

Day 8 – Python Session – Accenture bootcamp

Name: Pothumulla Kankanamge Mewan Madhusha

Table of Contents

Task 01	3
Task 2	6
List Operations:.....	6
Dictionary Operations:.....	9
Combining Lists and Dictionaries:.....	13
Extra work(<i>hands-on experience</i>)	17
List.....	17
Dictionary.....	18

Task 01

Please create a document, write a Python program, and share it in this chat.

Fibonacci Sequence

Factorial Calculation

Prime Numbers

Palindrome Check

Answer

```
# Fibonacci Sequence
# Factorial Calculation
# Prime Numbers
# Palindrome Check

def fibonacci(numOfterms):
    fibo_seq = [0,1]
    while len(fibo_seq) < numOfterms:
        fibo_seq.append(fibo_seq[-1] + fibo_seq[-2])
    return fibo_seq[:numOfterms]

def factorial(num):
    if num == 0 or num == 1:
        return 1
    return num * factorial(num - 1)

def prime_numbercheck(num):
    if num < 2:
        return False
    for i in range(2, int(num**0.5) + 1):
        if num % i == 0:
            return False
    return True

# I have just used python reversed string function
def is_palindrome(word):
    return word == word[::-1]

def main():
    while True:
        print("\nChoose an operation:")
        print("1. Fibonacci Sequence")
        print("2. Factorial Calculation")
        print("3. Prime Numbers Check")
        print("4. Palindrome Check")
        print("5. Exit")
        choice = input("Enter the number of your choice (1-5): ")

        if choice == '1':
```

```
n = int(input("Enter the number of terms for the Fibonacci sequence:"))
result = fibonacci(n)
print(f"Fibonacci Sequence: {result}")

elif choice == '2':
    num = int(input("Enter a number to calculate its factorial: "))
    result = factorial(num)
    print(f"Factorial of {num}: {result}")

elif choice == '3':
    num = int(input("Enter a number to check if it's prime: "))
    result = prime_numbercheck(num)
    print(f"{num} is {'prime' if result else 'not prime'}.")

elif choice == '4':
    word = input("Enter a word to check if it's a palindrome: ").lower()
    result = is_palindrome(word)
    print(f"{word.capitalize()} is {'a palindrome' if result else 'not a
palindrome'}.")

elif choice == '5':
    print("Exiting the program. Goodbye!")
    break

else:
    print("Invalid choice. Please enter a number between 1 and 5.")

if __name__ == "__main__":
    main()
```

Output answers

The screenshot shows a terminal window in VS Code executing a Python script named `task1.py`. The script contains several functions: `Fibonacci Sequence`, `Factorial Calculation`, `Prime Numbers`, and `Palindrome Check`. The terminal output shows the script prompting the user to choose an operation (1-5) and then performing the chosen operation. For example, it checks if the word "mewan" is a palindrome and asks for a prime number to check its primality.

```
# Fibonacci Sequence
# Factorial Calculation
# Prime Numbers
# Palindrome Check
# def Fibonacci(n):
#     if n == 0:
#         return 0
#     elif n == 1:
#         return 1
#     else:
#         return task1.Fibonacci(n-1) + task1.Fibonacci(n-2)

# Choose an operation:
# 1. Fibonacci Sequence
# 2. Factorial Calculation
# 3. Prime Numbers Check
# 4. Palindrome Check
# 5. Exit
# Enter the number of your choice (1-5): 3
# Enter a number to check if it's prime: 7
# 7 is prime.

# Choose an operation:
# 1. Fibonacci Sequence
# 2. Factorial Calculation
# 3. Prime Numbers Check
# 4. Palindrome Check
# 5. Exit
# Enter the number of your choice (1-5): 4
# Enter a word to check if it's a palindrome: mewan
# Mewan is not a palindrome.

# Choose an operation:
# 1. Fibonacci Sequence
# 2. Factorial Calculation
# 3. Prime Numbers Check
# 4. Palindrome Check
# 5. Exit
# Enter the number of your choice (1-5): ^[[A
# Invalid choice. Please enter a number between 1 and 5.

# Choose an operation:
# 1. Fibonacci Sequence
# 2. Factorial Calculation
# 3. Prime Numbers Check
# 4. Palindrome Check
# 5. Exit
# Enter the number of your choice (1-5): 4
# Enter a word to check if it's a palindrome: level
# Level is a palindrome.

# Choose an operation:
# 1. Fibonacci Sequence
# 2. Factorial Calculation
# 3. Prime Numbers Check
# 4. Palindrome Check
# 5. Exit
# Enter the number of your choice (1-5): ■
```

Task 2

Complete this task. You can seek help online, but if you understand the process, feel free to proceed. Otherwise, note down your questions, and we will discuss them in the end-time session.

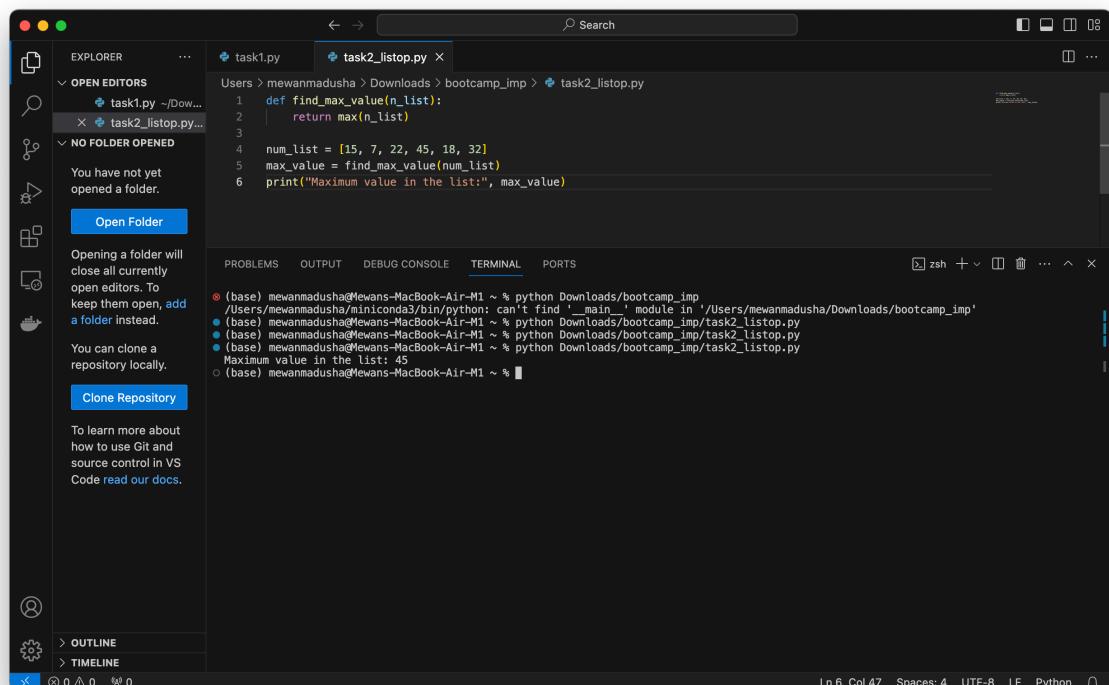
List Operations:

1. Write a Python function to find the maximum value in a list of numbers.

```
def find_max_value(n_list):
    return max(n_list)

num_list = [15, 7, 22, 45, 18, 32]
max_value = find_max_value(num_list)
print("Maximum value in the list:", max_value)
```

Answer



The screenshot shows the Visual Studio Code interface. The Explorer sidebar on the left indicates that 'task1.py' is open. The main editor area contains the following Python code:

```
def find_max_value(n_list):
    return max(n_list)

num_list = [15, 7, 22, 45, 18, 32]
max_value = find_max_value(num_list)
print("Maximum value in the list:", max_value)
```

Below the code, the Terminal tab is active, showing the output of running the script:

```
(base) mewanmadusha@Mewans-MacBook-Air-M1 ~ % python Downloads/bootcamp_im...
Users/mewanmadusha/miniconda3/bin/python: can't find '_main_' module in '/Users/mewanmadusha/Downloads/bootcamp_im...
(base) mewanmadusha@Mewans-MacBook-Air-M1 ~ % python Downloads/bootcamp_im.../task2_listop.py
(base) mewanmadusha@Mewans-MacBook-Air-M1 ~ % python Downloads/bootcamp_im.../task2_listop.py
(base) mewanmadusha@Mewans-MacBook-Air-M1 ~ % python Downloads/bootcamp_im.../task2_listop.py
Maximum value in the list: 45
(base) mewanmadusha@Mewans-MacBook-Air-M1 ~ %
```

The status bar at the bottom right shows 'Ln 6, Col 47' and 'Spaces: 4'.

2. append, Extend and Insert works explain in code.

```

# append: Adding an element to the end of the list
aei_list = [1, 2, 3]
aei_list.append(4)
print("aie_list after append:", aei_list)

# Extend: Adding elements from an iterable to the end of the list
aei_list.extend([5, 6, 7])
print("aie_list after extend:", aei_list)

# Insert: Inserting an element at a specific index in the list(we should mention
# the index)
aei_list.insert(1, 8)
print("aie_list after insert:", aei_list)

```

The screenshot shows the Visual Studio Code interface. The left sidebar displays the file structure with 'task1.py' and 'task2_listop.py' open. The main editor window contains the provided Python code for list manipulation. Below the editor, the terminal tab is active, showing the output of running the script:

```

(base) mewanmadusha@Mewans-MacBook-Air-M1 ~ % python Downloads/bootcamp_imp/task2_listop.py
aie_list after append: [1, 2, 3, 4]
aie_list after extend: [1, 2, 3, 4, 5, 6, 7]
aie_list after insert: [1, 8, 2, 3, 4, 5, 6, 7]
(base) mewanmadusha@Mewans-MacBook-Air-M1 ~ %

```

3. Implement a program to remove duplicates from a given list.

```

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

# Example usage:
first_list = [1, 2, 2, 3, 4, 4, 5]
unique_list = remove_duplicates(first_list)
print("List with duplicates:", first_list)
print("List without duplicates:", unique_list)

```

```
task1.py task2_listop.py
Users > mewanmadusha > Downloads > bootcamp_1mp > task2_listop.py
26
27     def remove_duplicates(n_lst):
28         return list(set(n_lst))
29
30     # Example usage:
31     first_list = [1, 2, 2, 3, 4, 4, 5]
32     unique_list = remove_duplicates(first_list)
33     print("List with duplicates:", first_list)
34     print("List without duplicates:", unique_list)

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
(base) mewanmadusha@Mewans-MacBook-Air-M1 ~ % python Downloads/bootcamp_1mp/task2_listop.py
List with duplicates: [1, 2, 2, 3, 4, 4, 5]
List without duplicates: [1, 2, 3, 4, 5]
(base) mewanmadusha@Mewans-MacBook-Air-M1 ~ %

Ln 34, Col 47 Spaces: 4 UTF-8 LF Python
```

4. Create a function to check if a list is sorted in ascending order.

```
def check_sortedAscending(n_list):
    return all(n_list[i] <= n_list[i + 1] for i in range(len(n_list) - 1))

sorted_list = [1, 3, 5, 7, 9]
unsorted_list = [4, 2, 8, 6, 10]

print("Is sorted (ascending):", check_sortedAscending(sorted_list))
print("Is sorted (ascending):", check_sortedAscending(unsorted_list))
```

```
task1.py task2_listop.py
Users > mewanmadusha > Downloads > bootcamp_1mp > task2_listop.py
32     # unique_list = remove_duplicates(first_list)
33     # print("List with duplicates:", first_list)
34     # print("List without duplicates:", unique_list)
35
36     # 1.4
37
38     def check_sortedAscending(n_list):
39         return all(n_list[i] <= n_list[i + 1] for i in range(len(n_list) - 1))
40
41     sorted_list = [1, 3, 5, 7, 9]
42     unsorted_list = [4, 2, 8, 6, 10]
43
44     print("Is sorted (ascending):", check_sortedAscending(sorted_list))
45     print("Is sorted (ascending):", check_sortedAscending(unsorted_list))

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
(base) mewanmadusha@Mewans-MacBook-Air-M1 ~ %
(base) mewanmadusha@Mewans-MacBook-Air-M1 ~ %
(base) mewanmadusha@Mewans-MacBook-Air-M1 ~ %
(base) mewanmadusha@Mewans-MacBook-Air-M1 ~ %
Is sorted (ascending): True
Is sorted (ascending): False
(base) mewanmadusha@Mewans-MacBook-Air-M1 ~ %

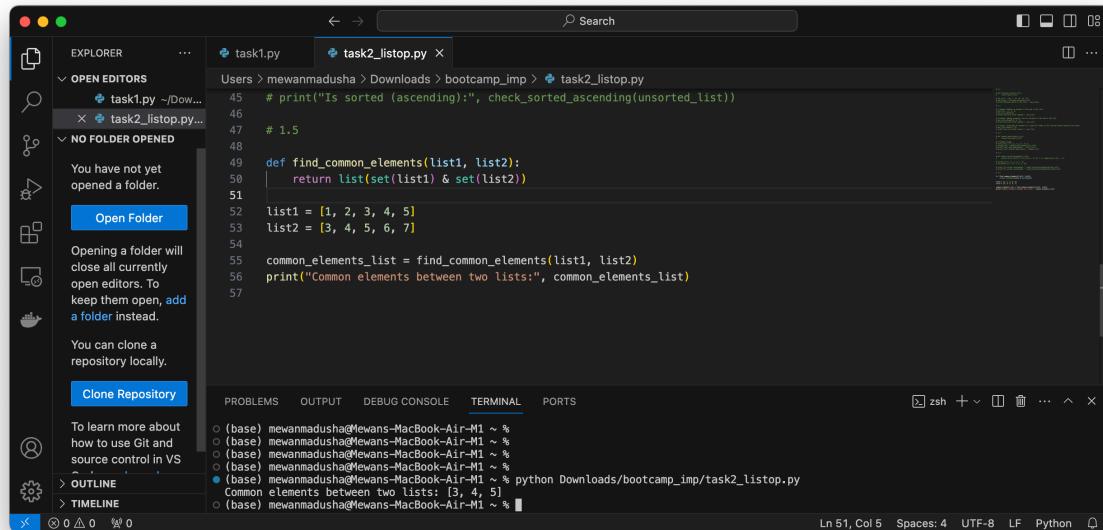
Ln 45, Col 55 Spaces: 4 UTF-8 LF Python
```

- Write a program to find the common elements between two lists.

```
def find_common_elements(list1, list2):
    return list(set(list1) & set(list2))

list1 = [1, 2, 3, 4, 5]
list2 = [3, 4, 5, 6, 7]

common_elements_list = find_common_elements(list1, list2)
print("Common elements between two lists:", common_elements_list)
```



Dictionary Operations:

- Create a Python dictionary representing a student's grades (subject: score). Write a function to find the average score.

```
def find_average_score(grades):
    return sum(grades.values()) / len(grades)

student_grades = {'Math': 85, 'English': 90, 'Science': 78, 'History': 92}

avg_score = find_average_score(student_grades)
print("Average score:", avg_score)
```

```

task1.py task2_listop.py task2_dictop.py
Users > mewanmadusha > Downloads > bootcamp_imp > task2_dictop.py
1 # 2.1
2
3 def find_average_score(grades):
4     return sum(grades.values()) / len(grades)
5
6 student_grades = {'Math': 85, 'English': 90, 'Science': 78, 'History': 92}
7
8 avg_score = find_average_score(student_grades)
9 print("Average score:", avg_score)

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

(base) mewanmadusha@Mewans-MacBook-Air-M1 ~ % python Downloads/bootcamp_imp/task2_dictop.py
(base) mewanmadusha@Mewans-MacBook-Air-M1 ~ %
Average score: 86.25
(base) mewanmadusha@Mewans-MacBook-Air-M1 ~ %

```

Ln 9, Col 35 Spaces: 4 UTF-8 LF Python

2. Implement a program to merge two dictionaries.

Hear I used dictionary unpacking syntax that available after python 3.5

```

def merge_dictionaries(dict1, dict2):
    merged_dict = {**dict1, **dict2}
    return merged_dict

# Example usage:
dict1 = {'a': 'pothumulla', 'b': 'kankanamge'}
dict2 = {'b': 'mewan', 'c': 'madusha'}
merged_dict = merge_dictionaries(dict1, dict2)
# merged_dict = merge_dictionaries(dict2, dict1)
print("Merged dictionary:", merged_dict)

```

Merged dictionary included all the keys but if there is any common keys, based on the unpacking second dictionary value will be the override value of the merged_dict.

In the below code I also shown both above scenarios.

```

task1.py task2_listop.py task2_dictop.py
Users > mewanmadusha > Downloads > bootcamp_imp > task2_dictop.py
5
6 # student_grades = {'Math': 85, 'English': 90, 'Science': 78, 'History': 92}
7
8 # avg_score = find_average_score(student_grades)
9 # print("Average score:", avg_score)
10
11 # 2.1
12
13 def merge_dictionaries(dict1, dict2):
14     merged_dict = {**dict1, **dict2}
15     return merged_dict
16
17 # Example usage:
18 dict1 = {'a': 'pothumulla', 'b': 'kankanamge'}
19 dict2 = {'b': 'mewan', 'c': 'madusha'}
20 merged_dict = merge_dictionaries(dict1, dict2)
21 # merged_dict = merge_dictionaries(dict2, dict1)
22 print(["Merged dictionary:", merged_dict])

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

- (base) mewanmadusha@Mewans-MacBook-Air-M1 ~ % python Downloads/bootcamp_imp/task2_dictop.py
 Average score: 86.25
- (base) mewanmadusha@Mewans-MacBook-Air-M1 ~ % python Downloads/bootcamp_imp/task2_dictop.py
 Merged dictionary: {'a': 'pothumulla', 'b': 'mewan', 'c': 'madusha'}
- (base) mewanmadusha@Mewans-MacBook-Air-M1 ~ % python Downloads/bootcamp_imp/task2_dictop.py
 Merged dictionary: {'b': 'kankanamge', 'c': 'madusha', 'a': 'pothumulla'}

Ln 22, Col 41 Spaces: 4 UTF-8 LF Python

3. Create a program to find the keys with the top N values in a dictionary.

Sorted list returns a new sorted list from the elements of the provided iterable

Iterable is values of dictionary, dictionary.get means sorting based on dictionary values

Reverse=true – sort in to descending order because in the slicing step we can cut out elements from start

[n] – slicing the list and then get only first N number of elements

Assume n=2

```

def top_n_keys(dictionary, n):
    sorted_keys = sorted(dictionary, key=dictionary.get, reverse=True)
    return sorted_keys[:n]

n=2
grades = {'Math': 85, 'English': 90, 'Science': 78, 'History': 92}
top_keys = top_n_keys(grades, n)
print(f"Top {n} keys with highest values: {top_keys}")

```

```
def top_n_keys(dictionary, n):
    sorted_keys = sorted(dictionary, key=dictionary.get, reverse=True)
    return sorted_keys[:n]

n=2
grades = {'Math': 85, 'English': 90, 'Science': 78, 'History': 92}
top_keys = top_n_keys(grades, n)
print("Top {} keys with highest values: {}".format(n, top_keys))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

(base) mewanmadusha@Mewans-MacBook-Air-M1 ~ %
(base) mewanmadusha@Mewans-MacBook-Air-M1 ~ %
● (base) mewanmadusha@Mewans-MacBook-Air-M1 ~ % python Downloads/bootcamp_omp/task2_dictop.py
Top 2 keys with highest values: ['History', 'English']
(base) mewanmadusha@Mewans-MacBook-Air-M1 ~ % []

4. Implement a function to sort a dictionary by its values.

```
def sort_dict_by_values(dictionary):
    sorted_dict = dict(sorted(dictionary.items(), key=lambda item: item[1]))
    return sorted_dict

unsorted_dict = {'b': 3, 'a': 1, 'c': 4}
sorted_dict = sort_dict_by_values(unsorted_dict)
print("Sorted dictionary by values:", sorted_dict)
```

```
def sort_dict_by_values(dictionary):
    sorted_dict = dict(sorted(dictionary.items(), key=lambda item: item[1]))
    return sorted_dict

unsorted_dict = {'b': 3, 'a': 1, 'c': 4}
sorted_dict = sort_dict_by_values(unsorted_dict)
print("Sorted dictionary by values:", sorted_dict)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

(base) mewanmadusha@Mewans-MacBook-Air-M1 ~ % python Downloads/bootcamp_omp/task2_dictop.py
Top 2 keys with highest values: ['History', 'English']
● (base) mewanmadusha@Mewans-MacBook-Air-M1 ~ % python Downloads/bootcamp_omp/task2_dictop.py
Sorted dictionary by values: {'a': 1, 'b': 3, 'c': 4}
(base) mewanmadusha@Mewans-MacBook-Air-M1 ~ % []

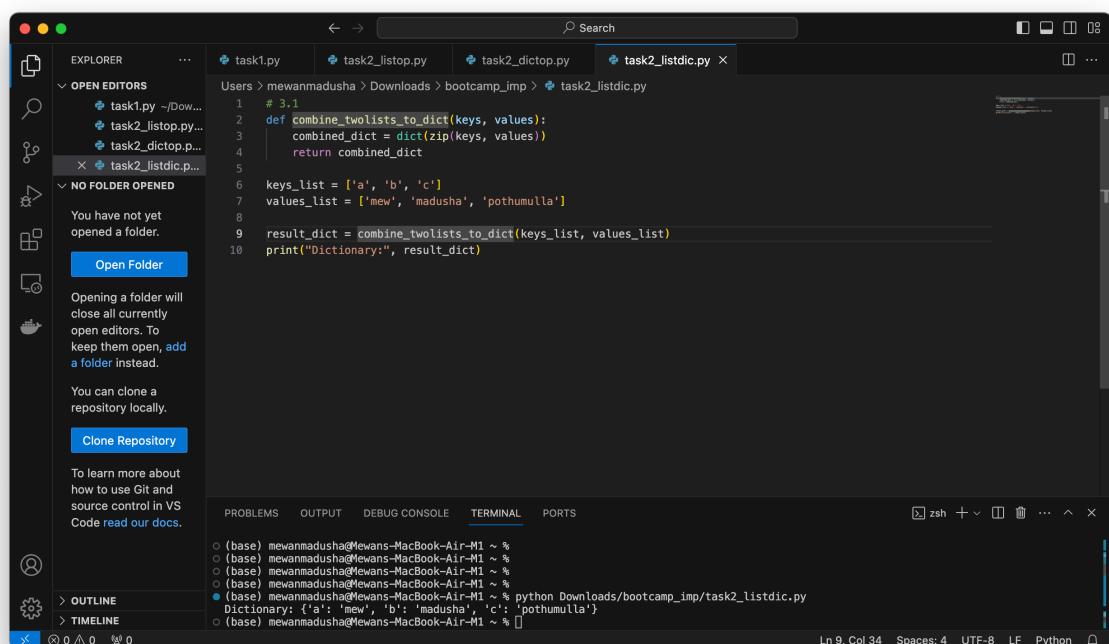
Combining Lists and Dictionaries:

1. Write a Python program that combines two lists into a dictionary where elements from the first list are keys and elements from the second list are values.

```
def combine_twolists_to_dict(keys, values):
    combined_dict = dict(zip(keys, values))
    return combined_dict

keys_list = ['a', 'b', 'c']
values_list = ['mew', 'madusha', 'pothumulla']

result_dict = combine_twolists_to_dict(keys_list, values_list)
print("Dictionary:", result_dict)
```



2. Write a program to find the most frequent element in a list and its count.

```
def get_most_frequent_element(lst):
    most_frequent_element = max(set(lst), key=lst.count)
    count = lst.count(most_frequent_element)
    return most_frequent_element, count

my_list = [1, 2, 2, 3, 3, 3, 4, 4, 4]
```

```
most_frequent, count = get_most_frequent_element(my_list)
print(f"Most frequent element: {most_frequent}, Count: {count}")
```

The screenshot shows the Visual Studio Code interface. On the left is the Explorer sidebar with a 'NO FOLDER OPENED' message. In the center, there are four tabs: 'task1.py', 'task2_listop.py', 'task2_dictop.py', and 'task2_listdic.py'. The 'task2_listdic.py' tab is active, displaying the following Python code:

```
22 # my_list = [1, 2, 2, 3, 3, 4, 4, 4]
23
24 # most_frequent, count = most_frequent_element(my_list)
25
26
27 def get_most_frequent_element(lst):
28     most_frequent_element = max(set(lst), key=lst.count)
29     count = lst.count(most_frequent_element)
30     return most_frequent_element, count
31
32
33 my_list = [1, 2, 2, 3, 3, 4, 4, 4]
34
35 most_frequent, count = get_most_frequent_element(my_list)
36 print(f"Most frequent element: {most_frequent}, Count: {count}")
```

Below the code editor is the Terminal panel, which shows the command-line output of running the script:

```
(base) mewanmadusha@Mewans-MacBook-Air-M1 ~ % python Downloads/bootcamp_1mp/task2_listdic.py
Dictionary: {'a': 'mew', 'b': 'madusha', 'c': 'pothumulla'}
Most frequent element: 4, Count: 4
(base) mewanmadusha@Mewans-MacBook-Air-M1 ~ % python Downloads/bootcamp_1mp/task2_listdic.py
Most frequent element: 4, Count: 4
(base) mewanmadusha@Mewans-MacBook-Air-M1 ~ %
```

The status bar at the bottom indicates 'Ln 36, Col 65'.

3. Create a program to calculate the total sum of values for each key in a list of dictionaries.

```
def calculate_total_sum_of_values(list_of_dicts):
    result_dict = {}
    for d in list_of_dicts:
        for key, value in d.items():
            result_dict[key] = result_dict.get(key, 0) + value
    return result_dict

list_of_dicts = [{'a': 10, 'b': 20}, {'a': 5, 'c': 15}, {'d': 6, 'c': 15}]

sum_of_values = calculate_total_sum_of_values(list_of_dicts)
print("Total sum of values for each key:", sum_of_values)
```

A screenshot of the Visual Studio Code interface. The Explorer sidebar shows a folder named 'mewanmadusha' containing several Python files: task1.py, task2_listop.py, task2_dictop.py, and task2_listdic.py. The task2_listdic.py file is currently open in the editor. The code defines a function calculate_total_sum_of_values that takes a list of dictionaries and returns a dictionary where each key's value is the sum of its original values across all dictionaries. A terminal window at the bottom shows the command 'python Downloads/bootcamp_imp/task2_listdic.py' being run, and the output 'Total sum of values for each key: {"a": 15, "b": 20, "c": 30, "d": 6}' is displayed.

```
def calculate_total_sum_of_values(list_of_dicts):
    result_dict = {}
    for d in list_of_dicts:
        for key, value in d.items():
            result_dict[key] = result_dict.get(key, 0) + value
    return result_dict

list_of_dicts = [{"a": 10, "b": 20}, {"a": 5, "c": 15}, {"d": 6, "c": 15}]
sum_of_values = calculate_total_sum_of_values(list_of_dicts)
print("Total sum of values for each key:", sum_of_values)
```

4. Write a Python program to flatten a nested list.

```
def flatten_nested_list(nested_list):
    return [item for sublist in nested_list for item in sublist]

nested_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
flattened_list = flatten_nested_list(nested_list)
print(flattened_list)
```

A screenshot of the Visual Studio Code interface. The Explorer sidebar shows the same folder structure as the previous screenshot. The task2_listdic.py file is now open in the editor. The code has been modified to define a new function flatten_nested_list that takes a nested list and returns a flat list. The terminal window shows the command 'python Downloads/bootcamp_imp/task2_listdic.py' being run, and the output '[1, 2, 3, 4, 5, 6, 7, 8, 9]' is displayed.

```
def flatten_nested_list(nested_list):
    return [item for sublist in nested_list for item in sublist]

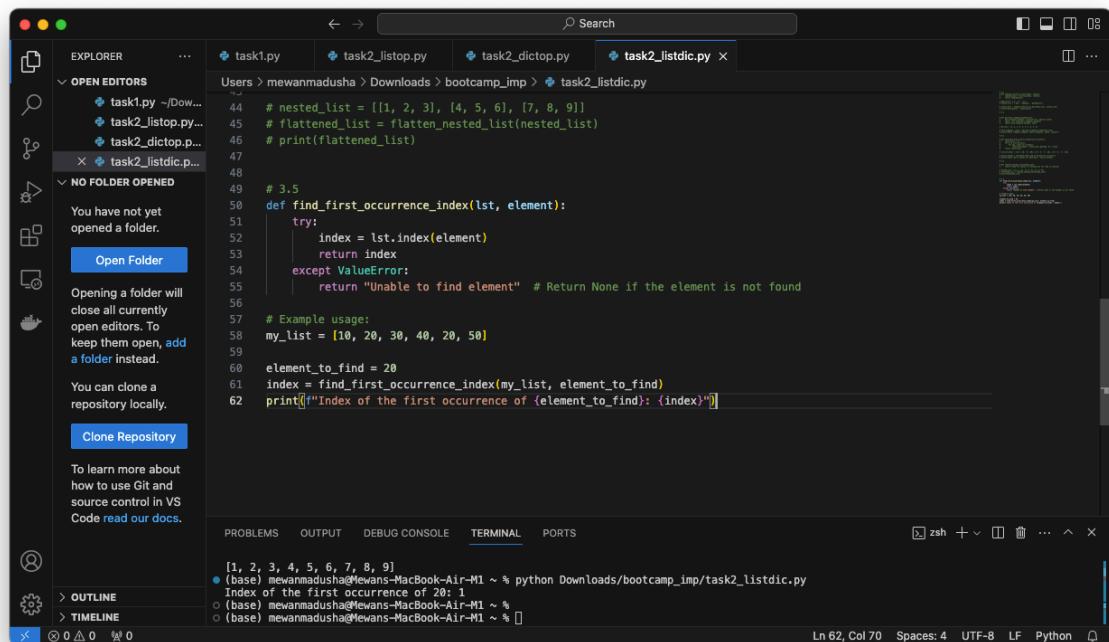
nested_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
flattened_list = flatten_nested_list(nested_list)
print(flattened_list)
```

5. Write a function to find the index of the first occurrence of a specific element in a list.

```
def find_first_occurrence_index(lst, element):
    try:
        index = lst.index(element)
        return index
    except ValueError:
        return "Unable to find element" # Return None if the element is not found

my_list = [10, 20, 30, 40, 20, 50]

element_to_find = 20
index = find_first_occurrence_index(my_list, element_to_find)
print(f"Index of the first occurrence of {element_to_find}: {index}")
```



Extra work(hands-on experience)

List

`list.append(elem)` -- adds a single element to the end of the list. Common error: does not return the new list, just modifies the original.

`list.insert(index, elem)` - inserts the element at the given index, shifting elements to the right.

`list.extend(list2)` adds the elements in list2 to the end of the list. Using + or += on a list is similar to using extend).

```
[>>> a="mewan 'madusha'"
[>>> print(a)
mewan 'madusha'
>>>
>>>
[>>> cities = ['Riga','Colombo','Rome','Tokyo','WDC']
[>>> print(cities)
['Riga', 'Colombo', 'Rome', 'Tokyo', 'WDC']
[>>> cities.append('Beijing')
[>>> print(cities)
['Riga', 'Colombo', 'Rome', 'Tokyo', 'WDC', 'Beijing']
[>>> cities.insert(1,'Moscow')
[>>> print(cities)
['Riga', 'Moscow', 'Colombo', 'Rome', 'Tokyo', 'WDC', 'Beijing']
[>>> list2 = ['test1','test2']
[>>> list2
['test1', 'test2']
[>>>
[>>> cities.extend(list2)
[>>> print(cities)
['Riga', 'Moscow', 'Colombo', 'Rome', 'Tokyo', 'WDC', 'Beijing', 'test1', 'test2']
[>>>
```

`list.index(elem)` -- searches for the given element from the start of the list and returns its index. Throws a `ValueError` if the element does not appear (use "in" to check without a `ValueError`).

```
[>>> if 'Moscow' in cities:
[...     print("Moscow available")
[... else:
[...     print("Moscow Not available")
[...
Moscow available
```

`list.remove(elem)` -- searches for the first instance of the given element and removes it (throws `ValueError` if not present)

```
[>>> try:
[...     cities.remove("WDC")
[... except ValueError:
[...     print("element is not in the list")
[...
>>> cities
['Riga', 'Moscow', 'Colombo', 'Rome', 'Tokyo', 'Beijing', 'test1', 'test2']
>>> try:
[...     cities.remove("WDC")
[... except:
[...     print("no value")
[...
no value
```

`list.sort()` -- sorts the list in place (does not return it). (The `sorted` function shown later is preferred.)

`list.reverse()` -- reverses the list in place (does not return it)

`list.pop(index)` -- removes and returns the element at the given index. Returns the rightmost element if index is omitted (roughly the opposite of `append()`).

```
[>>> cities.sort()
[>>> cities
['Beijing', 'Colombo', 'Moscow', 'Riga', 'Rome', 'Tokyo', 'test1', 'test2']
[>>> cities.reverse()
[>>> cities
['test2', 'test1', 'Tokyo', 'Rome', 'Riga', 'Moscow', 'Colombo', 'Beijing']
[>>>
[>>> removing_elem = cities.pop(1)
[>>> removing_elem
'test1'
[>>> cities
['test2', 'Tokyo', 'Rome', 'Riga', 'Moscow', 'Colombo', 'Beijing']
[>>> ]
```

Dictionary

Dictionary

```
[>>> dict = {}
[>>> dict['name'] = 'mewan'
[>>> dict['lname'] = 'madusha'
[>>>
[>>> print(dict)
{'name': 'mewan', 'lname': 'madusha'}
[>>> print(dict['name'])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'name' is not defined
[>>> print(dict['name'])
mewan
[>>> dict['name']= 'pothumulla'
[>>>
[>>> 'name' in dict
True
[>>> 'madusha' in dict
False
[>>> dict
{'name': 'pothumulla', 'lname': 'madusha'}
```



```
[>>> if 'name' in dict:
...     dict
...
{'name': 'pothumulla', 'lname': 'madusha'}
[>>>
[>>> try:
...     print(dict.get('name'))
... except:
...     print('no key')
...
pothumulla
```