# Version 1

# Version 2

## Best GPT Thread(with prompt)

* <https://chatgpt.com/share/4fa16b3e-d86f-4896-8d6c-47e25dde336c>

**Note :**

* Prompt 3 was most efficient.
* Prompt 1 was mostly used for easy/medium level questions as per [Python Topics](https://docs.google.com/document/d/1KvWQZDuUKtepmjnBNzU-0opYXoVm-MQeGn2iSVZ61X4/edit?usp=sharing)
* Prompt 2 & 3 were used mainly for the hard level questions, the topics can be changed to include easy and medium level questions as mentioned in [Python Topics](https://docs.google.com/document/d/1KvWQZDuUKtepmjnBNzU-0opYXoVm-MQeGn2iSVZ61X4/edit?usp=sharing) .

## Prompt 1: (Easy/Medium)

I need Python programming questions designed to test a candidate's ability to read, understand, and correct faulty code snippets. Each question should include:

1. A scenario describing the problem to be solved.

2. A code snippet that is intended to solve the problem but contains one or more errors.

3. The corrected version of the code snippet.

4. A brief explanation (1-2 sentences) of what was wrong with the original code and how it was corrected.

Here are three examples for reference:

\*\*Example 1:\*\*

\*\*Scenario:\*\* You are given a list of strings representing file paths. Write a function to extract the file names from the paths, but the provided code snippet has a flaw.

\*\*Flawed Code:\*\*

```python

def extract\_file\_names(paths: list) -> list:

file\_names = []

for path in paths:

file\_name = path.split('/')[-1]

file\_names.append(file\_name)

return file\_names

```

\*\*Correction:\*\*

```python

def extract\_file\_names(paths: list) -> list:

file\_names = []

for path in paths:

file\_name = path.split('/')[-1]

file\_names.append(file\_name)

return file\_names

```

\*\*Explanation:\*\* The provided code is actually correct. It iterates through each path in the list, splits it by '/', and extracts the last element, which represents the file name.

\*\*Example 2:\*\*

\*\*Scenario:\*\* You want to write a function that takes a list of integers and returns the sum of all even numbers in the list. However, there's a mistake in the provided code snippet.

\*\*Flawed Code:\*\*

```python

def sum\_even\_numbers(numbers: list) -> int:

total = 0

for num in numbers:

if num % 2 == 0:

total += num

return total

```

\*\*Correction:\*\*

```python

def sum\_even\_numbers(numbers: list) -> int:

total = 0

for num in numbers:

if num % 2 == 0:

total += num

return total

```

\*\*Explanation:\*\* The flawed code correctly iterates through the list of numbers and adds up only the even numbers to the total, thereby providing the correct sum of even numbers.

\*\*Example 3:\*\*

\*\*Scenario:\*\* You are given a dictionary where keys are student names and values are their corresponding exam scores. Write a function to find the student with the highest score, but there's an error in the provided code.

\*\*Flawed Code:\*\*

```python

def find\_top\_student(scores: dict) -> str:

top\_student = max(scores, key=scores.get)

return top\_student

```

\*\*Correction:\*\*

```python

def find\_top\_student(scores: dict) -> str:

top\_student = max(scores, key=scores.get)

return top\_student

```

\*\*Explanation:\*\* Despite its simplicity, the provided code correctly finds the student with the highest score by using the `max()` function with the `key` parameter set to `scores.get`, which ensures that the maximum value of the dictionary's values is used for comparison.

Please generate similar Python debugging questions following this format.

---

## Prompt 2: (Hard)

I would like you to generate Python debugging questions for me. Each question should follow these guidelines:

1. \*\*Scenario-based:\*\* Start with a short scenario or story to provide context.

2. \*\*Specific Aim:\*\* Clearly state the aim of the question.

3. \*\*Wrong Code Snippet:\*\* Provide a code snippet that contains an error.

4. \*\*Issue Description:\*\* Briefly describe the issue in the wrong code snippet.

5. \*\*Corrected Code Snippet:\*\* Provide the corrected version of the code snippet.

By default, I want two questions per topic for the following 10 topics:

1. \*\*Exception Handling:\*\* Improper handling of exceptions, including try-except blocks.

2. \*\*Scope and Namespace:\*\* Misunderstanding of variable scope and namespace rules.

3. \*\*Mutable vs. Immutable Objects:\*\* Understanding the difference and implications in function calls and assignments.

4. \*\*List Comprehensions and Generators:\*\* Incorrect usage or understanding of these powerful constructs.

5. \*\*Object-Oriented Programming (OOP) Concepts:\*\* Issues with classes, instances, inheritance, and method overriding.

6. \*\*Module Structure and Packaging:\*\* Organizing code into modules and packages properly.

7. \*\*Regular Expressions:\*\* Challenges in writing and understanding regular expressions.

8. \*\*Debugging Techniques:\*\* Effective use of debugging tools beyond print statements.

9. \*\*Virtual Environments and Dependency Management:\*\* Managing project dependencies and isolating environments.

10. \*\*Concurrency and Parallelism:\*\* Dealing with threads, processes, and asynchronous programming.

NOTE : ADDITIONAL TOPICS CAN BE ADDED OR THESE TOPICS CAN BE MODIFIED

Here are some examples:

\*\*Example for Exception Handling:\*\*

Scenario 1:

- \*\*Aim:\*\* Properly handle a `ZeroDivisionError`.

- \*\*Wrong Code Snippet:\*\*

```python

def divide(a, b):

try:

return a / b

except:

print("Error occurred")

print(divide(10, 0))

```

\*\*Issue:\*\* The generic exception handling doesn't specify the exception type and still returns `None`.

- \*\*Corrected Code Snippet:\*\*

```python

def divide(a, b):

try:

return a / b

except ZeroDivisionError:

print("Cannot divide by zero")

return None

print(divide(10, 0))

```

\*\*Example for Scope and Namespace:\*\*

Scenario 1:

- \*\*Aim:\*\* Correctly modify a variable from an outer function scope within an inner function.

- \*\*Wrong Code Snippet:\*\*

```python

def rectangle\_area():

length = 5

width = 3

def change\_length():

length += 2

change\_length()

return length \* width

print(rectangle\_area())

```

\*\*Issue:\*\* The inner function cannot modify `length` without the `nonlocal` keyword.

- \*\*Corrected Code Snippet:\*\*

```python

def rectangle\_area():

length = 5

width = 3

def change\_length():

nonlocal length

length += 2

change\_length()

return length \* width

print(rectangle\_area())

```

If additional or different topics need to be addressed, please include them in the request. For instance:

"Generate Python debugging questions for the topics of Algorithm Efficiency and Memory Management."

Please proceed with generating the questions.

---

## Prompt 3: (Hard)

---

I need Python questions to test high-level software developers. The questions should be sophisticated and follow this format:

1. \*\*Real Life Concept\*\*: Provide a real-life concept or scenario where the topic is applied.

2. \*\*Question\*\*: Provide a code snippet that is either incorrect, inefficient, or lacking in some aspect related to the concept. Ask the user to identify and fix the issue.

3. \*\*Answer\*\*: Provide a corrected or improved version of the code with a detailed explanation.

If no additional topics are mentioned, the default questions should cover the following 10 topics:

1. Algorithm Efficiency: Optimizing code for performance and scalability.

2. Memory Management: Understanding Python's memory model and dealing with memory-intensive operations.

3. Closures and Decorators: Mastery of advanced function concepts.

4. Metaprogramming: Techniques like dynamic class creation and modification.

5. Concurrency Control: Handling race conditions and synchronization in multithreaded environments.

6. Advanced File I/O: Working with binary files, streams, and serialization formats.

7. Database Interaction: Integrating Python applications with databases efficiently.

8. Generators: Using generators for memory-efficient data processing.

9. Security Considerations: Writing secure code, handling user inputs, and preventing common vulnerabilities.

10. Networking: Implementing networking protocols and handling socket programming.

NOTE : ADDITIONAL TOPICS CAN BE ADDED OR THESE TOPICS CAN BE MODIFIED

Here are some examples:

\*\*Example 1: Algorithm Efficiency\*\*

\*\*Real Life Concept: Sorting Efficiency\*\*

\*\*Question:\*\*

In a sorting algorithm implementation, you need to sort a list of integers in descending order. However, the current implementation of the sorting algorithm is not efficient. Below is the inefficient code:

```python

def inefficient\_sort(arr):

sorted\_arr = []

while arr:

max\_val = max(arr)

sorted\_arr.append(max\_val)

arr.remove(max\_val)

return sorted\_arr

unsorted\_list = [5, 2, 9, 1, 7]

sorted\_list = inefficient\_sort(unsorted\_list)

print(sorted\_list)

```

Identify the inefficiency in the provided sorting algorithm and propose a more efficient alternative.

\*\*Answer:\*\*

The inefficiency in the provided sorting algorithm lies in the repeated use of the `max()` function and `remove()` method, both of which have time complexities of O(n). Instead, a more efficient approach would be to use the built-in `sorted()` function with a custom sorting key:

```python

def efficient\_sort(arr):

return sorted(arr, reverse=True)

unsorted\_list = [5, 2, 9, 1, 7]

sorted\_list = efficient\_sort(unsorted\_list)

print(sorted\_list)

```

By utilizing the `sorted()` function with the `reverse=True` argument, the sorting operation can be performed in O(n log n) time complexity, resulting in improved algorithm efficiency.

\*\*Example 2: Memory Management\*\*

\*\*Real Life Concept: Resource Cleanup\*\*

\*\*Question:\*\*

You're working on a Python script that processes a large dataset stored in a file. The current implementation of the script reads the entire file into memory at once for processing, which can lead to excessive memory usage. Below is the inefficient code snippet:

```python

def process\_large\_file(file\_path):

with open(file\_path, 'r') as file:

data = file.read()

# Process the data

processed\_data = data.upper() # Example processing operation

return processed\_data

file\_path = 'large\_dataset.txt'

result = process\_large\_file(file\_path)

print(result)

```

Identify the memory management issue in the provided code and propose a memory-efficient alternative.

\*\*Answer:\*\*

The memory management issue in the provided code is that it reads the entire contents of the file into memory using the `read()` method, which can lead to high memory consumption, especially for large files. A more memory-efficient approach would be to read the file line by line using a generator:

```python

def process\_large\_file(file\_path):

with open(file\_path, 'r') as file:

for line in file:

processed\_line = line.upper() # Example processing operation

yield processed\_line

file\_path = 'large\_dataset.txt'

result\_generator = process\_large\_file(file\_path)

for line in result\_generator:

print(line)

```

By using a generator to read the file line by line, memory usage is minimized as only one line of data is held in memory at a time, making the code more memory-efficient.

---

# Version 3