Part 2: Write Small Functions / Programs

Instructions

Complete the following questions.

[P2 Q1] Find all duplicates in a list

Write a function `find_duplicates()` that takes in a list of integers from 1 to *n*, where *n* is the length of the list. The function returns the first value that is repeated in the list.

If the parameter is not a list or the elements in the list are outside the range, return `-1`. If no elements are repeated, then return `0`.

```
print(find_duplicates([1, 2, 3, 2]))    # 2
print(find_duplicates([1, 1, 2, 2]))    # 1
print(find_duplicates([4, 3, 1, 2]))    # 0
print(find_duplicates([1, 2, 2, 0]))    # -1
print(find_duplicates([]))              # 0
print(find_duplicates((1, 2, 3)))       # -1
```

RESTRICTIONS

Only allowed: variables, `if` statements, `while` loops. Functions: `len()`, `int()`, `float()`, `type()`, `isinstance()`. String methods: `format()`. List operators and methods: create `[ ]`, `list()`, `append()`. Keywords: `elif`, `else`, `return`, `break`, `continue`, `def`, `self`, `None`, `try`, `raise`, `except`, `finally`, `is`, `import sys`. Any exception handling.

Remember: Do NOT use `for` loops or the `in` keyword. Do NOT use `min()` / `max()`, `enumerate()` / `zip()` or `sorted()`. Do NOT use negative indexing (e.g. `ls[-1]`) or slicing (e.g. `[1:]`). Do NOT use *other* list operations and methods. Marks will be deducted.

[P2 Q2] No Repeats!

Write a program which continuously asks for user input until the a repeated value is given. The program should then print out `Input number <number> is not unique!` and terminate.

```
$ python3 no_repeats.py
Give me an input: # a
Give me an input: # b
Give me an input: # 375
Give me an input: # c#
Give me an input: # b

Input number 5 is not unique!
```

RESTRICTIONS

Only allowed: variables, `if` statements, `while` loops. Functions: `len()`, `int()`, `float()`, `type()`, `isinstance()`, `input()`, `print()`. String methods: `format()`. List operators and methods: create `[ ]`, `list()`, `append()`. Keywords: `elif`, `else`, `return`, `break`, `continue`, `def`, `None`, `try`, `raise`, `except`, `finally`, `is`, `import sys`. Any exception handling.

Remember: Do NOT use `for` loops or the `in` keyword. Do NOT use `min()` / `max()`, `enumerate()` / `zip()` or `sorted()`. Do NOT use negative indexing (e.g. `ls[-1]`) or slicing (e.g. `[1:]`). Do NOT use *other* list operations and methods. Marks will be deducted.

[P2 Q3] Adding up letters

Write a function `word_sum()` that takes in a string and returns the sum according to the following rules:

- Each letter in the alphabet is given a number corresponding to their position in the alphabet, i.e. `'a'` has value `1`, `'b'` has value `2`, `'c'` has value `3`, ... `'z'` has value `26`. Letters should be case insensitive.
- Numeric digits have the same value as their true numeric counterparts, i.e. `'1'` has value `1`, `'2'` has value `2`, etc.
- Characters which are not a letter in the alphabet and not a digit have a value of `0`.

The sum to be returned should be an integer equal to sum of all the corresponding values of the characters. If the input is not of the correct type, raise a `TypeError` with the message `Input must be a string.`

```
print(word_sum('abc'))       # 6
print(word_sum('ab*c!'))     # 6
print(word_sum('123ab'))     # 9
print(word_sum('!@#0'))      # 0
print(word_sum(''))          # 0
print(word_sum(123))         # TypeError
```

RESTRICTIONS

Only allowed: variables, `if` statements, `while` loops. Functions: `len()`, `int()`, `float()`, `type()`, `isinstance()`. String methods: `format()`. List operators and methods: create `[ ]`, `list()`, `append()`. Keywords: `elif`, `else`, `return`, `break`, `continue`, `def`, `self`, `None`, `try`, `raise`, `except`, `finally`, `is`, `import sys`. Any exception handling.

Remember: Do NOT use `for` loops or the `in` keyword. Do NOT use *other* string or list operations and methods. This includes, but is not limited to `isalpha()`, `isnumeric()`, `isalnum()` and `isdigit()`. Do NOT use `min()` / `max()`, `enumerate()` / `zip()` or `sorted()`. Do NOT use negative indexing (e.g. `ls[-1]`) or slicing (e.g. `[1:]`). Marks will be deducted.

[P2 Q4] Longest consecutive sequence

A consecutive sequence is one where the next number is 1 more than the previous one (e.g. 3, 4, 5, 6, etc.). Write a function `longest_consecutive_sequence()` that takes as input a list of integers and returns the starting index of the first longest consecutive sequence.

If the list value is not an integer, ignore it. If no consecutive sequence of two or greater exists or if the given input is not a list, return `-1`.

```
print(longest_consecutive_sequence([7, 1, 2, 3, 6, 9, 2, 5, 6, 7])) # 1
print(longest_consecutive_sequence([8, 4, 3, 1, 5, 6]))             # 4
print(longest_consecutive_sequence([9, 10, 11, 13, 14, 15, 16]))    # 3
print(longest_consecutive_sequence([2, "a", 3, 4]))                 # 2
print(longest_consecutive_sequence([1, 1, 1, 1, 1]))                # -1
print(longest_consecutive_sequence([0]))                            # -1
```

RESTRICTIONS

Only allowed: variables, `if` statements, `while` loops. Functions: `len()`, `int()`, `float()`, `type()`, `isinstance()`. String methods: `format()`. List operators and methods: create `[ ]`, `list()`, `append()`. Keywords: `elif`, `else`, `return`, `break`, `continue`, `def`, `self`, `None`, `try`, `raise`, `except`, `finally`, `is`, `import sys`. Any exception handling.

Remember: Do NOT use `for` loops or the `in` keyword. Do NOT use `min()` / `max()`, `enumerate()` / `zip()` or `sorted()`. Do NOT use negative indexing (e.g. `ls[-1]`) or slicing (e.g. `[1:]`). Do NOT use *other* list operations and methods. Marks will be deducted.

[P2 Q5] Testing Pour

Write **four** good unit tests for the `pour_into()` method from the Filling in Bottles question, without repeating the same style of test case. Give the inputs and expected outputs by filling in the corresponding fields in the text file template. Also, give your reasoning for each test case in the justification section.

SCAFFOLD

```
TEST 1

Justification:

Inputs:
```

```
    Bottle 1 capacity:
    Bottle 1 current amount:
    Bottle 2 capacity:
    Bottle 2 current amount:
    Amount (input argument):

Expected output:
    Bottle 1 current amount:
    Bottle 2 current amount:


TEST 2

Justification:

Inputs:
    Bottle 1 capacity:
    Bottle 1 current amount:
    Bottle 2 capacity:
    Bottle 2 current amount:
    Amount (input argument):

Expected output:
    Bottle 1 current amount:
    Bottle 2 current amount:


TEST 3

Justification:

Inputs:
    Bottle 1 capacity:
    Bottle 1 current amount:
    Bottle 2 capacity:
    Bottle 2 current amount:
    Amount (input argument):

Expected output:
    Bottle 1 current amount:
    Bottle 2 current amount:


TEST 4

Justification:

Inputs:
    Bottle 1 capacity:
    Bottle 1 current amount:
    Bottle 2 capacity:
    Bottle 2 current amount:
    Amount (input argument):

Expected output:
    Bottle 1 current amount:
    Bottle 2 current amount:
```