assignment-2

October 28, 2024

[1]:
```python
def count_case(s):
    upper_count = sum(1 for char in s if char.isupper())
    lower_count = sum(1 for char in s if char.islower())
    return upper_count, lower_count

# Sample usage
sample_string = 'The quick Brow Fox'
upper, lower = count_case(sample_string)
print(f'No. of Upper case characters: {upper}')
print(f'No. of Lower case Characters: {lower}')
```

```
No. of Upper case characters: 3
No. of Lower case Characters: 12
```

[2]:
```python
def unique_elements(lst):
    return list(set(lst))

sample_list = [1, 2, 3, 3, 3, 3, 4, 5]
unique_list = unique_elements(sample_list)
print(f'Unique List: {unique_list}')
```

```
Unique List: [1, 2, 3, 4, 5]
```

[13]:
```python
def is_perfect_number(n):
    if n < 1:
        return False
    divisors_sum = sum(i for i in range(1, n) if n % i == 0)
    return divisors_sum == n

number = 6
print(f'{number} is Perfect: {is_perfect_number(number)}')
```

```
6 is Perfect: True
```

[4]:
```python
def sort_hyphenated_words(words):
    sorted_words = sorted(words.split('-'))
    return '-'.join(sorted_words)
```

```
input_words = 'green-red-yellow-black-white'
sorted_sequence = sort_hyphenated_words(input_words)
print(f'Sorted Words: {sorted_sequence}')
```

```
Sorted Words: black-green-red-white-yellow
```

[7]:
```
def positional_arguments(a, b, c):
    return a + b + c

print(positional_arguments(6, 7, 8))
```

```
21
```

[8]:
```
def keyword_arguments(a, b, c):
    return a * b * c

print(keyword_arguments(c=6, a=7, b=8))  # Order doesn't matter
```

```
336
```

[14]:
```
def mixed_arguments(a, b, c=1):
    return a + b + c

print(mixed_arguments(6, 7))        # Using default for c
print(mixed_arguments(6, 7, 8))     # Overriding default for c
```

```
14
21
```

[15]:
```
def greet(name, greeting="Hello"):
    return f"{greeting}, {name}!"

print(greet("sam"))                # Uses default greeting
print(greet("Bro", "Hi"))            # Uses custom greeting
```

```
Hello, sam!
Hi, Bro!
```

[16]:
```
def variable_length_args(*args):
    return sum(args)

print(variable_length_args(1, 2, 3, 4))  # Outputs: 10
```

```
10
```

[19]:
```
def variable_length_kwargs(**kwargs):
    return {key: value for key, value in kwargs.items()}
```

```
print(variable_length_kwargs(a=6, b=7, c=8))
```

```
{'a': 6, 'b': 7, 'c': 8}
```

[20]:
```
#LIST

def remove_duplicates(lst):
    return list(set(lst))

sample_list = [1, 2, 3, 3, 4, 5, 5]
print(f'List after removing duplicates: {remove_duplicates(sample_list)}')
```

```
List after removing duplicates: [1, 2, 3, 4, 5]
```

[21]:
```
def have_common_member(list1, list2):
    return any(item in list1 for item in list2)

list1 = [1, 2, 3]
list2 = [3, 4, 5]
print(f'Common member exists: {have_common_member(list1, list2)}')
```

```
Common member exists: True
```

[22]:
```
def remove_even_numbers(lst):
    return [num for num in lst if num % 2 != 0]

sample_list = [1, 2, 3, 4, 5, 6, 7, 8, 9]
print(f'Odd numbers: {remove_even_numbers(sample_list)}')
```

```
Odd numbers: [1, 3, 5, 7, 9]
```

[23]:
```
def second_smallest(lst):
    unique_sorted = sorted(set(lst))
    if len(unique_sorted) >= 2:
        return unique_sorted[1]
    else:
        return None

sample_list = [3, 1, 4, 1, 5, 9, 2]
print(f'Second smallest number: {second_smallest(sample_list)}')
```

```
Second smallest number: 2
```

[25]:
```
def split_every_n(lst, n):
    return [lst[i:i+n] for i in range(0, len(lst), n)]

sample_list = [11, 12, 13, 14, 15, 16, 17, 18, 19]
n = 3
```

```
print(f'Split list every {n} elements: {split_every_n(sample_list, n)}')
```

```
Split list every 3 elements: [[11, 12, 13], [14, 15, 16], [17, 18, 19]]
```

[27]:
```
def union_and_intersection(list1, list2):
    union = list(set(list1) | set(list2))
    intersection = list(set(list1) & set(list2))
    return union, intersection

list1 = [11, 12, 13, 14]
list2 = [13,14, 15, 16]
union, intersection = union_and_intersection(list1, list2)
print(f'Union: {union}, Intersection: {intersection}')
```

```
Union: [11, 12, 13, 14, 15, 16], Intersection: [13, 14]
```

[28]:
```
def is_palindrome(lst):
    return lst == lst[::-1]

sample_list = [1, 2, 3, 2, 1]
print(f'Is the list a palindrome? {is_palindrome(sample_list)}')
```

```
Is the list a palindrome? True
```

[30]:
```
def menu():
    lst = []
    while True:
        print("\nMenu:")
        print("1. Insert element")
        print("2. Delete element")
        print("3. Access element")
        print("4. Update element")
        print("5. Traverse list")
        print("6. Exit")

        choice = int(input("Enter your choice: "))

        if choice == 1:
            element = input("Enter element to insert: ")
            lst.append(element)
        elif choice == 2:
            element = input("Enter element to delete: ")
            lst.remove(element) if element in lst else print("Element not found.
 ↪")
        elif choice == 3:
            index = int(input("Enter index to access: "))
            print(f'Element at index {index}: {lst[index]}')
```

```
        elif choice == 4:
            index = int(input("Enter index to update: "))
            new_value = input("Enter new value: ")
            lst[index] = new_value
        elif choice == 5:
            print(f'List elements: {lst}')
        elif choice == 6:
            break
        else:
            print("Invalid choice. Please try again.")

menu()
```

```
Menu:
1. Insert element
2. Delete element
3. Access element
4. Update element
5. Traverse list
6. Exit
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[30], line 34
     31         else:
     32             print("Invalid choice. Please try again.")
---> 34 menu()

Cell In[30], line 12, in menu()
      9 print("5. Traverse list")
     10 print("6. Exit")
---> 12 choice = int(input("Enter your choice: "))
     14 if choice == 1:
     15     element = input("Enter element to insert: ")

ValueError: invalid literal for int() with base 10: ''
```

[31]:
```
nested_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

# Accessing an element
print(f'Access element [1][2]: {nested_list[1][2]}')  # Outputs 6

# Adding an element
nested_list[0].append(10)
print(f'After adding 10 to the first sublist: {nested_list}')
```

```
# Modifying an element
nested_list[2][1] = 20
print(f'After modifying second element of third sublist: {nested_list}')

# Deleting an element
del nested_list[1][0]
print(f'After deleting first element of second sublist: {nested_list}')
```

```
Access element [1][2]: 6
After adding 10 to the first sublist: [[1, 2, 3, 10], [4, 5, 6], [7, 8, 9]]
After modifying second element of third sublist: [[1, 2, 3, 10], [4, 5, 6], [7,
20, 9]]
After deleting first element of second sublist: [[1, 2, 3, 10], [5, 6], [7, 20,
9]]
```

[32]:
```
a = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

# i. Print Complete list
print(f'Complete list: {a}')

# ii. Print 4th element of list
print(f'4th element: {a[3]}')  # Indexing starts at 0

# iii. Print list from 0th to 4th index.
print(f'List from 0th to 4th index: {a[0:5]}')

# iv. Print list from -7th to 3rd element
print(f'List from -7th to 3rd element: {a[-7:4]}')

# v. Appending an element to list.
a.append(110)
print(f'List after appending 110: {a}')

# vi. Sorting the elements of list.
a.sort()
print(f'Sorted list: {a}')

# vii. Popping an element.
popped_element = a.pop()  # Pops the last element
print(f'Popped element: {popped_element}')
print(f'List after popping: {a}')

# viii. Removing a specified element.
a.remove(40)  # Remove first occurrence of 40
print(f'List after removing 40: {a}')
```

```
# ix. Entering an element at specified index.
a.insert(2, 35)  # Insert 35 at index 2
print(f'List after inserting 35 at index 2: {a}')

# x. Counting the occurrence of a specified element.
count_of_50 = a.count(50)
print(f'Occurrence of 50: {count_of_50}')

# xi. Extending the list.
a.extend([120, 130, 140])  # Extend with another list
print(f'List after extending: {a}')

# xii. Reversing the list.
a.reverse()
print(f'Reversed list: {a}')
```

```
Complete list: [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
4th element: 40
List from 0th to 4th index: [10, 20, 30, 40, 50]
List from -7th to 3rd element: [40]
List after appending 110: [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110]
Sorted list: [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110]
Popped element: 110
List after popping: [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
List after removing 40: [10, 20, 30, 50, 60, 70, 80, 90, 100]
List after inserting 35 at index 2: [10, 20, 35, 30, 50, 60, 70, 80, 90, 100]
Occurrence of 50: 1
List after extending: [10, 20, 35, 30, 50, 60, 70, 80, 90, 100, 120, 130, 140]
Reversed list: [140, 130, 120, 100, 90, 80, 70, 60, 50, 30, 35, 20, 10]
```

[33]:
```
def add_matrices(mat1, mat2):
    return [[mat1[i][j] + mat2[i][j] for j in range(len(mat1[0]))] for i in␣
 ↪range(len(mat1))]

matrix1 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
matrix2 = [[9, 8, 7], [6, 5, 4], [3, 2, 1]]

result_matrix = add_matrices(matrix1, matrix2)
print(f'Sum of matrices:\n{result_matrix}')
```

```
Sum of matrices:
[[10, 10, 10], [10, 10, 10], [10, 10, 10]]
```

[34]:
```
l1 = [1, 'x', 4, 5.6, 'z', 9, 'a', 0, 4]
l2 = [item for item in l1 if isinstance(item, int)]
print(f'List with only integer elements: {l2}')
```

```
List with only integer elements: [1, 4, 9, 0, 4]
```

[35]:
```python
#TUPLE

def elementwise_sum(tuples):
    return tuple(sum(values) for values in zip(*tuples))

tuple1 = (1, 2, 3, 4)
tuple2 = (3, 5, 2, 1)
tuple3 = (2, 2, 3, 1)

result = elementwise_sum((tuple1, tuple2, tuple3))
print(f'Element-wise sum of the said tuples: {result}')
```

```
Element-wise sum of the said tuples: (6, 9, 8, 6)
```

[36]:
```python
def tuples_to_lists(tuples_list):
    return [list(t) for t in tuples_list]

original_tuples = [(1, 2), (2, 3), (3, 4)]
converted_lists = tuples_to_lists(original_tuples)
print("Convert the said list of tuples to a list of lists:", converted_lists)
```

```
Convert the said list of tuples to a list of lists: [[1, 2], [2, 3], [3, 4]]
```

[38]:
```python
def remove_empty_tuples(tuples_list):
    return [t for t in tuples_list if t]

original_list = [(1, 2), (2,3), (3, 4), (4,5), (5,6)]
cleaned_list = remove_empty_tuples(original_list)
print("List after removing empty tuples:", cleaned_list)
```

```
List after removing empty tuples: [(1, 2), (2, 3), (3, 4), (4, 5), (5, 6)]
```

[40]:
```python
def string_to_tuple(s):
    return tuple(s)

# Example usage
original_string = "python"
result = string_to_tuple(original_string)
print("Converted string to tuple:", result)
```

```
Converted string to tuple: ('p', 'y', 't', 'h', 'o', 'n')
```

[42]:
```python
import functools
import operator

def tuple_product(t):
    return functools.reduce(operator.mul, t, 1)
```

```
# Example usage
numbers_tuple = (1, 2, 3, 4, 5, 6)
result = tuple_product(numbers_tuple)
print("Product of the numbers in the tuple:", result)
```

```
Product of the numbers in the tuple: 720
```

[43]:
```
squares = [x ** 2 for x in range(1, 21)]
print("List of squares from 1 to 20:", squares)
```

```
List of squares from 1 to 20: [1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144,
169, 196, 225, 256, 289, 324, 361, 400]
```

[44]:
```
#SET

def remove_item_from_set(s, item):
    s.discard(item)  # discard won't raise an error if the item is not present
    return s

# Example usage
my_set = {1, 2, 3, 4}
item_to_remove = 3
result = remove_item_from_set(my_set, item_to_remove)
print("Set after removing item:", result)
```

```
Set after removing item: {1, 2, 4}
```

[45]:
```
def have_no_common_elements(set1, set2):
    return set1.isdisjoint(set2)

# Example usage
set_a = {11, 12, 13}
set_b = {14, 15, 16}
result = have_no_common_elements(set_a, set_b)
print("Do the sets have no common elements?", result)
```

```
Do the sets have no common elements? True
```

[47]:
```
def unique_items(set1, set2):
    return set1.symmetric_difference(set2)

# Example usage
set_a = {11, 12, 13}
set_b = {13, 14, 15}
result = unique_items(set_a, set_b)
print("Unique items from both sets:", result)
```

```
Unique items from both sets: {11, 12, 14, 15}
```

[48]:
```
def set_to_string(s):
    return ''.join(map(str, s))

# Example usage
my_set = {1, 2, 3}
result = set_to_string(my_set)
print("Set converted to string:", result)
```

```
Set converted to string: 123
```

[49]:
```
def count_vowels(s):
    vowels = {'a', 'e', 'i', 'o', 'u'}
    return len(set(c for c in s.lower() if c in vowels))

# Example usage
input_string = "Hello World"
result = count_vowels(input_string)
print("Number of unique vowels in the string:", result)
```

```
Number of unique vowels in the string: 2
```

[50]:
```
cubes_of_even = {x ** 3 for x in range(2, 13) if x % 2 == 0}
print("Set of cubes of even numbers from 2 to 12:", cubes_of_even)
```

```
Set of cubes of even numbers from 2 to 12: {512, 64, 1728, 8, 1000, 216}
```

[52]:
```
def sort_dict_by_value(d):
    ascending = dict(sorted(d.items(), key=lambda item: item[1]))
    descending = dict(sorted(d.items(), key=lambda item: item[1], reverse=True))
    return ascending, descending

# Example usage
my_dict = {'apple': 3, 'mango': 1, 'custurd apple': 2}
asc, desc = sort_dict_by_value(my_dict)
print("Sorted (ascending):", asc)
print("Sorted (descending):", desc)
```

```
Sorted (ascending): {'mango': 1, 'custurd apple': 2, 'apple': 3}
Sorted (descending): {'apple': 3, 'custurd apple': 2, 'mango': 1}
```

[53]:
```
def remove_duplicates(d):
    return {key: value for key, value in d.items()}

# Example usage
my_dict = {'a': 1, 'b': 2, 'a': 3}
result = remove_duplicates(my_dict)
print("Dictionary after removing duplicates:", result)
```

```
Dictionary after removing duplicates: {'a': 3, 'b': 2}
```

[55]:
```python
def combine_dicts(dict1, dict2):
    combined = dict1.copy()  # create a copy of the first dictionary
    for key, value in dict2.items():
        combined[key] = combined.get(key, 0) + value
    return combined

# Example usage
dict_a = {'a': 7, 'b': 8}
dict_b = {'b': 9, 'c': 10}
result = combine_dicts(dict_a, dict_b)
print("Combined dictionary:", result)
```

```
Combined dictionary: {'a': 7, 'b': 17, 'c': 10}
```

[56]:
```python
def count_letters(s):
    return {char: s.count(char) for char in set(s) if char.isalpha()}

# Example usage
input_string = "hello world"
result = count_letters(input_string)
print("Letter count dictionary:", result)
```

```
Letter count dictionary: {'w': 1, 'd': 1, 'e': 1, 'r': 1, 'h': 1, 'o': 2, 'l':
3}
```

[57]:
```python
def match_key_value(dict1, dict2):
    return {key: value for key, value in dict1.items() if key in dict2 and␣
 ↪dict1[key] == dict2[key]}

# Example usage
dict_a = {'a': 1, 'b': 2, 'c': 3}
dict_b = {'b': 2, 'c': 4}
result = match_key_value(dict_a, dict_b)
print("Matching keys and values:", result)
```

```
Matching keys and values: {'b': 2}
```

[58]:
```python
def convert_price_to_pound(old_price):
    return {item: round(price * 0.73, 2) for item, price in old_price.items()} ␣
 ↪# Assuming 1 dollar = 0.73 pounds

# Example usage
old_price = {'milk': 1.02, 'coffee': 2.5, 'bread': 2.5}
result = convert_price_to_pound(old_price)
print("Prices in pounds:", result)
```

```
Prices in pounds: {'milk': 0.74, 'coffee': 1.82, 'bread': 1.82}
```

[60]:
```python
def filter_even_age(original_dict):
    return {name: age for name, age in original_dict.items() if age % 2 == 0}

# Example usage
original_dict = {'jayden': 28, 'michael': 38, 'mia': 47, 'emma': 33}
result = filter_even_age(original_dict)
print("Dictionary with even ages:", result)
```

```
Dictionary with even ages: {'jayden': 28, 'michael': 38}
```