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

From the drop-down menu choose the model “GPT-5 Nano”

In the `Prompt Instructions` field, please enter the following prompt inorder 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.

```
Start with a clear explanation of the problem and goal.
Break down the solution into small, easy-to-understand steps or plan.
Provide a complete working Python code example using only basic concepts like lists, sets, loops, and built-in functions. Avoid advanced topics, input validation, or complex type annotations.
Run the code on a sample input and show the expected output.
Explain the code line-by-line or section-by-section in simple terms so that beginners can understand how and why the code works.
Keep the tone instructional, clear, and beginner-friendly throughout
```

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

Problem and goal
We want a simple Python program that finds the minimum and maximum values in a list of numbers. The goal is to implement this using basic Python constructs (lists, loops, and simple conditionals) without relying on advanced features.

Plan
Define a function that takes a list of numbers. Check that the list is not empty and raise an error if it is. Initialize both min and max with the first element. Iterate through the rest of the elements, updating min and max as needed. Return the min and max. Demonstrate with a sample input and print the results.
Complete Code Example
Here's a complete Python program that implements the above steps:

```
def find_min_max(nums):
    # Return the minimum and maximum values from the non-empty list nums.
    if not nums:
        raise ValueError('The list must not be empty.')
    # Initialize min and max with the first element
    min_val = max_val = nums[0]
    # Scan the remaining elements
    for num in nums[1:]:
        if num < min_val:
            min_val = num
        if num > max_val:
            max_val = num
    return min_val, max_val
# Example usage with a sample list
sample_list = [3, 1, 4, 1, 5, 9, 2, 6, -1, 7]
minimum, maximum = find_min_max(sample_list)
print('Minimum value:', minimum)
print('Maximum value:', maximum)
```

Sample Run
Input (sample list):

[3, 1, 4, 1, 5, 9, 2, 6, -1, 7]

Expected Output:
Minimum value: -1

Maximum value: 9

Code Walkthrough

The function find_min_max(nums) defines a function that accepts a list of numbers.

It first checks if the list is empty and raises an error if so it initializes min_val and max_val to the first element of the list.

It loops through the remaining elements and updates min_val and max_val whenever a smaller or larger value is found.

The function returns a tuple (min_val, max_val).

In the example usage, we define sample_list, call the function, and print the results.

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

We will evaluate the provided function for finding the largest element in a list, identify potential issues, and provide a clean, beginner-friendly optimized version using basic Python concepts.

Problem and goal
Goal: Given a list of numbers, return the largest value using simple Python constructs (no advanced features). We also want to ensure the function behaves safely for edge cases (e.g., empty lists).

Plan

1. Check that the input list is not empty and raise a clear error if it is.
2. Initialize the current largest value with the first element.
3. Iterate over the remaining elements (start from the second element) and update the largest value when a bigger element is found.
4. Return the largest value. Include a sample usage to illustrate the result.

Issue with the Provided Code

The original function has a few potential problems:

- It accesses `arr[0]` before checking if the list is empty, which will raise an `IndexError` for an empty list.
- It iterates over the entire list including the first element, which is unnecessary since the first element is already used to initialize \`largest\`. This is a minor inefficiency.
- It uses tabs for indentation in the example, which can lead to inconsistent formatting across editors.
- It could be made more explicit with a clear error message for empty input.

Optimized Python Solution

Here\`s a simple, robust version that uses basic concepts and handles the empty-list case gracefully:

```python
def find_largest(arr):
    """Return the largest value in a non-empty list arr.
    Args:
        arr (list): A non-empty list of numbers.
    Returns:
        The largest number in arr.
    Raises:
        ValueError: If arr is empty.
    """
    if not arr:
        raise ValueError('The list must not be empty.')
    # Start with the first element as the current largest
    largest = arr[0]
    # Check remaining elements (start from index 1)
    for num in arr[1:]:
        if num > largest:
            largest = num
    return largest
# Example usage
sample = [3, 1, 4, 1, 5, -2, 9]
print('Largest value:', find_largest(sample))
```

Sample Run

Input (sample list):
[3, 1, 4, 1, 5, -2, 9]

Expected Output:

Largest value: 9

Code Walkthrough (Line-by-Line)

Function definition: def find_largest(arr): creates a function named find_largest that accepts one parameter, a list arr.

Docstring: A multi-line string at the start explains what the function does, its arguments, return value, and possible exceptions.

Empty check: if not arr: raise ValueError(`The list must not be empty.`) ensures we don’t crash on an empty input and provides a clear error message.

Initialization: largest = arr[0] initializes the current largest value with the first element of the list.

Loop over the rest: for num in arr[1:]: starts from the second element and goes through the end. This avoids unnecessary comparison with the first element.

Comparison: if num > largest: largest = num updates the largest value when a bigger number is found.

Return: return largest sends the found maximum back to the caller.

Example usage: sample = [3, 1, 4, 1, 5, -2, 9] and print(‘Largest value:’, find_largest(sample)) demonstrate how to call the function and display the result.

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 along with explanation

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

Problem and goal:

Create a Python function that takes a list of country names, removes duplicates, and returns the list sorted alphabetically.
Plan:

Define unique_sorted_countries(countries).
Use a set to drop duplicates, then sort the unique items with sorted().
Return the resulting list.
Provide an example input with duplicates and show the output.
Notes:

This solution uses basic concepts: lists, sets, and the built-in sorted() function.
The example input contains ten country names with duplicates to demonstrate effectiveness.

```
def unique_sorted_countries(countries):
    """Return alphabetically sorted list of unique country names.
    This function removes duplicates from the input list and returns
    the result sorted in ascending alphabetical order.
    It uses only basic Python features: lists, sets, and the built-in sorted().
    """
    if not isinstance(countries, list):
        raise TypeError('Input must be a list of country names.')
    # Remove duplicates by converting to a set, then sort the unique items.
    unique_sorted = sorted(set(countries))
    return unique_sorted
# Example usage with ten country names (with duplicates)
countries = [
    'Canada', 'Brazil', 'India', 'United States', 'Canada',
    'Mexico', 'Japan', 'Germany', 'France', 'India'
]
print('Input list (with duplicates):', countries)
result = unique_sorted_countries(countries)
print('Unique, alphabetically sorted:', result)
```

Sample run:
Input list (with duplicates): [‘Canada’, ‘Brazil’, ‘India’, ‘United States’, ‘Canada’, ‘Mexico’, ‘Japan’, ‘Germany’, ‘France’, ‘India’]

Output (duplicates removed, sorted): [‘Brazil’, ‘Canada’, ‘France’, ‘Germany’, ‘India’, ‘Japan’, ‘Mexico’, ‘United States’]

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

Goal: Verify whether the provided Python function can be optimized and provide beginner-friendly, working alternatives that remove duplicates from a list while preserving the order of first appearances. We'll discuss trade-offs, present multiple implementations (with code), and show a sample run.

Optimized implementations

Preserve Order (Returns a New List)

```
def remove_duplicates_preserve_order(nums):
    """Return a new list with duplicates removed, preserving first occurrence order.
    Uses dict.fromkeys to preserve order (requires Python 3.7+).
    """
    # dict.fromkeys creates keys in the order seen; duplicates are collapsed
    return list(dict.fromkeys(nums))
```

In-place variant (mutates input to remove duplicates, preserves order)

```
def remove_duplicates_inplace(nums):
    """Mutates nums to remove duplicates in-place while preserving order.
    Returns the mutated list shortened to the number of unique elements.
    This avoids creating a separate list for the result.
    """
    seen = set()
    write = 0
    for i in range(len(nums)):
        if nums[i] not in seen:
            seen.add(nums[i])
            nums[write] = nums[i]
            write += 1
    # Shorten the original list to the new length
    del nums[write:]
    return nums  # nums is mutated and shortened in-place
```

When the input is known to be sorted (two-pointer technique)

```
def remove_duplicates_sorted(nums):
    """If nums is sorted, remove duplicates in-place using the two-pointer method.
    Returns the mutated list shortened to unique elements.
    """
    if not nums:
        return []
    write = 1
    for i in range(1, len(nums)):
        if nums[i] != nums[i - 1]:
            nums[write] = nums[i]
            write += 1
    del nums[write:]
    return nums
```

Sample Run (Demonstration)

Let`s try each approach on a sample list and verify the results.

---

1. Preserve Order (Returns a New List)

```
sample = [1, 2, 2, 3, 1, 4, 3]
print(remove_duplicates_preserve_order(sample))
```

Expected Output:
[1, 2, 3, 4]

2. In-Place Modification (Modifies Original List)

```
nums = [1, 2, 2, 3, 1, 4, 3]
remove_duplicates_inplace(nums)
print(nums)
```

Expected Output:

[1, 2, 3, 4]
3. Two-Pointer Approach (For Sorted Lists Only)

```
sorted_nums = [1, 1, 2, 2, 3, 4, 4]
remove_duplicates_sorted(sorted_nums)
print(sorted_nums)
```

Expected Output:

[1, 2, 3, 4]

Line-by-line explanation (focus on the first approach)
Here is a line-by-line explanation of the first, concise approach using dict.fromkeys:

```
def remove_duplicates_preserve_order(nums):
    # 1) Define the function and its parameter (nums): the input list
    """Return a new list with duplicates removed, preserving first occurrence order.
    Uses dict.fromkeys to preserve order (requires Python 3.7+).
    """
    # 2) Create a new list from the keys of a dict constructed from nums
    #    dict.fromkeys(nums) creates a dict where each item in nums becomes a key.
    #    Duplicates are automatically collapsed because dict keys are unique.
    # 3) Convert the dict keys back to a list and return it
    return list(dict.fromkeys(nums))
```

- Define the function with a descriptive name.
- Docstring explaining purpose and a note about Python version requirement.
  dict.fromkeys(nums) creates a dict with each element of nums as a key, removing duplicates because dict keys are unique while preserving insertion order (in Python 3.7+).
- Convert the dict keys back into a list to return the result.
- This version is short, readable, and preserves the order of first appearances without needing an explicit loop or a separate result list

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

Problem: Given a list of integers, identify which elements appear more than once and return them as a new list. Each duplicate should appear only once in the output, and in the order of their first duplicate appearance.

Plan:

Use a set 'seen' to track numbers we've seen at least once.
Use a set 'duplicates' to track numbers we’ve already recorded as duplicates (to avoid re-adding).
Use a list 'result' and append duplicates to it in the order they are first identified.

Algorithm steps:

Iterate through nums.
If a number has been seen before and is not yet recorded as a duplicate, add it to 'duplicates' and append it to 'result'.
If a number hasn't been seen, add it to 'seen'.
Return the 'result' list.

Why this works:

Each element is touched once (O(n) time).
We avoid duplicates in the output by tracking already-identified duplicates in a set.
We preserve the order of first duplicate discovery by appending to ‘result’ when the duplicate is first detected.
Space/time:

Time complexity: O(n)

Space complexity: O(k), where k is the number of distinct duplicates (plus the 'seen' and 'duplicates' sets).
Sample run demonstrates the function on a list with duplicates and shows the expected output [2, 3, 4, 5].

```
def find_duplicate_elements(nums):
    """Return a list of duplicate elements from nums, preserving the order
    of their first duplicate appearance. Each duplicate element appears only once
    in the result.
    This implementation uses append to build the result list as required.
    """
    seen = set()
    duplicates = set()
    result = []  # Will store the duplicates in the order they are first identified
    for num in nums:
        if num in seen:
            if num not in duplicates:
                duplicates.add(num)
                result.append(num)  # Using append as requested
        else:
            seen.add(num)
    return result
# Sample input with duplicates to illustrate the function's operation
sample = [1, 2, 2, 3, 3, 3, 4, 5, 4, 6, 5]
print("Input:", sample)
print("Duplicates:", find_duplicate_elements(sample))
# Expected output: [2, 3, 4, 5]
```

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.