Functions in Python

Problem Set - Part II

Table of contents

1 Problem 5: Recursive sum 1

2 Problem 6: Unique words 2

3 Problem 7: Higher-Order Functions 3

4 Problem 8: Flatten a Nested List 3

5 Problem 9: Recursive Function to Reverse a List 4

6 Problem 10: Find Missing Numbers 4

1 Problem 5: 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`

- Solution

```
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
```

2 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`

- 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
```

- 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
```

- Solution #3

```
def count_unique_words_v3(words):

  return len({word for word in words}) # Using list comprehension

count_unique_words_v3(['apple', 'banana', 'apple', 'cherry', 'banana', 'date'])
```

```
4
```

3 Problem 7: Higher-Order Functions

Write a function named `apply_operation` that takes a list of numbers, a function that performs an operation on a single number (e.g., square, cube), and applies that operation to each number in the list, returning a new list.

- Example Input: `apply_operation([1, 2, 3, 4], lambda x: x**2)`
- Example Output: `[1, 4, 9, 16]`

- Solution

```
def apply_operation(numbers, operation):
    return [operation(number) for number in numbers]


apply_operation([1, 2, 3, 4], lambda x: x**2)
```

```
[1, 4, 9, 16]
```

4 Problem 8: Flatten a Nested List

Write a recursive Python function `flatten` that takes a nested list (a list containing other lists) and returns a flat list containing all the elements in the nested list, in the same order.

- Example Input: `flatten([1, [2, 3], [4, [5, 6]], 7])`
- Example Output: `[1, 2, 3, 4, 5, 6, 7]`

- Solution

```python
def flatten(nested_list):
    flat_list = []
    for item in nested_list:
        if type(item) == list:
            flat_list.extend(flatten(item))
        else:
            flat_list.append(item)
    return flat_list


flatten([1, [2, 3], [4, [5, 6]], 7])
```

```
[1, 2, 3, 4, 5, 6, 7]
```

5 Problem 9: Recursive Function to Reverse a List

Write a *recursive* function named `reverse_list` that takes a list and returns a new list with the elements in reverse order.

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

- Solution

```python
def reverse_list(lst):
    if len(lst) == 0:
        return []
    else:
        return [lst[-1]] + reverse_list(lst[:-1])

reverse_list([1, 2, 3, 4, 5])
```

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

6 Problem 10: Find Missing Numbers

Given a list of unique integers sorted in increasing order, write a Python function named `find_missing` that returns a list of any missing integers in the sequence from the minimum to the maximum value.

- Example Input: `[1, 2, 4, 6, 7]`
- Example Output: `[3, 5]`

- Solution

```python
def find_missing(numbers):
    full_set = set(range(min(numbers), max(numbers) + 1))
    missing = full_set - set(numbers)
    return sorted(list(missing))

find_missing([1, 2, 4, 6, 7])
```

```
[3, 5]
```