

Lab: Get a Solution to the Given Coding Problem



Estimated Time: 40 mins

Introduction:

In the dynamic world of coding, ChatGPTs serve as invaluable assistants for swiftly tackling coding problems. Knowing how to effectively utilize a ChatGPT can streamline your problem-solving process and provide timely, personalized guidance. This guide explores the best practices and strategies for using ChatGPTs to obtain solutions to coding challenges, making the coding experience more efficient and rewarding. Let's dive into the world of ChatGPT-assisted coding solutions and elevate your problem-solving capabilities.

Learning Objectives:

After completing this lab, you should be able to perform the following tasks:

- Understand how to use generative AI in a step-by-step approach to obtain solutions for coding problems.
- Craft questions for the ChatGPT to ensure accurate and anticipated outcomes for coding problems.

Please note that generative AI is an evolving field. As you attempt the labs, your experience and output might be different than what is seen here.

In case you need familiarity with the Interface/classroom please see the [Hands-on-Lab: Get familiar with GenAI Classroom](#) for reference.

Task 1

Step 1: Asking questions to ChatGPT

In the Prompt Instructions field, please enter the following prompt in order to instruct ChatGPT to offer thorough and detailed step-by-step solutions for coding problems within the Python programming language. The goal is to receive in-depth explanations and guidance, facilitating a comprehensive understanding of the coding solutions provided by ChatGPT.

Provide a comprehensive and detailed step-by-step solution to the coding problems in the Python programming language.

Now, let us ask a few questions to ChatGPT to get solutions for coding problems.

Problem Statement:

Write a Python program to find the maximum and minimum elements in a list.

Explanation:

- The problem statement explicitly outlines the task of identifying the maximum and minimum elements, ensuring clarity in the question.
- Python is explicitly specified as the programming language to be used.
- The prompt emphasizes the importance of requesting a step-by-step solution.
- It is important to verify the accuracy of the output, ensuring alignment with the expected results.

▼ [Click here to view the sample response generated](#)

Sure! Here's a step-by-step solution to write a Python program that finds the maximum and minimum elements in a list.

Step 1: Define the List

First, you need to have a list of numbers. You can either define it manually or take input from the user.

Step 2: Finding Maximum and Minimum

You can use Python's built-in `max()` and `min()` functions to find the maximum and minimum values in the list.

Step 3: Print the Results

Finally, you will print the maximum and minimum values.

Complete Code Example

Here's a complete Python program that implements the above steps:

```
Step 1: Define the list
numbers = [3, 5, 1, 8, 2, 7, 4]
Alternatively, you can get input from the user
numbers = list(map(int, input("Enter numbers separated by space: ").split()))
Step 2: Find maximum and minimum
maximum_value = max(numbers)
minimum_value = min(numbers)
Step 3: Print the results
print("Maximum value in the list:", maximum_value)
print("Minimum value in the list:", minimum_value)
```

Additional Details:

1. **Input from User:** If you want to allow the user to input the list, you can uncomment the input code and comment out the predefined list.
2. **Error Handling:** You may want to add error handling to manage cases where the list is empty. Here's how you could do that:

```
# Check if the list is empty
if not numbers:
```

```
    print("The list is empty. Cannot find maximum and minimum values.")
else:
    maximum_value = max(numbers)
    minimum_value = min(numbers)
    print("Maximum value in the list:", maximum_value)
    print("Minimum value in the list:", minimum_value)
```

3. **Performance:** The `max()` and `min()` functions both run in $O(n)$ time complexity, where n is the number of elements in the list, making them efficient for this task.
4. **Using Loops:** If you want to implement the maximum and minimum finding logic without using built-in functions, you can do so with a loop:

```
if numbers:
    maximum_value = numbers[0]
    minimum_value = numbers[0]

    for number in numbers:
        if number > maximum_value:
            maximum_value = number
        if number < minimum_value:
            minimum_value = number

    print("Maximum value in the list:", maximum_value)
    print("Minimum value in the list:", minimum_value)
else:
    print("The list is empty. Cannot find maximum and minimum values.")
```

Feel free to modify the list or input method based on your needs!

Task 2

Problem Statement:

Verify whether the provided code can be optimized:

```
def find_largest(arr):  
    largest = arr[0]  
  
    for num in arr:  
        if num > largest:  
            largest = num  
    return largest
```

Explanation:

- Develop a Python function to find the largest element in an array.
- Python is explicitly chosen as the programming language.

▼ Click here to view the sample response generated

The provided code for finding the largest number in a list can be optimized in terms of readability and efficiency. However, the algorithm itself is already quite efficient with a time complexity of $O(n)$, where n is the number of elements in the list.

Review of the Provided Code:

Here's the original code for reference:

```
def find_largest(arr):  
    largest = arr[0]  
  
    for num in arr:  
        if num > largest:  
            largest = num  
    return largest
```

Potential Improvements

1. **Handling Empty Lists:** The current implementation assumes that the list is non-empty. It's a good practice to handle the case when the list is empty to avoid an `IndexError`.
2. **Using Built-in Functions:** Python provides a built-in `max()` function that is optimized and easier to read. Using built-in functions is generally preferred unless there's a specific reason to implement the logic manually.
3. **Improving Readability:** The variable names can be made more descriptive.

Optimized Code Example

Here's an optimized version of the function that includes error handling and uses the built-in `max()` function:

```
def find_largest(arr):  
    if not arr: # Check if the list is empty  
        return None # or raise an exception, depending on your needs  
    return max(arr) # Using the built-in max function  
Example usage  
numbers = [3, 5, 1, 8, 2]  
largest_number = find_largest(numbers)  
if largest_number is not None:  
    print("The largest number in the list is:", largest_number)  
else:  
    print("The list is empty.")
```

Explanation of the Optimized Code

1. **Empty List Check:** The function checks if the list is empty. If it is, it returns `None`. You could also choose to raise an exception if that fits your use case better.
2. **Use of `max()`:** The built-in `max()` function is used to find the largest number in the list, which simplifies the code and leverages Python's internal optimizations.

Additional Considerations

- **Performance:** The built-in `max()` function is implemented in C and is generally faster than a manually implemented loop in Python, especially for large lists.
- **Readability:** Using built-in functions makes the code cleaner and more understandable for others who may read your code.

This optimized version is efficient, handles edge cases, and is easier to read and maintain.

Task 3

Problem Statement:

Create a Python function to eliminate duplicates from a list of ten country names, sorting the result alphabetically. Provide an example input list containing duplicates to showcase the function's effectiveness.

Explanation:

- Develop a Python function to eliminate duplicates and sort a list of 10 country names.
- Python is explicitly chosen as the programming language.

▼ Click here to view the sample response generated

Sure! Below is a step-by-step guide to creating a Python function that eliminates duplicates from a list of country names and sorts the result alphabetically. I'll also provide an example input list containing duplicates to demonstrate the function's effectiveness.

Step 1: Define the Function

We will create a function named `remove_duplicates_and_sort` that takes a list of country names as input.

Step 2: Use a Set to Remove Duplicates

Sets in Python automatically discard duplicate values, so we can convert the list to a set to eliminate duplicates.

Step 3: Sort the Result

After removing duplicates, we will convert the set back to a list and sort it alphabetically using the `sorted()` function.

Step 4: Return the Result

Finally, the function will return the sorted list of unique country names.

Complete Code Example

Here's the complete implementation of the function along with an example input list:

```
def remove_duplicates_and_sort(countries):  
    # Step 2: Use a set to eliminate duplicates  
    unique_countries = set(countries)
```



```
# Step 3: Sort the result alphabetically
sorted_countries = sorted(unique_countries)

# Step 4: Return the sorted list
return sorted_countries

Example input list containing duplicates
country_list = [
    "Canada", "Brazil", "Germany", "Canada", "France",
    "Brazil", "Australia", "Germany", "India", "Japan",
    "Australia"
]
Call the function and print the result
result = remove_duplicates_and_sort(country_list)
print("Sorted unique country names:", result)
```

Explanation of the Code

1. **Function Definition:** The function `remove_duplicates_and_sort` accepts a list of country names.
2. **Removing Duplicates:** The list is converted to a set called `unique_countries`, which automatically removes any duplicates.
3. **Sorting:** The `sorted()` function is used to sort the set of unique countries, and the result is stored in `sorted_countries`.
4. **Returning the Result:** The sorted list of unique country names is returned.

Example Output

When you run the above code with the provided `country_list`, the output will look like this:

```
Sorted unique country names: ['Australia', 'Brazil', 'Canada', 'France', 'Germany', 'India', 'Japan']
```

Additional Considerations

- **Case Sensitivity:** The sorting is case-sensitive, so “australia” would come before “Brazil”. If you want case-insensitive sorting, you can modify the `sorted()` function like this:

```
sorted_countries = sorted(unique_countries, key=str.lower)
```

- **Performance:** The use of a set for duplicate removal and the `sorted()` function ensures that the function is both efficient and straightforward.

Feel free to modify the input list or the function as needed!

Task 4

Problem Statement:

Verify whether the provided code can be optimized:

```
def remove_duplicates(nums):  
    seen = set()  
    unique_nums = []
```

```
for num in nums:
    if num not in seen:
        seen.add(num)
        unique_nums.append(num)
return unique_nums
```

Explanation:

- Developing a Python function for eliminating duplicates in a list can be enhanced by emphasizing the improvement of code efficiency.
- Evaluate the code for potential optimizations, alternative data structures, and avoiding redundant operations.

▼ [Click here to view the sample response generated](#)

The provided code for removing duplicates from a list is already quite efficient and clear, with a time complexity of $O(n)$, where n is the number of elements in the input list. However, there are a few areas where we can optimize the code for readability and performance. Let's review the original code first:

Original Code

```
def remove_duplicates(nums):
    seen = set()
    unique_nums = []

    for num in nums:
        if num not in seen:
            seen.add(num)
            unique_nums.append(num)
    return unique_nums
```

Potential Improvements

1. **Using a List Comprehension:** You can use a list comprehension to make the code more concise and Pythonic.
2. **Eliminating the Intermediate List:** You can directly convert the set back to a list after removing duplicates, which simplifies the code.
3. **Maintaining Order:** If you want to maintain the order of the first occurrence of each element, the current approach is valid. However, using `dict.fromkeys()` can also achieve this in a more concise way.

Optimized Code Example

Here's an optimized version of the function that maintains the order of elements while removing duplicates:

```
def remove_duplicates(nums):  
    # Using dict.fromkeys to maintain order and remove duplicates  
    return list(dict.fromkeys(nums))  
  
Example usage  
numbers = [1, 2, 3, 2, 1, 4, 5, 3, 6, 5]  
unique_numbers = remove_duplicates(numbers)  
print("Unique numbers:", unique_numbers)
```

Explanation of the Optimized Code

1. **Using `dict.fromkeys()`:** This method creates a dictionary where the keys are the elements of the list. Since dictionaries cannot have duplicate keys, this effectively removes duplicates while preserving the order of their first occurrence.
2. **Converting Back to List:** The result is then converted back to a list using `list()`, which gives the final list of unique numbers.

Example Output

When you run the optimized code with the provided `numbers` list, the output will be:

```
Unique numbers: [1, 2, 3, 4, 5, 6]
```

Additional Considerations

- **Performance:** Both the original and optimized versions have a time complexity of $O(n)$, but the optimized version is more concise and leverages Python's built-in capabilities.
- **Readability:** The optimized version is easier to read and understand, which is beneficial for code maintenance and collaboration.

This optimized function effectively removes duplicates while maintaining the order of elements, making it a good choice for this task.

Task 5

Problem Statement:

Write a Python function using the 'append' function to identify and return a list of duplicate elements from a given list of

integers. Include a sample input list with duplicates to illustrate the function's operation.

Explanation:

- Create a Python function using 'append' to find and return duplicate elements in an integer list.

▼ Click here to view the sample response generated

Sure! Below is a step-by-step guide to creating a Python function that identifies and returns a list of duplicate elements from a given list of integers. We will use the `append` method to build the list of duplicates.

Step 1: Define the Function

We will create a function named `find_duplicates` that takes a list of integers as input.

Step 2: Use a Set to Track Seen Elements

We will use a set to keep track of elements we have already encountered as we iterate through the list.

Step 3: Identify Duplicates

For each element in the list, we will check if it is already in the set of seen elements. If it is, we will append it to a list of duplicates.

Step 4: Return the List of Duplicates

Finally, we will return the list of duplicates.

Complete Code Example

Here's the complete implementation of the function along with a sample input list:

```
def find_duplicates(nums):  
    seen = set() # To track seen elements  
    duplicates = [] # To store duplicate elements  
  
    for num in nums:  
        if num in seen:  
            if num not in duplicates: # Ensure we only append the duplicate once  
                duplicates.append(num) # Step 3: Append to duplicates list  
        else:  
            seen.add(num) # Step 2: Add to seen set  
  
    return duplicates # Step 4: Return the list of duplicates  
Sample input list containing duplicates
```

```
input_list = [1, 2, 3, 4, 5, 3, 2, 1, 6, 7, 8, 5]
Call the function and print the result
result = find_duplicates(input_list)
print("Duplicate elements:", result)
```

Explanation of the Code

1. **Function Definition:** The function `find_duplicates` accepts a list of integers as input.
2. **Tracking Seen Elements:** A set named `seen` is used to keep track of elements that have already been encountered.
3. **Identifying Duplicates:** As we iterate through the list, if an element is found in the `seen` set, it is appended to the `duplicates` list (only if it hasn't already been added).
4. **Returning Duplicates:** The function returns the list of duplicates.

Example Output

When you run the above code with the provided `input_list`, the output will be:

```
Duplicate elements: [3, 2, 1, 5]
```

Additional Considerations

- **Order of Duplicates:** The order of duplicates in the output list reflects the order of their first occurrence in the

original list.

- **Performance:** The time complexity of this function is $O(n)$, where n is the number of elements in the input list, making it efficient for this task.
- **Handling Large Lists:** This approach is suitable for large lists, as using a set for membership checking is on average $O(1)$.

Feel free to modify the input list or the function as needed!

Summary

This hands-on lab provides a comprehensive guide on effectively utilizing ChatGPTs for obtaining solutions to coding challenges. The introduction emphasizes the importance of ChatGPTs as valuable coding assistants and the benefits of streamlining the problem-solving process. The hands-on experience with ChatGPT helps users enhance their problem-solving capabilities and efficiency in coding.

Congratulations! You have leveraged generative AI and obtained solutions for various coding problems.

Author(s)

Manvi Gupta
Rajashree Patil

IBM Corporation. All rights reserved.