Python Assignments"

1.Find the Second Largest Element in a List.

• Question: Write a function to find the second largest element in a list of numbers. • Input: A list of numbers. • Output: The second largest number. • Requirements: Handle exceptions for lists with fewer than two elements.

In [7]:
```
list=[2,3,4,2,4,6,77,8,45,23,3,3,44,5,34,55,7,3,5,3]
list.sort()
print(list)
print(list[-2])
```

```
[2, 2, 3, 3, 3, 3, 3, 4, 4, 5, 5, 6, 7, 8, 23, 34, 44, 45, 55, 77]
55
```

In [21]:
```
def second_largest(nums):
    try:
        #Ensure that list has atleat two distinct elements
        if len(set(nums))<2:
            raise ValueError("List must contain atleast two distinct elements.")
        #Remove duplicates and sort the list in descending order
        unique_nums=sorted(set(nums),reverse=True)
        #Return the second largest number in the sorted list.
        return unique_nums[1]
    except ValueError as e:
        return str(e)


list=[2,3,4,5,67,7,88,24,32,2,2]
result=second_largest(list)
print("second largest element is :",result)
```

```
second largest element is : 67
```

In [39]:
```
def second_largest(nums):
    try:
        if len(set(nums))<2:
            raise ValueError("List must contain at atleast two distinct values")
        first=second =None #initialise Both as None
        for num in nums:
            if first is None or num > first:
                second=first #Update second largest to the previous largest
                first=num    #Update largest to the current numbe
            elif second is None or first>num>second:
                second=num # Update second largest if it's smaller than first but larger than second
        return second
    except ValueError as e:
        return str(s)

list=[2,3,4,5,67,7,88,24,32,2,2]
result=second_largest(list)
print("second largest element is :",result)
```

```
second largest element is : 67
```

# 2. Merge Two Lists Alternately

• Question: Write a function that merges two lists by taking elements alternately from each list. • Input: Two lists of equal length. • Output: A merged list with alternating elements. • Requirements: Handle exceptions if the lists are of different lengths.

In [55]:
```
List1=[1,2,44,5,5,2,4,2,4,2,4]
List2=[3,2,4,5,2,4,2,4,24,2,4]
List3=[]
for i in range(len(List1)):
    List3.append(List1[i])
    List3.append(List2[i])

List3
```

Out[55]:
```
[1, 3, 2, 2, 44, 4, 5, 5, 5, 2, 2, 4, 4, 2, 2, 4, 4, 24, 2, 2, 4, 4]
```

In [63]:
```
def Merged_list(List1,List2):
    try:
        if len(List1)!=len(List2):
            raise ValueError("Enter the same length of element in two lists")
        Merged_list=[]
        for i in range(len(List1)):
            Merged_list.append(List1[i])
            Merged_list.append(List2[i])
        return Merged_list
    except ValueError as e:
        return str(e)
Merged_list(List1,List2)
List1=[1,2,44,5,5,2,4,2,4,2,4]
List2=[3,2,4,5,2,4,2,4,24,2,4]
```

Out[63]:
```
[1, 3, 2, 2, 44, 4, 5, 5, 5, 2, 2, 4, 4, 2, 2, 4, 4, 24, 2, 2, 4, 4]
```

In [65]:
```
List1.extend(List2)
print(List1)
```

```
[1, 2, 44, 5, 5, 2, 4, 2, 4, 2, 4, 3, 2, 4, 5, 2, 4, 2, 4, 24, 2, 4]
```

It will append the values at end of the list1 but not in alternatively.

3. Check if All Characters are Unique

• Question: Write a function to check if all characters in a given string are unique. • Input: A string. • Output: "Unique" or "Not Unique". • Requirements: Handle exceptions for non-string inputs.

In [112…
```
string="vasu"
char_set=set()
is_unique=True
for i in string:
    if i in char_set:
        print("Not Unique")
        is_unique =False
        break
    char_set.add(i)
else:
    print("Unique")
```

```
Unique
```

In [94]:
```
def are_characters_unique(input_string):
    try:
        # Ensure the input is a string
        if not isinstance(input_string, str):
            raise TypeError("Input must be a string.")

        # Use a set to track characters
        char_set = set()

        # Iterate over each character in the string
        for char in input_string:
            if char in char_set:
                return "Not Unique"
            char_set.add(char)

        return "Unique"

    except TypeError as e:
        return str(e)

# Example usage:
input_str = "hello"
result = are_characters_unique(input_str)
print(result)  # Output: Not Unique
```

```
Not Unique
```

4. Group Anagrams Together

• Question: Write a function to group anagrams from a list of strings. • Input: A list of strings. • Output: A list of lists where each sublist contains anagrams. • Requirements: Handle exceptions for non-string elements.

In [ ]:
```
"What is anagram --> anagram is Phrase of two words should have same letters and same lengtth."
```

In [8]:
```
def anagram(str1,str2):
    if sorted(str1)==sorted(str2):
        print("anagram")
    else:
        print("Not a Anagram")
```

```
anagram("listen","silent")
```

```
anagram
```

In [ ]:
```
"Problem"
```

In [20]:
```
def group_anagrams(words):
    try:
        #check if all elements are strings
        if not all(isinstance(word,str) for word in words):
            raise TypeError("All Elements in the list must be strings")
        anagram_groups={}
        for word in words:
            #Sort the word to create a key and group anagrams
            sorted_word=''.join(sorted(word))
            #Manually check if the sorted_word key exists and append to the group
            if sorted_word in anagram_groups:
                anagram_groups[sorted_word].append(word)
            else:
                anagram_groups[sorted_word]=[word]
        return list(anagram_groups.values())
    except TypeError as e:
        return str(e)

inpu_list=["eat","ate","tan","tea","nat","bat"]
result=group_anagrams(inpu_list)
print(result)
```

```
[['eat', 'ate', 'tea'], ['tan', 'nat'], ['bat']]
```

In [14]:
```
word="vasu"
sorted_words=''.join(sorted(word))
sorted_words
```

Out[14]:
```
'asuv'
```

5. Count the Frequency of Elements in a List

• Question: Write a function to count the frequency of each element in a list. • Input: A list of elements. • Output: A dictionary where keys are elements and values are their frequencies. • Requirements: Handle exceptions for invalid inputs.

In [28]:
```
List=["a","s","e","e","r","f","f","e","e","r","v","f","e","e","e","r"]

dict={}
for i in List:
    if i in dict:
        dict[i]+=1
    else:
        dict[i]=1
print(dict)
```

```
{'a': 1, 's': 1, 'e': 7, 'r': 3, 'f': 3, 'v': 1}
```

In [32]:
```
def count_frequencies(elements):
    try:
        if not isinstance(elements,list):
            raise TypeError("Input Must be a list.")
        #Dictionary to store the frequency of each elements
        frequency_dict={}
        for elem in elements:
            if elem in frequency_dict:
                frequency_dict[elem]+=1
            else:
                frequency_dict[elem]=1
        return frequency_dict
    except TypeError as e:
        return str(e)

count_frequencies(List)
```

Out[32]:
```
{'a': 1, 's': 1, 'e': 7, 'r': 3, 'f': 3, 'v': 1}
```

6. Find the Missing Number in a Sequence

• Question: Write a function to find the missing number in a sequence of numbers from 1 to n. • Input: A list of numbers with one missing element. • Output: The missing number. • Requirements: Handle exceptions for lists with no missing elements.

In [ ]:
```
sequence means if  the elements should be -->1,2,3,4,5,6,7 like that if any value missed in that sequence we sh
```

In [76]:
```
List=[1,2,3,5]
n=len(List)+1
sum1=n*(n+1)//2
sum2=sum(List)
missing_number=sum1-sum2
print(missing_number)
```

```
4
```

In [82]:
```
def find_missing_number(nums):
    try:
        if not all(isinstance(num,int) for num in nums):
            raise TypeError("All the elements should be integer values")
        #calculated the expected sum of numbers from 1 to n
        n=len(nums)+1
        expected_sum=n*(n+1)//2
        #calculated the actual sum of the numbers in the list
        actual_sum=sum(nums)
        #find the missing number
        missing_number=expected_sum-actual_sum
        #raise an exception if no number is missing (missing_numbershould be grater then 0)
        if missing_number<=0:
            raise ValueError("There is no missing number in the sequence")
        return missing_number
    except (TypeError,ValueError) as e:
        return str(e)

input_list=[1,2,4,5]
result=find_missing_number(input_list)
print(result)
```

```
3
```

7. Sum of Elements in a Tuple

• Question: Write a function to calculate the sum of all elements in a tuple. • Input: A tuple of numbers. • Output: The sum of the elements. • Requirements: Handle exceptions for non-numeric elements.

In [3]:
```
tuple=(2,32,4,2,4,2,4,2,2,4,3,44,43,55)

sum=0
for i in tuple:
    sum+=i
print("sum of tuple elements:",sum)
```

```
sum of tuple elements: 203
```

In [25]:
```
def sum_tuple(tuple):
    try:
        if not all(isinstance(i,int) for i in tuple):
            raise TypeError("Enter the integers values into the tuple")
        #Calculated the sum
        sum_tuple_elements=0
        for num in tuple:
            sum_tuple_elements+=num
        return sum_tuple_elements
    except (TypeError,ValueError) as e:
        return int(e)

tuple=(22,3,2,2,4,2,4,5,66,7,88,8,3,45,66)
result=sum_tuple(tuple)
print("sum of tuple elements:",result)
```

```
sum of tuple elements: 327
```

8. Find the Longest Word in a Sentence

• Question: Write a function to find the longest word in a sentence. • Input: A sentence. • Output: The longest word in the sentence. • Requirements: Handle exceptions for non-string inputs.

In [35]:
```
string="Hi hello mr I ma Vasu and I am Student in the St.Mary's Group of institutions Guntur"
Longest_word=[]
for word in string.split(" "):
    if len(word)>len(Longest_word):
        Longest_word=word


print("Longest_word in the senetence:",Longest_word)
```

```
Longest_word in the senetence: institutions
```

In [43]:
```python
def find_longest_word(sentence):
    try:
        if not isinstance(sentence,str):
            raise ValueError("Input must be a string")
        #split the sentence into words
        words=sentence.split()
        #Find the largest word,defualt to an empty string if no words found
        longest_word=max(words,key=len,default="")
        return longest_word
    except ValueError as e:
        return str(e)

sentence="Hi hello Namasthey Vanakkam Adhab ,Nenu Mee Vasu "
result=find_longest_word(sentence)
print("Longest_word is:",result)
```

```
Longest_word is: Namasthey
```

9. Find Intersection of Two Lists

• Question: Write a function to find the intersection of two lists. • Input: Two lists. • Output: A list containing the common elements. • Requirements: Handle exceptions for invalid inputs.

In [89]:
```python
list1=[12,23,42,5,3,34,4,55,3,24]
list2=[23,323,323,2,3,2,3,2,3,2]
list3=[]
for i in list1:
    if i in list2 and i not in list3:
        list3.append(i)


print(list3)
```

```
[23, 3]
```

In [65]:
```python
def find_intersection(list1,list2):
    try:
        #Check if both inputs are lists
        if not isinstance(list1,list) or not isinstance(list2,list):
            raise ValueError("Both inputs must be a lists")
        #Find the intersection  using set intersection
        intersection=list(set(list1) & set(list2))
        return intersection
    except TypeError as e:
        return list(e)
list1=[12,23,42,5,3,34,4,55,3,24]
list2=[23,323,323,2,3,2,3,2,3,2]
find_intersection(list1,list2)
```

Out[65]:
```
[3, 23]
```

10. Check for Substring

• Question: Write a function to check if one string is a substring of another. • Input: Two strings. • Output: "Substring" or "Not Substring".

In [103…
```python
string1="Srinivasulu"
string2="vasu"
if string2 in string1:
    print("Substring")
else:
    print("Not Substring")
```

```
Substring
```

In [109…
```python
def check_substring(string1,string2):
    try:
        if not isinstance(string,str) or not isinstance(substring,str):
            raise ValueError("Two string values must be a string")
        if substring in string:
            print("Substring")
        else:
            print("Not Substring")
    except TypeError as e:
        return str(e)

string1="Maheshbabu"
string2="Mahesh"
```

```
check_substring(string1,string2)
```

```
Substring
```

11. String Palindrome Check

• Question: Write a function to check if a given string is a palindrome. • Input: A string. • Output: "Palindrome" or "Not Palindrome". • Requirements: Handle exceptions for non-string inputs.

In [111…

```
string="malayalam"

if string==string[::-1]:
    print("Palindrom")
else:
    print("Not Palindrom")
```

```
Palindrom
```

In [129…

```
def check_palindrom(string):
    try:
        if not isinstance(string,str):
            raise TypeError("Enter the string value")
        if string==string[::-1]:
            print("Palindrom")
        else:
            print("Not Palindrom")
    except ValueError as e:
        return str(e)

string="madam"
check_palindrom(string)
```

```
Palindrom
```

In [123…

```
def check_palindrom(string):
    try:
        if not isinstance(string,str):
            raise TypeError("Enter the string value")
        # Remove spaces and convert to lowercase for case-insensitive comparison
        cleaned_string = string.replace(" ", "").lower()
        if cleaned_string==cleaned_string[::-1]:
            print("Palindrom")
        else:
            print("Not Palindrom")
    except ValueError as e:
        return str(e)

string="sri rs"
check_palindrom(string)
```

```
Palindrom
```

12. Temperature Conversion

• Question: Write a function to convert temperature between Celsius and Fahrenheit. • Input: A temperature and the unit (C/F). • Output: The converted temperature. • Requirements: Handle invalid unit inputs and non-numeric temperature values.

In [13]:

```
def convert_temparature(value,unit):
    try:
        #check if the value is a numer
        if not isinstance(value,(int,float)):
            raise ValueError("Temparature must be a number")
        #Convert the temparature based on the unit
        if unit.upper()=="C":
            #convert Celcius to Fahrenheit
            converted =(value*9/5)+32
            return f"{converted:.2f}°F"
        elif unit.upper()=="F":
            #Convert Fahrenheit to Celius
            converted=(value-32)*5/9
            return f"{converted:.2f}°C"
        else:
            #Handld invalid unit input
            raise ValueError("Unir=t must be 'C' for Celcius or 'F' for Fahsrenheit")
    except ValueError as e:
        return str(e)

print(convert_temparature(25.24,"C"))
print(convert_temparature(253,"F"))
```

```
77.43°F
122.78°C
```

13. Simple Calculator

• Question: Write a function-based calculator to perform basic arithmetic operations (+, -, *, /). • Input: Two numbers and an operator. • Output: The result of the operation. • Requirements: Handle division by zero and invalid operator exceptions.

In [61]:
```
def Calculator(num1,num2,operator):
    try:
        if not isinstance(num1,int) or  not isinstance(num2,int):
            raise TypeError("Elements must be numbers")
        if operator=="+":
            Addition=num1+num2
            return Addition

        elif operator=="-":
            Substraction=num1-num2
            return Substraction
        elif operator=="*":
            Multiplication=num1*num2
            return Multiplication
        elif operator=="/":
            if num2==0:
                raise ZeroDivisionError("cannot divide by zero")
            Division=num1/num2
            return Division
        else:
            raise ValueError("Invalid opearator .Use +,-,*,/")
    except (ValueError,ZeroDivisionError) as e:
        return str(e)


print(Calculator(12,23,"+"))
print(Calculator(12,23,"-")  )
print(Calculator(12,23,"*")  )
print(Calculator(12,23,"/")  )
print(Calculator(12,0,"/")  )
print(Calculator(12,23,"**")  )
```

```
35
-11
276
0.5217391304347826
cannot divide by zero
Invalid opearator .Use +,-,*,/
```

14. Reverse a String

• Question: Write a function to reverse a given string. • Input: A string. • Output: The reversed string. • Requirements: Handle exceptions for non-string inputs.

In [63]:
```
string="srinivasulu"

print(string[::-1])
```

```
ulusavinirs
```

In [71]:
```
def reverse_string(string):
    try:
        if not isinstance(string,str):
            raise ValueError("Input must be a string")
        reversed_str=string[::-1]
        return reversed_str
    except ValueError as e:
        return str(e)

print(reverse_string("srinu"))
print(reverse_string(2224))
```

```
unirs
Input must be a string
```

15. Count Vowels in a String

• Question: Write a function to count the number of vowels in a given string. • Input: A string. • Output: The number of vowels. • Requirements: Handle exceptions for non-string inputs.

In [75]:
```
string="srinivasulu"
count=0
for i in string:
    if i in "aeiouAEIOU":
        count+=1

print(count)
```
```
5
```

In [77]:
```
def count_vowels(string):
    try:
        if not isinstance(string,str):
            raise ValueError("Input must be string or char")
        vowels="aeiouAEUOI"
        vowel_count=sum(1 for char in string if char in vowels)
        return vowel_count
    except ValueError as e:
        return str(e)

count_vowels("ravikumarreddy")
```
Out[77]:
```
5
```

16. Check Armstrong Number

• Question: Write a function to check if a number is an Armstrong number. • Input: An integer. • Output: "Armstrong" or "Not Armstrong". • Requirements: Handle exceptions for non-integer inputs.

In [ ]:
```
What is an Armstrong number?
An Armstrong number (also called a narcissistic number) for a given number of digits is an integer such that th
1^3+5^3_3^3=153
```

In [95]:
```
def check_armstrong(num):
    try:
        if not isinstance(num,int):
            raise ValueError("Input must be integer")
        num_str=str(num)
        len_digits=len(num_str)
        sum=0
        for digit in num_str:
            total =int(digit)**len_digits
            sum+=total

          # return total
        if sum==num:
            print("ArmStrong")
        else:
            print("Not a ArmStrong")
    except ValueError as e:
        return str(e)

print(check_armstrong(153))
print(check_armstrong(234))
```
```
ArmStrong
None
Not a ArmStrong
None
```

17. Generate Multiplication Table

• Question: Write a function to generate and print the multiplication table for a given number. • Input: An integer. • Output: The multiplication table for the number. • Requirements: Handle exceptions for non-integer inputs.

In [109…
```
num=3

for i in range(1,11):
    mul=f"{num} X {i} = {num*i}"
    print(mul)
```

```
3 X 1 = 3
3 X 2 = 6
3 X 3 = 9
3 X 4 = 12
3 X 5 = 15
3 X 6 = 18
3 X 7 = 21
3 X 8 = 24
3 X 9 = 27
3 X 10 = 30
```

In [113…
```python
def generate_multiplication_table(number):
    try:
        if not isinstance(number,int):
            raise ValueError("Input must be integer")
        for i in range(1,11):
            mul_table=f"{number} X {i} = {number*i}"
            print(mul_table)
    except ValueError as e:
        return str(e)

generate_multiplication_table(10)
```

```
10 X 1 = 10
10 X 2 = 20
10 X 3 = 30
10 X 4 = 40
10 X 5 = 50
10 X 6 = 60
10 X 7 = 70
10 X 8 = 80
10 X 9 = 90
10 X 10 = 100
```

18. Greatest Common Divisor (GCD)

• Question: Write a function to find the GCD of two numbers. • Input: Two integers. • Output: The GCD of the two numbers. • Requirements: Handle exceptions for non-integer inputs.

In [ ]:
```
The Greatest Common Divisor (GCD) of two integers is the largest positive integer that divides both numbers witl

For example:

The GCD of 12 and 8 is 4 because 4 is the largest number that divides both 12 and 8 exactly.
The GCD of 27 and 18 is 9 because 9 is the largest number that divides both 27 and 18.
```

In [119…
```python
import math

def find_gcd(num1,num2):
    try:
        #check if both inputs are integers
        if not isinstance(num1,int) or not isinstance(num2,int):
            raise ValueError("Input must be integrs")
        return math.gcd(num1,num2)
    except ValueError as e:
        return str(e)

find_gcd(24,12)
```

Out[119… 12

In [185…
```python
def find_gcd(num1,num2):
    try:
        if not isinstance(num1,int) or not isinstance(num2,int):
            raise ValueError("Input must be a integr values")
        while num2!=0:
            num1,num2=num2,num1%num2
        return num1
    except ValueError as e:
        return str(e)

find_gcd(12,32)
```

Out[185… 4

19. Convert List to Set

• Question: Write a function that converts a list to a set, removing duplicate elements. • Input: A list. • Output: A set with unique elements. • Requirements: Handle exceptions if the input is not a list.

In [189…
```
List=[2,3,4,5,67,7,77,53,3,4,55,4,3,3,4,5]
set=set(List)
print(set)
```
```
{2, 67, 3, 4, 5, 7, 77, 53, 55}
```

In [1]:
```
def list_to_set(input_lst):
    try:
        if not isinstance(input_lst,list):
            raise ValueError("Input values must be integrs or list")

        set_values=set(input_lst)

        return set_values
    except ValueError as e:
        return str(e)

list_to_set(([2,3,4,5,67,8,8,9,4,4,33,31,456,6,3,778]))
```
Out[1]:
```
{2, 3, 4, 5, 6, 8, 9, 31, 33, 67, 456, 778}
```

20. Remove Vowels from String

• Question: Write a function to remove all vowels from a given string. • Input: A string. • Output: The string without vowels. • Requirements: Handle exceptions for non-string inputs.

In [31]:
```
string="srinivasulu"
string1=""
for i in string:
    if i in "aeiouAEIOU":
        continue
    else:
        string1+=i

string1
```
Out[31]:
```
'srnvsl'
```

In [39]:
```
def del_vowels(string):
    try:
        if not isinstance(string,str):
            raise ValueError("Input must be string")
        new_string=""
        for i in string:
            if i in "aeiuoAEIOU":
                del i
            else:
                new_string+=i
        return new_string
    except ValueError as e:
        return str(e)

del_vowels("ravikumarvadde")
```
Out[39]:
```
'rvkmrvdd'
```

21. Merge Two Dictionaries

• Question: Write a function to merge two dictionaries into one. • Input: Two dictionaries. • Output: A merged dictionary. • Requirements: Handle exceptions for invalid inputs.

In [57]:
```
dict1={"name":"vasu","age":21}
dict2={"Gender":"Male","study":"Btech"}

dict1.update(dict2)

print(dict1.copy())

print(dict1)
```
```
{'name': 'vasu', 'age': 21, 'Gender': 'Male', 'study': 'Btech'}
{'name': 'vasu', 'age': 21, 'Gender': 'Male', 'study': 'Btech'}
```

In [59]:
```
def merged_dict(dict1,dict2):
    try:
        if not isinstance(dict1,dict) or not isinstance(dict2,dict):
            raise ValueError("Input values must be dictionaries")
        merged_dict={**dict1,**dict2}
        return merged_dict
    except ValuError as e:
        return str(e)

merged_dict(dict1,dict2)
```

Out[59]:
```
{'name': 'vasu', 'age': 21, 'Gender': 'Male', 'study': 'Btech'}
```

22. Find Maximum Element in List

• Question: Write a function to find the maximum element in a list. • Input: A list of numbers. • Output: The maximum number. • Requirements: Handle exceptions for empty lists or non-numeric elements.

In [69]:
```
list=[2,2,4,4,5,6,77,31,88,2,8,8]
max_num=list[0]
for i in list:
    if i> max_num:
        max_num=i

print(max_num)
```

```
88
```

In [1]:
```
def find_max_element(lst):
    try:
        if not isinstance(lst,list):
            raise ValueError("Input must be List")
        if not lst:
            raise ValueError("The list is empty")
        if  not all(isinstance(i,(int, float)) for i in lst):
            raise ValueError("Values must be either integer or float")

        max_num=lst[0]
        for i in lst:
            if i> max_num:
                max_num=i
        return max_num
    except ValueError as e:
        return str(e)

find_max_element([2,2,4,4,5,6,77,31,88,2,8,8])
```

Out[1]:
```
88
```

23. Calculate Simple Interest

• Question: Write a function to calculate simple interest given the principal, rate of interest, and time. • Input: Three numbers representing principal, rate, and time. • Output: The calculated simple interest. • Requirements: Handle exceptions for invalid inputs.

In [8]:
```
def simple_interest(principle,rate,time):
    try:
        if not isinstance(principle,(int,float)) or not isinstance(rate,(int,float)) or not isinstance(time,(in
            raise ValueError("Input values must integers or float")
        simple_interest=(principle*rate*time)/100
        return simple_interest
    except ValueError as e:
        return str(e)

simple_interest(12000,23,24)
```

Out[8]:
```
66240.0
```

24. Count Words in a Sentence

• Question: Write a function to count the number of words in a given sentence. • Input: A sentence. • Output: The word count. • Requirements: Handle exceptions for non-string inputs.

In [35]:
```
def count_words(sentence):
    try:
        if not isinstance(sentence,str):
            raise ValuError("Input must be a string values")
        word_count=0
        for word in sentence.split(" "):
            word_count+=1
        return word_count
    except ValueError as e:
        return str(e)
sentence="I am srnin"
count_words(sentence)
```

Out[35]: 3

In [37]:
```
def count_words(sentence):
    try:
        # Check if the input is a string
        if not isinstance(sentence, str):
            raise ValueError("Input must be a string.")

        # Split the sentence into words based on whitespace
        words = sentence.split()

        # Return the number of words
        return len(words)

    except ValueError as ve:
        return str(ve)

# Example usage
print(count_words("Hello world! This is a test."))  # Output: 7
print(count_words("   Leading and trailing spaces   "))  # Output: 5
print(count_words(12345))  # Output: "Input must be a string."
```

```
6
4
Input must be a string.
```

25. Check for Anagram

• Question: Write a function to check if two strings are anagrams of each other. • Input: Two strings. • Output: "Anagram" or "Not Anagram". • Requirements: Handle exceptions for non-string inputs.

In [18]:
```
def check_anagram(str1,str2):
    try:
        if not isinstance(str1,str) or not isinstance(str2,str):
            raise ValueError("Input values must be in string format")
        str1_cleaned=str1.replace(" ","").lower()
        str2_cleaned=str2.replace(" ","").lower()
        if sorted(str1_cleaned)== sorted(str2_cleaned):
            return "Anagram"
        else:
            return "Not Anagram"
    except ValueError as e:
        return str(e)

print(check_anagram("listen", "silent"))        # Output: "Anagram"
print(check_anagram("hello", "world"))          # Output: "Not Anagram"
print(check_anagram("Dormitory", "Dirty Room"))  # Output: "Anagram"
print(check_anagram("abc", 123))
```

```
Anagram
Not Anagram
Anagram
Input values must be in string format
```

26. Count Occurrences of a Character

• Question: Write a function to count the number of times a specific character appears in a string. • Input: A string and a character. • Output: The count of occurrences. • Requirements: Handle exceptions for non-string inputs.

In [27]:
```
string="srinivasulu"
dict={}

if "v" in dict:
    dict["v"]+=1
else:

    dict["v"]=1

dict
```

Out[27]: {'v': 1}

In [33]:
```python
def count_char(string,char):
    try:
        if not isinstance(string,str) or not isinstance(char,str) or len(char)!=1:
            raise ValueError("Input must be a str and length of char must be greater then equal 1")
        count_char=string.count(char)
        return count_char
    except ValueError as e:
        return str(e)

count_char("srinivaulu","i")
```

Out[33]:
```
2
```

27. Flatten a Nested List

• Question: Write a function to flatten a nested list into a single list. • Input: A nested list. • Output: A flat list. • Requirements: Handle exceptions for non-list inputs.

In [48]:
```python
def flatten_nested_list(nested_list):
    try:
        # Check if the input is a list
        if not isinstance(nested_list, list):
            raise ValueError("Input must be a list.")

        # Initialize an empty list to hold the flattened elements
        flat_list = []

        # Define a helper function to recursively flatten the list
        def flatten(item):
            if isinstance(item, list):
                for sub_item in item:
                    flatten(sub_item)  # Recursively flatten sub-items
            else:
                flat_list.append(item)  # Append non-list items to flat_list

        # Start the flattening process
        flatten(nested_list)

        return flat_list

    except ValueError as ve:
        return str(ve)

# Example usage
print(flatten_nested_list([1, [2, 3], [4, [5, 6]], 7]))  # Output: [1, 2, 3, 4, 5, 6, 7]
print(flatten_nested_list([[1, 2], [3, [4, 5]], 6]))     # Output: [1, 2, 3, 4, 5, 6]
print(flatten_nested_list("not a list"))                  # Output: "Input must be a list."
```

```
[1, 2, 3, 4, 5, 6, 7]
[1, 2, 3, 4, 5, 6]
Input must be a list.
```

28. Find the Unique Elements in a List

• Question: Write a function to find all unique elements in a list, excluding duplicates. • Input: A list of elements. • Output: A list of unique elements. • Requirements: Handle exceptions for invalid inputs (e.g., non-list inputs).

In [54]:
```python
List=[2,2,3,4,4,3,5]
list=[]
for i in List:
    if i not in list:
        list.append(i)
list
```

Out[54]:
```
[2, 3, 4, 5]
```

In [1]:
```python
def find_unique_elements(input_list):
    try:
        # Check if the input is a list
        if not isinstance(input_list, list):
            raise ValueError("Input must be a list.")

        # Initialize an empty list to hold unique elements
        unique_elements = []

        # Iterate through the list and add unique elements
        for item in input_list:
            if item not in unique_elements:
                unique_elements.append(item)

        return unique_elements

    except ValueError as ve:
        return str(ve)

# Example usage
print(find_unique_elements([1, 2, 2, 3, 4, 4, 5]))  # Output: [1, 2, 3, 4, 5]
print(find_unique_elements(['apple', 'banana', 'apple', 'orange']))  # Output: ['apple', 'banana', 'orange']
print(find_unique_elements("not a list"))  # Output: "Input must be a list."
```

```
[1, 2, 3, 4, 5]
['apple', 'banana', 'orange']
Input must be a list.
```

29. Convert a String to Title Case

• Question: Write a function to convert a given string to title case (capitalize the first letter of each word). • Input: A string. • Output: The string in title case. • Requirements: Handle exceptions for non-string inputs.

In [5]:
```
string="vasu"
string.title()
str=""
```

Out[5]:
```
'Vasu'
```

In [1]:
```
def convert_to_title_case(input_string):
    try:
        # Check if the input is a string
        if not isinstance(input_string, str):
            raise ValueError("Input must be a string.")

        # Convert the string to title case
        title_case_string = input_string.title()

        return title_case_string

    except ValueError as ve:
        return str(ve)

# Example usage
print(convert_to_title_case("hello world"))                        # Output: "Hello World"
print(convert_to_title_case("python programming language"))    # Output: "Python Programming Language"
print(convert_to_title_case(12345))                                 # Output: "Input must be a string."
```

```
Hello World
Python Programming Language
Input must be a string.
```

30. Calculate Compound Interest

• Question: Write a function to calculate compound interest given the principal, rate of interest, time, and number of times interest is compounded per year. • Input: Four numbers representing principal, rate, time, and the number of times interest is compounded per year. • Output: The calculated compound interest. • Requirements: Handle exceptions for invalid inputs (e.g., negative numbers).

In [3]:
```
def calculate_compound_interest(principal, rate, time, n):
    try:
        # Check if all inputs are positive numbers
        if not all(isinstance(x, (int, float)) and x >= 0 for x in [principal, rate, time, n]):
            raise ValueError("All inputs must be non-negative numbers.")

        # Convert the rate from percentage to decimal
        rate_decimal = rate / 100

        # Calculate the compound interest
        amount = principal * (1 + rate_decimal / n) ** (n * time)
        compound_interest = amount - principal

        return compound_interest

    except ValueError as ve:
        return str(ve)

# Example usage
print(calculate_compound_interest(1000, 5, 10, 12))  # Output: 647.609
print(calculate_compound_interest(1500, 4.5, 6, 4))   # Output: 407.59
print(calculate_compound_interest(-1000, 5, 10, 12))  # Output: "All inputs must be non-negative numbers."
```

```
647.0094976902801
461.9868396123495
All inputs must be non-negative numbers.
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js