Functions Problem Set

Part 1

Table of contents

1 Problem 1: Even numbers

Write a Python function `filter_even_numbers` that takes a list of numbers and returns a new list containing only the even numbers.

- Example Input: `[1, 2, 3, 4]`
- Example Output: `[2, 4]`

1.1 Solution:

```
def filter_even_numbers(numbers):
  even_numbers = [] # create an empty list for the even numbers
  for num in numbers:
    if num % 2 == 0: # if the number is divisable by 2 then it is even
      even_numbers.append (num) # if so, add it to the list of even numbers
  return even_numbers

filter_even_numbers([1, 2, 3, 4])
```

```
[2, 4]
```

1.2 Explanation for the Solution:

The task is to filter out only the even numbers from a list and return them in a new list. The function `filter_even_numbers` uses a simple loop to achieve this.

1.2.1 Step-by-Step Breakdown:

1. **Initialization:**
   - An empty list `even_numbers` is created to store the even numbers found in the input list `numbers`.
2. **Loop through the List:**
   - For each number `num` in the input list, the function checks if the number is divisible by 2 using `num % 2 == 0`. If the condition is true, the number is even.
   - If the number is even, it is appended to the `even_numbers` list.
3. **Return the Result:**
   - After processing all numbers in the input list, the function returns the `even_numbers` list, which contains only the even numbers.

1.3 Solution #2:

```
def filter_even_numbers_v2(numbers):
  return [num for num in numbers if num % 2 == 0]

filter_even_numbers_v2([1, 2, 3, 4])
```

```
[2, 4]
```

1.4 Explanation for Solution #2:

This version of the solution uses Python's list comprehension feature to make the function more concise and efficient.

1.4.1 Step-by-Step Breakdown:

1. **List Comprehension:**
   - The expression `[num for num in numbers if num % 2 == 0]` creates a new list by iterating over each number `num` in the input list `numbers`.
   - For each number, it checks if the number is divisible by 2 (`num % 2 == 0`), which means the number is even.
   - Only even numbers are added to the new list.

2. **Return the Result:**
    - The list comprehension generates the list of even numbers, which is immediately returned by the function.

2 Problem 2: Second largest

Write a Python function `find_second_largest` that takes a list of positive numbers and returns the second largest number in the list *without* sorting the list.

- Example Input: `[2, 5, 7, 1, 8, 3, 9]`
- Example Output: `8`

2.1 Solution:

```
def find_second_largest(numbers):
  largest = -1
  second_largest = -1
  for num in numbers:
    if num > largest:
      second_largest = largest
      largest = num
    elif num > second_largest:
      second_largest = num
  return second_largest

find_second_largest([2, 5, 7, 1, 8, 3, 9])
```

```
8
```

2.2 Explanation for the Solution:

The task is to find the second largest number in a list of positive numbers without sorting the list. The function `find_second_largest` solves this by scanning the list once and keeping track of both the largest and second-largest numbers.

2.2.1 Step-by-Step Breakdown:

1. **Initialization:**
   - Two variables `largest` and `second_largest` are initialized to `-1`. This ensures that any number in the list will replace these values since the list contains only positive numbers.
2. **Loop through the List:**
   - The function iterates through the list of numbers.
   - For each number `num`:
     - If `num` is greater than `largest`, update `second_largest` to the value of `largest`, then update `largest` to `num`. This ensures that the previous largest number becomes the second largest.
     - Otherwise, if `num` is greater than `second_largest` but less than `largest`, update `second_largest` to `num`.
3. **Return the Result:**
   - After the loop finishes, `second_largest` will hold the second largest number in the list.

2.2.2 Example Walkthrough:

For the input `[2, 5, 7, 1, 8, 3, 9]`:

- After processing:
  - `2` becomes the largest, second largest remains `-1`.
  - `5` becomes the largest, `2` becomes the second largest.
  - `7` becomes the largest, `5` becomes the second largest.
  - `1` is ignored.
  - `8` becomes the largest, `7` becomes the second largest.
  - `3` is ignored.
  - `9` becomes the largest, `8` becomes the second largest.
- The function returns `8`.

3 Problem 3: Most common letter

Write a Python function `find_most_common_letter` that takes a string and returns the most common letter in the string.

- Example Input: `"Hello, World!"`
- Example Output: `'L'`

3.1 Solution:

```
def find_most_common_letter(string):
  letter_counts = {} # empty dictionary of the characters and their counts
  for letter in string.upper(): # for each character in the string (upper
   ↪  case)
    if letter in letter_counts: # if the char in the dictionary
      letter_counts[letter] += 1 # increase the count by 1
    else:
      letter_counts[letter] = 1 # otherwise, set the count to 1

  # Now let's find the largest count
  max_count = -1 # set the max count to -1
  max_letter = '' # set the letter with the max count to ''
  for letter in letter_counts:
    if letter_counts[letter] > max_count:
      max_count = letter_counts[letter]
      max_letter = letter

  return (max_letter)

find_most_common_letter("Hello, World!")
```

```
'L'
```

3.2 Explanation:

The task is to find the most common letter in a given string. The function `find_most_common_letter` counts the occurrences of each letter, ignoring case, and returns the letter with the highest frequency.

3.2.1 Step-by-Step Breakdown:

1. **Initialization:**

- We initialize an empty dictionary `letter_counts` to store each letter and its corresponding count.

2. **Loop through the string:**
   - Convert the string to uppercase using `string.upper()` to handle case insensitivity.
   - For each character:
     - If the character is already in `letter_counts`, increase its count by 1.
     - If the character is not in the dictionary, add it with an initial count of 1.
3. **Find the most common letter:**
   - Initialize `max_count` to `-1` and `max_letter` to an empty string.
   - Loop through the dictionary to find the letter with the highest count.
4. **Return the result:**
   - The function returns the letter with the maximum count.

3.2.2 Example Walkthrough:

For the input `"Hello, World!"`:

- After converting to uppercase: `"HELLO, WORLD!"`
- The dictionary `letter_counts` will look like: `{'H': 1, 'E': 1, 'L': 3, 'O': 2, 'W': 1, 'R': 1, 'D': 1}`.
- The most common letter is `'L'` with a count of 3.

4 Problem 4: Recursive sum

Write a *recursive* Python function `recursive_sum` that takes a list of integers and returns the sum of all the numbers in the list.

- Example Input: `[1, 2, 3, 4, 5]`
- Example Output: `15`

4.1 Solution:

```python
def recursive_sum(numbers):
  if not numbers: # if the list of numbers is empty
    return 0 # return zero
  else: # otherwise
    # return sum the first num in the list and
    # the sum of the remaining items in the list
    return numbers[0] + recursive_sum(numbers[1:])
recursive_sum([1, 2, 3, 4, 5])
```

```
15
```

4.2 Explanation:

The task is to calculate the sum of a list of integers using recursion. The function `recursive_sum` achieves this by breaking down the problem into smaller subproblems.

4.2.1 Step-by-Step Breakdown:

1. **Base Case:**
   - If the list `numbers` is empty (`not numbers`), the sum is `0`. This is the base case for the recursion, stopping the function when no numbers are left to sum.
2. **Recursive Case:**
   - If the list is not empty, the function takes the first number in the list (`numbers[0]`) and adds it to the result of a recursive call on the rest of the list (`numbers[1:]`).
   - This recursive process continues until the base case is reached (an empty list).
3. **Return the Result:**
   - Each recursive call returns a partial sum, and when the base case is reached, the accumulated sums are returned all the way back up the call stack.

4.2.2 Example Walkthrough:

For the input `[1, 2, 3, 4, 5]`:

- The function first returns `1 + recursive_sum([2, 3, 4, 5])`.
- Then it returns `2 + recursive_sum([3, 4, 5])`, and so on, until it reaches the empty list and returns `0`.
- The final result is `1 + 2 + 3 + 4 + 5 = 15`.

5 Problem 5: Sum of Squares

Write a Python function `sum_of_squares` that takes a list of numbers and returns the sum of their squares.

- Example Input: `[1, 2, 3]`
- Example Output: `14`

5.1 Solution:

```
def sum_of_squares(numbers):
    return sum([num**2 for num in numbers])

sum_of_squares([1, 2, 3])
```

```
14
```

5.2 Explanation:

The task is to calculate the sum of the squares of a list of numbers. The function `sum_of_squares` achieves this using list comprehension and the built-in `sum()` function.

5.2.1 Step-by-Step Breakdown:

1. **List Comprehension:**
   - `[num**2 for num in numbers]` creates a list of the squares of each number in the input list `numbers`. For each `num` in the list, `num**2` computes the square of that number.
2. **Sum of the Squares:**
   - The `sum()` function then adds up all the squared numbers in the list produced by the list comprehension.
3. **Return the Result:**
   - The function returns the sum of the squares of all numbers in the input list.

5.2.2 Example Walkthrough:

For the input `[1, 2, 3]`: - The list comprehension `[num**2 for num in numbers]` produces `[1, 4, 9]`. - The `sum()` function adds these values: `1 + 4 + 9 = 14`.

6 Problem 6: Unique words

Write a Python function `count_unique_words` that takes a list of strings and returns the number of unique words in the list.

- Example Input: `['apple', 'banana', 'apple', 'cherry', 'banana', 'date']`
- Example Output: `4`

6.1 Solution:

```
def count_unique_words(words):
  unique_words = [] # create an empty list to store the unique words
  for word in words: # for each word in the original list of words
    if word not in unique_words: # if the word is not the unique list
      unique_words.append(word) # add that word to the unique list
  return len(unique_words) # return the length of the list of the unique
   ↪ words

count_unique_words(['apple', 'banana', 'apple', 'cherry', 'banana', 'date'])
```

```
4
```

6.2 Explanation for the Solution:

The task is to count the number of unique words in a given list of strings. The function `count_unique_words` uses a simple approach by checking each word and building a list of unique words.

6.2.1 Step-by-Step Breakdown:

1. **Initialization:**
   - We start by creating an empty list `unique_words` that will store the words that appear only once.
2. **Loop through the list of words:**
   - For each word in the input list `words`, check if the word is already in the `unique_words` list.
   - If the word is not in the list, it is added to the `unique_words` list.
3. **Return the result:**
   - After processing all words, the length of the `unique_words` list is returned, which represents the number of unique words.

6.3 Solution #2:

```
def count_unique_words_v2(words):
  return len(set(words)) # return the length of the set version of the list
   ↪  words

count_unique_words_v2(['apple', 'banana', 'apple', 'cherry', 'banana',
 ↪  'date'])
```

```
4
```

6.4 Explanation for Solution #2:

This version of the solution leverages Python's built-in `set` data structure, which automatically removes duplicates, making the process more efficient.

6.4.1 Step-by-Step Breakdown:

1. **Convert List to Set:**
   - The function converts the input list `words` to a set using `set(words)`. Since sets only store unique elements, any duplicate words are automatically removed.
2. **Return the Length of the Set:**

- The length of the set is then calculated using `len()`, which gives the number of unique words.