(1) The function should:

-- Create a list of three numbers. -- Create a tuple with the same three numbers.

-- Modify the list by changing the second element. -- Try to modify the tuple’s second element.

-- If it fails, catch the error and return the error message. -- Return both the modified list and the tuple error message.

In [17]:
```python
def list_vs_tuple_demo():
    list_num = [10,13,17]
    tuple_num = (10,13,17)

 # Modify list
    list_num[1] = 56

 # Try modifying tuple
    try:
        tuple_num[1] = 56
        tuple_error = "No error"  # This will never happen
    except TypeError as e:
        tuple_error = str(e)

    return {"modified_list":list_num,
            "tuple_error":tuple_error}

list_vs_tuple_demo()
```

Out[17]:
```
{'modified_list': [10, 56, 17],
 'tuple_error': "'tuple' object does not support item assignment"}
```

(2) Write a function:

-- Takes an integer score (0–100).

Returns:"A" if score ≥ 90 "B" if score ≥ 80 "C" if score ≥ 70 "D" if score ≥ 60 "F" otherwise

-- If the score is outside 0–100, return "Invalid score".

In [24]:
```python
def grade_classifier(score):
    if score < 0 or score > 100:
        return "Invalid score"

    if score >= 90:
        return "A"
    elif score >= 80:
        return "B"
    elif score >= 70:
        return "C"
    elif score >= 60:
        return "D"
    else:
        return "F"

grade_classifier(300)
```

Out[24]:
```
'Invalid score'
```

(3) Requirements:

-- The function takes an integer n.

-- Use a for loop to calculate the sum of all even numbers from 1 to n (inclusive).

-- Use a while loop to count how many even numbers are in that range.

-- Return a dictionary:

In [28]:
```
def integer(n):
    even_sum = 0
    for i in range(1 , n+1):
        if i % 2 == 0:
            even_sum += i

    count = 0
    current_number = 1
    while current_number <= n:
        if current_number % 2 == 0:
            count += 1
        current_number += 1

    return {"sum_of_evens": even_sum, "count_of_evens":count}

integer(10)
```

Out[28]:
```
{'sum_of_evens': 30, 'count_of_evens': 5}
```

(4) Functions + Error Handling

-- Requirements: The function takes two numbers: a and b.

Return the result of a / b.

If division by zero occurs, return the string: "Error: division by zero"

If either a or b is not a number, return: "Error: invalid input"

No printing — only return.

In [42]:
```
def divide(a, b):
    try:
        return a / b
    except ZeroDivisionError:
        return "Error: division by zero"
    except TypeError:
        return "Error: invalid input"

divide("x",3)
```

Out[42]:
```
'Error: invalid input'
```

(5) Lists & List Comprehension

-- Requirements:

The function receives a list of integers nums.

Return a new list containing the squares of only the even numbers.

You must solve it using list comprehension.

If the input is not a list, return "Invalid input".

In [48]:
```
def squares(nums):
    if not isinstance(nums, list):
        return "Invalid input"

    return [x*x for x in nums if x % 2 == 0]
```

In [49]:
```
squares([1, 2, 3, 4, 5])
```

Out[49]:
```
[4, 16]
```

In [51]:
```
squares("abc")

### isinstance(object, type) checks whether a variable is of a certain type.
```

Out[51]:
```
'Invalid input'
```

(6) Dictionaries

-- Requirements:

Input: a string text. and Convert the string to lowercase.

Split it into words (split by spaces). Then Count how many times each word appears.

Return a dictionary where:key = word value = count

Ignore empty strings (e.g., multiple spaces).

Do NOT print anything — only return.

Example: word_frequency("Hello hello world") {"hello": 2, "world": 1}

In [58]:
```
   def word_frequency(text):
    text = text.lower()
    words = text.split()

    word_counts = {}

    for word in words:
        if word.strip() == "":
            continue
        if word in word_counts:
            word_counts[word] += 1
        else:
            word_counts[word] = 1

    return word_counts
```

In [59]:
```
word_frequency("Hello hello world")
```

Out[59]:
```
{'hello': 2, 'world': 1}
```

(7) Sets (Data Cleaning, Uniqueness, Membership)

-- Requirements:

Input: a list of integers nums. Remove duplicates using a set.

Return the unique values sorted in ascending order. If nums is not a list, return "Invalid input"

In [12]:
```
def unique_sorted(nums):
    if not isinstance(nums, list):
        return "Invalid input"

    values = set(nums)
    return sorted(values)
```

In [13]:
```
unique_sorted([4, 1, 2, 4, 3, 2])
```

Out[13]:
```
[1, 2, 3, 4]
```

(8) Strings

-- Requirements:

Input: a string text.

Remove:leading & trailing spaces extra spaces between words (convert multiple spaces → one space)

Convert the entire string to lowercase.

Remove punctuation characters:. , ! ? : ;

Return the cleaned string. If the input is not a string, return "Invalid input".

```
In [26]: import re

         def clean_text(text):
             if not isinstance(text, str):
                 return "Invalid input"

             # lowercase
             text = text.lower()

             # remove punctuation: . , ! ? : ;
             text = re.sub(r'[.,!?:;]', '', text)

             # remove extra spaces
             text = " ".join(text.split())

             return text
```

```
In [27]: clean_text("  Hello,,,   World!!  ")
```

```
Out[27]: 'hello world'
```

(9) Lambda, Map, Filter

-- Requirements:

Input: a list of integers nums.

Use filter() + lambda to keep only positive numbers.

Use map() + lambda to square those positive numbers.

Convert the final result to a list and return it.

If input is not a list, return "Invalid input".

```
In [32]: def process_numbers(nums):
             if not isinstance(nums, list):
                 return "Invalid input"

             nums = list(filter(lambda x : x > 0 , nums))
             nums = list(map(lambda x: x * x, nums))

             return nums
```

```
In [33]: process_numbers([-1, 0, 2, 3, -5])
```

```
Out[33]: [4, 9]
```

```
In [34]: def process_numbers(nums):
             if not isinstance(nums, list):
                 return "Invalid input"

             return list(map(lambda x: x*x, filter(lambda x: x > 0, nums)))
```

```
In [35]: process_numbers([-1, 0, 2, 3, -5])
```

```
Out[35]: [4, 9]
```

(10)List of Dictionaries

-- Requirements:

records is a list of dictionaries where each dict has:{"name": "...", "score": number}

Return a new list containing only the dictionaries where score >= threshold.

If records is not a list of dicts, return "Invalid input".

No printing — only return.

```
In [40]: def filter_records(records, threshold):
             if not isinstance(records, list):
                 return "Invalid Output"
             for item in records:
                 if not isinstance(item, dict):
                     return "Invalid Output"

             result = [r for r in records if r.get("score", -1) >= threshold]
             return result
```

```
In [41]: filter_records(
             [{"name": "A", "score": 50},
              {"name": "B", "score": 85},
              {"name": "C", "score": 90}],
             80)
```

```
Out[41]: [{'name': 'B', 'score': 85}, {'name': 'C', 'score': 90}]
```

(11) File Handling

-- Requirements:

The function receives a filename (string).

Open the file in read mode. Read all lines.

Remove: newline characters (\n) leading/trailing spaces empty lines

Return a list of cleaned lines.

Use try/except to handle errors:

If the file does not exist → return "File not found"

If filename is not a string → return "Invalid input"

In [5]:
```
def read_clean_file(filename):
    if not isinstance(filename, str):
        return "Invalid input"

    try:
        with open(filename, "r") as f:
            cleaned_lines = []

            for line in f:
                cleaned = line.strip()
                if cleaned:
                    cleaned_lines.append(cleaned)

            return cleaned_lines

    except FileNotFoundError:
        return "File not found"
    except Exception:
        return "Error reading file"
```

(12) List Comprehension + Conditional Logic

-- Requirements:

Given a list of integers nums, return a new list using list comprehension with the following rules:

If a number is even, store its square. If a number is odd, store its cube.

Ignore any item that is not an integer.

If nums is not a list → return "Invalid input".

In [16]:
```
def transform_list(nums):
    if not isinstance(nums, list):
        return "Invalid input"

    return [(x**2 if x % 2 == 0 else x**3)
            for x in nums
           if isinstance(x, int)]
```

In [17]:
```
transform_list([1, 2, 3, 4])
```

```
Out[17]: [1, 4, 27, 16]
```

(13) Generators

-- Requirements:

Use yield (not return).

Generate even numbers from 0 to n (inclusive).

If n is not an integer → return "Invalid input".

```
In [30]: def generate_even(n):
             if not isinstance(n, int):
                 return "Invalid input"

             num = 0
             while num <= n:
                 if num % 2 == 0:
                     yield num
                 num += 1
```

```
In [31]: gen = generate_even(10)
         list(gen)
```

```
Out[31]: [0, 2, 4, 6, 8, 10]
```

(14)Dictionary Comprehension

-- Requirements:

Given a list of integers nums, return a dictionary comprehension where:

Key = the number Value = its square

Additional rules: Ignore non-integer items.

If nums is not a list → return "Invalid input".

Use dictionary comprehension only, not loops.

```
In [40]: def square_dict(nums):
             if not isinstance(nums , list):
                 return "Invalid Output"

             return {x:x**2 for x in nums if isinstance(x,int)}
```

```
In [41]: square_dict([1, 2, 3])
```

```
Out[41]: {1: 1, 2: 4, 3: 9}
```

(15) Write a function multiples_list(n, m):

Input: integers n and m

Output: a list of first n multiples of m

In [10]:
```
def multiples_list(n,m):
    if not isinstance(n, int):
        return "Invalid Output"

    return [(m*i) for i in range(1,n+1)]
```

In [11]:
```
multiples_list(5, 3)
```

Out[11]:
```
[3, 6, 9, 12, 15]
```

(16) Function + Dictionary + Loops

Write a function char_count(s):

Input: a string s

Output: a dictionary with character counts

In [23]:
```
def char_count(n):
    if not isinstance(n,str):
        return "Invalid Output"

    cha_counts = {}

    for text in n:
        cha_counts[text] = cha_counts.get(text,0)+1

    return cha_counts
```

In [24]:
```
char_count("data")
```

Out[24]:
```
{'d': 1, 'a': 2, 't': 1}
```

(17) Write a function filter_primes(nums):

Input: a list of integers nums

Output: a list containing only the prime numbers

In [29]:
```
import math

def is_prime(num):
    if not isinstance(num, int):
        return False
    if num <= 1:
        return False
    if num == 2:
        return True
    if num % 2 == 0:
        return False

    for i in range(3, int(math.sqrt(num)) + 1, 2):
        if num % i == 0:
            return False
    return True

def filter_primes(nums):
    if not isinstance(nums, list):
        return "Invalid input"

    prime_numbers = []
    for number in nums:
        if is_prime(number):
            prime_numbers.append(number)
    return prime_numbers
```

In [30]:
```
filter_primes([1, 2, 3, 4, 5, 6, 7])
```

Out[30]:
```
[2, 3, 5, 7]
```

LIST OPERATIONS

(1) Write a function that removes duplicates from a list while keeping the original order.

In [8]:
```
def remove_duplicates(nums):
    if not isinstance(nums,list):
        return "INVALID OUTPUT"

    seen = set()
    result = []

    for x in nums:
        if x not in seen:
            seen.add(x)
            result.append(x)

    return result
```

In [9]:
```
remove_duplicates([1,12,2,13,13])
```

Out[9]:
```
[1, 12, 2, 13]
```

(2) Write a function that returns a sorted version of the list both: ascending descending

{"asc": [...], "desc": [...]}

```
In [21]: def sorted_values(nums):
             if not isinstance(nums,list):
                 return "Invalid Output"


             ascending = sorted(nums)
             descending = sorted(nums,reverse=True)
             return {"asc":ascending ,"des":descending}
```

```
In [22]: sorted_values([4, 1, 7, 3])
```

```
Out[22]: {'asc': [1, 3, 4, 7], 'des': [7, 4, 3, 1]}
```

(3)Write a function that takes a list and returns a dictionary of item counts.

```
In [23]: def count(s):
             if not isinstance(s,list):
                 return "INVALID OUTPUT"

             counts = {}

             for x in s:
                 counts[x] = counts.get(x,0)+1

             return counts
```

```
In [25]: count(["a","b","a"])
```

```
Out[25]: {'a': 2, 'b': 1}
```

(4)Write a function that filters values greater than a given threshold.

```
In [32]: def filter(nums,threshold):
             if not isinstance(nums,list):
                 return "Invalid Output"

             for x in nums:
                 if not isinstance(x,int):
                     return "Invalid Output"

             return [s for s in nums if s >=threshold]
```

```
In [33]: filter([3,10,7],5)
```

```
Out[33]: [10, 7]
```

or

```
In [34]: def filter(nums, threshold):
             if not isinstance(nums, list):
                 return "Invalid Output"

             if not isinstance(threshold, int):
```

```
        return "Invalid Output"

    for x in nums:
        if not isinstance(x, int):
            return "Invalid Output"

    return [s for s in nums if s >= threshold]
```