current\_guests)

return max\_guests

T = int(input("enter the T value:"))

E = [7,0,5,1,3]

L = [1,2,1,3,4]

print("the max no.of guest on cruise at an instance is:",maxGuests(T, E, L))

**Sample Input:**

5 ---> Value of T

[7,0,5,1,3] --->E[], element of E[0] to E[N-1], where input each element is separated by new line

[1,2,1,3,4] ----->L[],element of L[0] to L[N-1], where input each element is separated by new line

**Sample Output**:

8 -----> Maximum number of guests on cruise at an instance.

6.The year is divided into four seasons: spring, summer, fall and winter. While the exact dates that the seasons change vary a little bit from year to year because of the way that the calendar is constructed, we will use the following dates for this exercise:

Season First day

Summer March 20

Spring June 21

Fall September 22

Winter December 21

Create a program that reads a month and day from the user. The user will enter the name of the month as a string, followed by the day within the month as an integer. Then your program should display the season associated with the date that was entered. Note: Enter First three letter for month example: Jan for January, Feb for February and so on....and first letter of the month should be capital

**Program:**

def get\_season(month, day):

# Define the start dates of each season

seasons = {

"Spring": [(3, 20), (4, 30)],

"Summer": [(6, 21), (8, 31)],

"Fall": [(9, 22), (11, 30)],

"Winter": [(12, 21), (2, 28)]

}

# Determine the season based on the given month and day

for season, (start\_date, end\_date) in seasons.items():

if (month == start\_date[0] and day >= start\_date[1]) or (month == end\_date[0] and day <= end\_date[1]):

return season

# If no season is found, return None

return None

def main():

# Dictionary to map month names to their corresponding numbers

month\_numbers = {

"Jan": 1, "Feb": 2, "Mar": 3, "Apr": 4,

"May": 5, "Jun": 6, "Jul": 7, "Aug": 8,

"Sep": 9, "Oct": 10, "Nov": 11, "Dec": 12

}

# Get input from the user

month = input("Enter the month (First three letters, e.g., Jan): ").capitalize()

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

# Get the month number from the dictionary

month\_number = month\_numbers.get(month)

# Check if the month is valid

if month\_number is None or day < 1 or day > 31:

print("Invalid input. Please enter a valid month and day.")

else:

# Get the season and display the result

season = get\_season(month\_number, day)

if season:

print(f"The season for {month} {day} is {season}.")

else:

print("Invalid date. Please enter a valid date.")

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

main()

**Input:**

Enter the month: march

Enter the date: 21

**Output:**

The season is currently summer

# *7.Python program to remove words that are common in two Strings*

*Given two strings S1 and S2, representing sentences, the task is to print both sentences after removing all words which are present in both sentences.*

**Program:**

def remove\_common\_words(s1, s2):

# Split the sentences into lists of words

words1 = s1.split()

words2 = s2.split()

# Create sets of unique words for both sentences

set1 = set(words1)

set2 = set(words2)

# Find the words that are common in both sentences

common\_words = set1.intersection(set2)

# Remove common words from both sentences

updated\_s1 = ' '.join(word for word in words1 if word not in common\_words)

updated\_s2 = ' '.join(word for word in words2 if word not in common\_words)

return updated\_s1, updated\_s2

# Test the function

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

S1 = "This is a sample sentence"

S2 = "This is another sentence"

updated\_S1, updated\_S2 = remove\_common\_words(S1, S2)

print("Sentence 1 after removing common words:", updated\_S1)

print("Sentence 2 after removing common words:", updated\_S2)

*8. Given a list of strings strs, group* ***the anagrams*** *together. You can return the answer in* ***any order****.An* ***Anagram*** *is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once.*

**Program:**

def groupAnagrams(strs):

anagrams = {}

for s in strs:

key = ''.join(sorted(s))

if key not in anagrams:

anagrams[key] = [s]

else:

anagrams[key].append(s)

return list(anagrams.values())

# Test Cases

test\_cases = [

["eat", "tea", "tan", "ate", "nat", "bat"], # Test case 1

[""], # Test case 2

["a"], # Test case 3

]

# Applying function to each test case

results = []

for tc in test\_cases:

result = groupAnagrams(tc)

results.append(result)

# Displaying results for each test case

for i, result in enumerate(results, start=1):

print(f"Test Case {i}: Input: {test\_cases[i-1]}")

print(f"Output: {result}\n")

**Input:** strs = ["eat","tea","tan","ate","nat","bat"]

**Output:** [["bat"],["nat","tan"],["ate","eat","tea"]]

*9.Given two strings word1 and word2, return the minimum number of operations required to convert word1 to word2.*

*You have the following three operations permitted on a word:*

* *Insert a character*
* *Delete a character*
* *Replace a character*

**Program:**

def minDistance(word1, word2):

m, n = len(word1), len(word2)

dp = [[0] \* (n + 1) for \_ in range(m + 1)]

# Initialize first column and first row

for i in range(m + 1):

dp[i][0] = i

for j in range(n + 1):

dp[0][j] = j

# Fill dp table

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

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

if word1[i-1] == word2[j-1]:

dp[i][j] = dp[i-1][j-1] # No operation needed

else:

dp[i][j] = 1 + min(dp[i-1][j], # Delete

dp[i][j-1], # Insert

dp[i-1][j-1]) # Replace

return dp[m][n]

# Test cases

test\_cases = [

("horse", "ros"),

("intention", "execution"),

("sunday", "saturday"),

("cat", "cut"),

("girl", "grill"),

]

# Applying function to each test case

results = [minDistance(tc[0], tc[1]) for tc in test\_cases]

# Displaying results for each test case

for i, result in enumerate(results, start=1):

print(f"Test Case {i}: Input: word1 = '{test\_cases[i-1][0]}', word2 = '{test\_cases[i-1][1]}'")

print(f"Output: {result}\n")

* **Input:** word1 = "horse", word2 = "ros"

**Output:** 3

* .**Input:** word1 = "intention", word2 = "execution"

**Output:** 5

*10. Scramble String. We can scramble a string s to get a string t using the following algorithm:*

*If the length of the string is 1, stop.*

*If the length of the string is > 1, do the following:*

*Split the string into two non-empty substrings at a random index, i.e., if the string is s, divide it to x and y where s = x + y.*

*Randomly decide to swap the two substrings or to keep them in the same order. i.e., after this step, s may become s = x + y or s = y + x.*

*Apply step 1 recursively on each of the two substrings x and y.*

*Given two strings s1 and s2 of the same length, return true if s2 is a scrambled string of s1, otherwise, return false.*

**Program:**

def is\_scramble(s1, s2):

if len(s1) != len(s2) or sorted(s1) != sorted(s2):

return False

if s1 == s2:

return True

n = len(s1)

for i in range(1, n):

if (is\_scramble(s1[:i], s2[:i]) and is\_scramble(s1[i:], s2[i:])) or \

(is\_scramble(s1[:i], s2[-i:]) and is\_scramble(s1[i:], s2[:-i])):

return True

return False

# Get input from the user

s1 = input("Enter the first string: ")

s2 = input("Enter the second string: ")

# Check if s2 is a scrambled string of s1

result = is\_scramble(s1, s2)

# Print the result

if result:

print("s2 is a scrambled string of s1.")

else:

print("s2 is not a scrambled string of s1.")

**Input:** s1 = "great", s2 = "eatgr"

**Output:** true

*11. Given a string s which represents an expression, evaluate this expression and return its value.*

*The integer division should truncate toward zero.*

*You may assume that the given expression is always valid. All intermediate results will be in the range of [-231, 231 - 1].*

*·       s consists of integers and operators ('+', '-', '\*', '/') separated by some number of spaces.*

*·       s represents a valid expression.*

*·       All the integers in the expression are non-negative integers in the range [0, 231 - 1].*

*·       The answer is guaranteed to fit in a 32-bit integer*.

**Program:**

def evaluate\_expression(s):

stack = []

num = 0

sign = '+'

operators = {'+', '-', '\*', '/'}

for i, char in enumerate(s):

if char.isdigit():

num = num \* 10 + int(char)

if char in operators or i == len(s) - 1:

if sign == '+':

stack.append(num)

elif sign == '-':

stack.append(-num)

elif sign == '\*':

stack[-1] \*= num

elif sign == '/':

stack[-1] = int(stack[-1] / num) # Integer division truncates towards zero

num = 0

sign = char

return sum(stack)

# Get input from the user

expression = input("Enter the expression: ")

# Remove extra spaces and evaluate the expression

result = evaluate\_expression(expression.replace(" ", ""))

# Print the result

print("Result:", result)

**Input:** s = "3+2\*2"

**Output:** 7

**Input:** s = " 3/2 "

**Output:** 1

*12. Magic Dictionary:*

*Design a data structure that is initialized with a list of different words. Provided a string, you should determine if you can change exactly one character in this string to match any word in the data structure.*

*Implement the MagicDictionary class:*

*MagicDictionary() Initializes the object.*

*void buildDict(String[] dictionary) Sets the data structure with an array of distinct strings dictionary.*

*bool search(String searchWord) Returns true if you can change exactly one character in searchWord to match any string in the data structure, otherwise returns false.*

**Program:**

class MagicDictionary:

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

self.words = set()

def buildDict(self, dictionary):

self.words = set(dictionary)

def search(self, searchWord):

for word in self.words:

if len(word) != len(searchWord):

continue

diff\_count = sum(1 for a, b in zip(word, searchWord) if a != b)

if diff\_count == 1:

return True

return False

# Test the MagicDictionary class

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

obj = MagicDictionary()

obj.buildDict(["hello", "hallo", "world", "word", "test"])

print(obj.search("hello")) # Output: False

print(obj.search("hallo")) # Output: True

print(obj.search("hell")) # Output: True

print(obj.search("helloo")) # Output: True

print(obj.search("hella")) # Output: False

*13(a). Write a program to print all the Non-Prime numbers between A and B?*

**Program:**

def print\_non\_prime\_numbers(A, B):

for num in range(A + 1, B): # Start from A+1 to exclude A and go up to B-1 to exclude B

if num > 1: # Prime numbers are greater than 1

for i in range(2, int(num \*\* 0.5) + 1): # Check for factors other than 1 and the number itself

if num % i == 0:

print(num, end=', ')

break

print() # Just to add a newline after printing all numbers

**Sample Input:** A = 12 B = 19

**Sample Output:** 14, 15, 16, 18

*13(b). Find the year of the given Anniversary is leap year or not. If leap year then print the*

*next Anniversary, if not leap year then print the previous Anniversary*

**Program:**

def is\_leap\_year(year):

# Leap year rules

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

def print\_anniversary\_date(date):

day, month, year = map(int, date.split('/'))

leap\_year = is\_leap\_year(year)

if leap\_year:

print(f"Given Anniversary Year: Leap Year. Anniversary Date: {day:02d}/{month:02d}/{year + 1}")

else:

print(f"Given Anniversary Year: Non Leap Year. Anniversary Date: {day:02d}/{month:02d}/{year - 1}")

**Sample Input:**

Enter Date: 04/11/1947

**Sample Output:**

Given Anniversary Year: Non Leap Year. Anniversary Date: 04/11/1946

*14. Program to find row, column and diagonal sum in Matrix*

**Program:**

def matrix\_sum(a):

n = len(a) # Assuming it's a square matrix

diagonal\_sum = 0

for i in range(n):

row\_sum = sum(a[i])

column\_sum = sum(a[j][i] for j in range(n))

diagonal\_sum += a[i][i]

print(f"Sum of {i+1} row: {row\_sum}")

print(f"Sum of {i+1} column: {column\_sum}")

print(f"Diagonal sum: {diagonal\_sum}")

a = [[1, 2, 3],

[4, 5, 6],

[7, 8, 9]]

**Input&Output:**

Sum of 1 row: 6

Sum of 2 row: 15

Sum of 3 row: 24

Sum of 1 column: 12

Sum of 2 column: 15

Sum of 3 column: 18

Diagonal sum 15

*16(a). Write a program to find the number of special characters in the given statement*

**Program:**

def count\_special\_characters(statement):

special\_characters="!@#$%^&\*()\_-=+[]{}\|:;',.<>?/"

count=0

for char in statement:

if char in special\_characters:

count+=1

return count

given\_statement="modi Birthday @ spetember 17,#&amp;$% is the wishes code for him."

result=count\_special\_characters(given\_statement)

print(f"the no.of special cahracters in the statement is:{result}")

**Sample Input:**

Given statement: Modi Birthday @ September 17, #&amp;$% is the wishes code for him.

**Sample Output**:

Number of special Characters: 5

*16(b). Write a program to print the number of vowels and number of consonants in the given*

*statement and which is maximum?*

**Program:**

def count\_vowels\_consonants(statement):

vowels="aeiouAEIOU"

consonants="bcdfghjklmnpqrstvwxyzBCDFGHJKLMNPQRSTVWXYZ"

num\_vowels=sum(1 for char in statement if char in vowels)

num\_consonants=sum(1 for char in statement if char in consonants)

return num\_vowels,num\_consonants

def find\_maximum(count):

max\_count=max(count)

max\_index=count.index(max\_count)

if max\_index==0:

return "vowels"

else:

return "consonants"

statement=input("entre the given stemt:")

vowels\_count, consonants\_count=count\_vowels\_consonants(statement)

maximum\_type=find\_maximum([vowels\_count,consonants\_count])

print(f"no.of vowels:{vowels\_count}")

print(f"no.of consonants:{consonants\_count}")

print(f"the maximum is in:{maximum\_type}")

**Sample Input:**

Saveetha School of Engineering

**Sample Output**:

Number of vowels = 12 Number of Consonants = 15

*17. Find the Mean, Median and Mode of the array of numbers?*

**Program:**

from statistics import mean,median,mode

array\_of\_elements={12,22,36,12,2}

mean\_value=mean(array\_of\_elements)

median\_value=median(array\_of\_elements)

mode\_value=mode(array\_of\_elements)

print(f"mean:{mean\_value}")

print(f"median:{median\_value}")

print(f"mode:{mode\_value}")

**Sample Input:**

Array of elements = {16, 18, 27, 16, 23, 21, 19}

**Sample Output:**

Mean = 20

Median = 19

Mode = 16

*18(a). Python Program to create a list of all numbers in a range which are perfect squares*

*and the sum of the digits of the number is less than 10.*

**Program:**

def is\_perfect\_square(num):

square\_root = int(num\*\*0.5)

return square\_root\*\*2 == num

def sum\_of\_digits\_less\_than\_10(num):

digit\_sum = sum(int(digit) for digit in str(num))

return digit\_sum < 10

def find\_perfect\_squares\_with\_sum\_less\_than\_10(start, end):

result = [num for num in range(start, end + 1) if is\_perfect\_square(num) and sum\_of\_digits\_less\_than\_10(num)]

return result

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

m = int(input("entre the value of m:"))

perfect\_squares = find\_perfect\_squares\_with\_sum\_less\_than\_10(n, m)

print("Perfect squares with sum of digits less than 10:", perfect\_squares)

**Sample Input &amp; Output:**

Enter lower range: 1

*18(b). Python Program to Find the Nth Largest Number in a List*

**Program:**

def nth\_largest\_number(nums, n):

unique\_nums = list(set(nums))

unique\_nums.sort(reverse=True)

if 1 <= n <= len(unique\_nums):

return unique\_nums[n - 1]

else:

return "N is out of range"

number\_list = [14, 67, 48, 23, 5, 62]

N = 1

result = nth\_largest\_number(number\_list, N)

print(f"The {N}th largest number in the list is: {result}")

**Sample Input:**

List : {14, 67, 48, 23, 5, 62}

N = 4

**Sample Output**:

4th Largest number: 23

*19. Python Program to Create a List of Tuples with the First Element as the Number and Second Element as the Square of the Number.*

**Program:**

def create\_list\_of\_tuples(n):

# Create a list of tuples using a list comprehension

return [(i, i \*\* 2) for i in range(1, n + 1)]

# Test the function

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

result = create\_list\_of\_tuples(n)

print("List of tuples with the first element as the number and the second element as the square:")

print(result)

**Sample Input:**

Enter the lower range:45 Enter the upper range:49

**Sample Output:**

[(45, 2025), (46, 2116), (47, 2209), (48, 2304), (49, 2401)]

*20. You have n jobs and m workers. You are given three arrays: difficulty, profit, and worker*

*where:*

*For example, if three workers attempt the same job that pays $1, then the total profit will be*

*$3. If a worker cannot complete any job, their profit is $0.*

*Return the maximum profit we can achieve after assigning the workers to the jobs.*

**Program:**

def max\_profit(difficulty, profit, worker):

# Create a list of tuples (difficulty, profit) sorted by difficulty

jobs = sorted(zip(difficulty, profit))

# Sort workers by their skill level

worker.sort(reverse=True)

max\_profit = 0

i = 0

for skill in worker:

while i < len(jobs) and skill < jobs[i][0]:

i += 1

if i < len(jobs):

max\_profit += jobs[i][1]

return max\_profit

# Test the function

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

difficulty = [2, 4, 6, 8, 10]

profit = [10, 20, 30, 40, 50]

worker = [4, 5, 6, 7]

result = max\_profit(difficulty, profit, worker)

print("Maximum profit:", result)

**Input:** difficulty = [2,4,6,8,10], profit = [10,20,30,40,50], worker = [4,5,6,7]

**Output:** 100