Python Exercices

– La deuxième séance

A.Belcaid **[**24/09/2024**]**

Exercise: banish Function in Python

Write a function named banish that accepts two lists of integers (a1 and a2) as parameters and removes all occurrences of the values in a2 from a1. An element is "removed" by shifting all subsequent elements one index to the left to cover it up, placing a 0 into the last index. The original relative ordering of a1's elements should be retained.

Example:

a1 = [42, 3, 9, 42, 42, 0, 42, 9, 42, 42, 17, 8, 2222, 4, 9, 0, 1]

a2 = [42, 2222, 9]

banish(a1, a2)

After calling the function, the contents of a1 should become:

[3, 0, 17, 8, 4, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

Explanation:

1. Function Definition: Start by defining the banish function that takes two lists as input.

2. Checking Values: Iterate through each element in a1, and if an element is found in a2, it should be removed.

3. Shifting Elements: When an element is removed, shift all subsequent elements to the left.

4. Adding Zeros: Insert zeros at the end to compensate for the removed elements.

Lessons Learned: Understanding how to use loops to manipulate data.

The importance of maintaining the order of elements while modifying lists.

How to handle empty data and avoid modifying lists when conditions are not met.

Source Code:

```python
def banish(a1, a2):
    # Convert a2 to a set for faster lookup
    banish_set = set(a2)
    n = len(a1)  # Number of elements in a1
    index = 0  # Index for the next element to keep in a1

    for i in range(n):
        if a1[i] not in banish_set:
            a1[index] = a1[i]  # Place the element in the new i
            index += 1  # Increment the index

    # Fill the remaining positions with zero using while loop
    while index < n:
        a1[index] = 0
        index += 1

# Test the function
a1 = [42, 3, 9, 42, 42, 0, 42, 9, 42, 42, 17, 8, 2222, 4, 9, 0,
a2 = [42, 2222, 9]
banish(a1, a2)
print(a1)
```

Exercise: collapse_pairs Function in Python

Write a function named collapse_pairs that accepts a list of integers as a parameter and modifies the list so that each pair of neighbouring integers (such as the pair at indexes 0-1, and the pair at indexes 2-3, etc.) are combined into a single sum of that pair. The sum is stored at the even index if the sum is even and at the odd index if the sum is odd. The other index of the pair is set to 0.

Example:

Suppose the following list is declared and the following call is made:

a = [7, 2, 8, 9, 4, 22, 7, 1, 9, 10]

collapse_pairs(a)

After calling the function, the contents of a should become: [0, 9, 0, 17, 26, 0, 8, 0, 0, 19] Explanation:

1. Function Definition: Start by defining the collapse_pairs function that takes a single list of integers as input.

2. Processing Pairs of Elements: The function should iterate through the list, two elements at a time, processing each pair (like index 0-1, 2-3, etc.).

3. Storing the Sum: For each pair of integers, sum the two values.

If the sum is even, store it at the even index (0, 2, 4, ...).

If the sum is odd, store it at the odd index (1, 3, 5, ...).

4. Set the Other Index to Zero: Once the sum is stored, set the other index (the one not storing the sum) to 0.

5. Handling Odd Length Lists: If the list has an odd number of elements, leave the last element as it is.

Source Code:

```
def collapse_pairs(a):
    # Iterate through the list in steps of 2 to process pairs
    for i in range(0, len(a) - 1, 2):
        pair_sum = a[i] + a[i + 1]  # Sum the pair

        if pair_sum % 2 == 0:  # If the sum is even
            a[i] = pair_sum
            a[i + 1] = 0
        else:  # If the sum is odd
            a[i + 1] = pair_sum
            a[i] = 0

# Test the function
a = [7, 2, 8, 9, 4, 22, 7, 1, 9, 10]
collapse_pairs(a)
print(a)  # Output: [0, 9, 0, 17, 26, 0, 8, 0, 0, 19]

```

Exercise: find_median Function in Python

Write a function named find_median that accepts a list of integers and returns the median of the numbers in the list. The median is the number that appears in the middle when the elements are arranged in order. Assume that the list is of odd size, and the numbers range between 0 and 99 inclusive.

Example:

Example 1

numbers = [5, 2, 4, 17, 55, 4, 3, 26, 18, 2, 17]

find_median(numbers) # Output: 5

Example 2

numbers = [42, 37, 1, 97, 1, 2, 7, 42, 3, 25, 89, 15, 10, 29, 27]

find_median(numbers) # Output: 25

Explanation:

1. Function Definition: Begin by defining the find_median function that takes a list of integers as input.

2. Sorting the List: The median requires sorting the list of numbers in non-decreasing order. Once the list is sorted, the middle element is the median.

3. Identifying the Median: Since the list size is always odd, the median is the element located at the index len(list) // 2 after sorting.

4. Returning the Result: The function should return the number at the middle index.

Source Code:

```python
def find_median(numbers):
    # Sort the list of numbers
    numbers.sort()

    # Find the middle index
    middle_index = len(numbers) // 2

    # Return the median (the number at the middle index)
    return numbers[middle_index]

# Test the function with example inputs
numbers1 = [5, 2, 4, 17, 55, 4, 3, 26, 18, 2, 17]
print(find_median(numbers1))  # Output: 5

numbers2 = [42, 37, 1, 97, 1, 2, 7, 42, 3, 25, 89, 15, 10, 29, 27]
print(find_median(numbers2))  # Output: 25

```

Exercise: flip_half Function in Python

Write a function named flip_half that reverses the order of the elements in odd-numbered positions of a list of integers. The function should only modify the odd-numbered elements (positions 1, 3, 5, etc.), leaving the elements in even-numbered positions unchanged.

Example:

Original list:

index: 0 1 2 3 4 5 6 7

[1, 8, 7, 2, 9, 18, 12, 0]

Modified list after calling flip_half:

index: 0 1 2 3 4 5 6 7

[1, 0, 7, 18, 9, 2, 12, 8]

Notice that the numbers in odd positions (1, 3, 5, 7) have been reversed, but the numbers in even positions (0, 2, 4, 6) remain the same.

Explanation:

1. Function Definition: Start by defining the flip_half function that takes a list as input.

2. Identifying Odd Positions: The odd-numbered positions in the list are indexed as 1, 3, 5, 7, etc. You need to extract the elements at these positions.

3. Reversing the Odd-Positioned Elements: Reverse the order of the elements at odd positions.

4. Inserting the Reversed Elements: Replace the original odd-positioned elements with the reversed ones.

Source Code:

```
1   def flip_half(numbers):
2       # Extract the odd-positioned elements
3       odd_elements = numbers[1::2]  # Slicing from index 1, stepping by 2
4
5       # Reverse the odd-positioned elements
6       odd_elements.reverse()
7
8       # Place the reversed elements back into their original positions
9       numbers[1::2] = odd_elements
10
11  # Test the function with example input
12  numbers = [1, 8, 7, 2, 9, 18, 12, 0]
13  flip_half(numbers)
14  print(numbers)  # Output: [1, 0, 7, 18, 9, 2, 12, 8]
15
```

Conclusion:

Through these four exercises, we explored key Python programming concepts such as list manipulation, filtering, reversing sequences, and modifying list elements based on specific conditions. These tasks helped reinforce our understanding of loops, list slicing, and efficient data handling, which are essential skills in writing effective and optimised code.