Examples of a good prompt 🤓

Code Generation:

"**Generate Python code for a program that simulates a basic banking system.** The program should allow users to create new accounts, deposit and withdraw funds, and check their account balance. Ensure the code includes error handling for invalid inputs (e.g., negative deposit amounts) and uses appropriate data structures to store account information. The code should be well-documented with comments explaining the functionality of each part."

"**Implement a graph traversal algorithm in Java that identifies the shortest path between two nodes in a weighted, directed acyclic graph**. The graph can be represented using an adjacency list. The algorithm should have a time complexity of O(V+E), where V is the number of vertices and E is the number of edges. Include a main method that demonstrates the usage of the algorithm with a sample graph and two nodes."

Problem Reflection:

"**Analyze the challenges of implementing a concurrent system with multiple threads accessing shared resources**. Discuss potential issues like race conditions and deadlocks. Explore different synchronization mechanisms (e.g., mutexes, semaphores) and their trade-offs in terms of complexity and performance. How can these mechanisms be used to ensure data integrity and prevent concurrency-related bugs?"

"**Consider the problem of designing an efficient caching system for a web application.** What are the key factors to consider when choosing a caching strategy (e.g., eviction policies, cache size, consistency)? Discuss the advantages and disadvantages of different caching techniques, such as LRU (Least Recently Used), FIFO (First-In, First-Out), and LFU (Least Frequently Used). How can you measure the effectiveness of a caching system?"

Tests Reasoning:

"You are given a function that implements a binary search algorithm on a sorted array. The following test cases are provided:

- Input array: [2, 5, 8, 12, 16, 23, 38, 56, 72, 91], Target value: 23, Output: 5 (index of the target value)
- Input array: [2, 5, 8, 12, 16, 23, 38, 56, 72, 91], Target value: 4, Output: -1 (target value not found)
- Input array: [1], Target value: 1, Output: 0

Explain why the binary search algorithm produces these outputs for each of the given test cases. How does the algorithm handle cases where the target value is not present in the array? What is the time complexity of binary search and why is it efficient for searching sorted data?"

"Consider a machine learning model trained to classify images of cars and bikes. You are given the following test images and the model's predictions:

- Image 1: A clear picture of a car, Prediction: car (correct)
- Image 2: A blurry picture of a bike, Prediction: bike (correct)
- Image 3: A picture of a car surrounded by bikes, Prediction: bike (incorrect)

Analyze the model's predictions and explain why it might have misclassified the image of the car with a bike. What factors could be influencing the model's decision? How can you improve the model's accuracy in such cases?"

Code Refactoring:

Example 1++++++++++++++++++
"Refactor the following Java code that calculates the area of different geometric shapes. The code currently uses a long series of if-else statements to determine the shape type. Improve the code by using polymorphism and abstract classes or interfaces to represent different shapes. Ensure the refactored code is more maintainable, extensible, and adheres to object-oriented design principles."

```
// Existing code to be refactored
public class AreaCalculator {
    public double calculateArea(String shapeType, double… dimensions) {
        if (shapeType.equals(“circle”))
            // Calculate area of circle
        } else if (shapeType.equals(“rectangle”)) {
            // Calculate area of rectangle
        } else if (shapeType.equals(“triangle”)) {
            // Calculate area of triangle
        } // … more else-if statements for other shapes
        return 0;
    }
}
```

Example 2+++++++++++++++++
“You are given Python code that implements a simple web server using sockets. The code works correctly but lacks proper error handling and logging. Refactor the code to include robust error handling for potential exceptions, such as network errors and invalid requests. Add logging statements to record important events, like client connections, requests, and errors. Ensure the refactored code is more reliable and easier to debug.”

```
def longest_common_subsequence(str1, str2):
  “”"
  Finds the longest common subsequence of two strings (BUGGY!).
  Args:
    str1: The first string.
```

```
  str2: The second string.
 Returns:
  The longest common subsequence as a string.
 “”"
 m = len(str1)
 n = len(str2)
 # Initialize a 2D array INCORRECTLY (should be (m+1) x (n+1))
 dp = [[0] * n for _ in range(m)]
 for i in range(1, m + 1):
  for j in range(1, n + 1):
   if str1[i-1] == str2[j-1]:
    dp[i][j] = dp[i-1][j-1] + 1  # IndexError due to incorrect initialization
   else:
    dp[i][j] = max(dp[i-1][j], dp[i][j-1])  # IndexError here as well
 # Backtracking with the same bug as before
 i = m
 j = n
 lcs = “”
 while i > 0 and j > 0:
  if str1[i-1] == str2[j-1]:
   lcs = str1[i-1] + lcs
   i -= 1
   j -= 1
  else:
   if dp[i-1][j] > dp[i][j-1]:  # Incorrect comparison
    i -= 1
   else:
    j -= 1
 return lcs
```

Bug Fixing:

Example 1 +++++++++++++++

"A Java program that implements a multithreaded chat application is experiencing intermittent crashes. The crashes seem to occur randomly when multiple users are sending messages simultaneously. Analyze the code and identify the potential race conditions or synchronization issues that might be causing these crashes. Provide a corrected version of the code that addresses these concurrency problems and ensures thread safety."

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.ArrayList;
import java.util.List;
public class ChatServer {
  private List<PrintWriter> clientWriters = new ArrayList<>();
  public void start(int port) throws IOException {
    ServerSocket serverSocket = new ServerSocket(port);
    System.out.println("Chat server started on port " + port);
    while (true) {
      Socket clientSocket = serverSocket.accept();
      System.out.println("New client connected");
      // BUG: No synchronization when accessing shared clientWriters list
      clientWriters.add(new PrintWriter(clientSocket.getOutputStream(), true));
```

```
      Thread clientThread = new Thread(() -> {
        try {
          BufferedReader reader = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
          String message;
          while ((message = reader.readLine()) != null) {
            // BUG: No synchronization when accessing shared clientWriters list
            for (PrintWriter writer : clientWriters) {
              writer.println(message);
            }
          }
        } catch (IOException e) {
          System.err.println("Error handling client: " + e.getMessage());
        }
      });
      clientThread.start();
    }
  }
  public static void main(String[] args) throws IOException {
    ChatServer server = new ChatServer();
    server.start(8080);
  }
}
```

Example 2++++++++++++++++++

"This Python code is supposed to find the longest common subsequence of two strings, but it's returning incorrect results for some inputs. Debug the code and identify the source of the error. Provide a corrected version of the code that produces the correct output for all valid input strings. Explain the bug you fixed and how your correction addresses the issue."

```
def longest_common_subsequence(str1, str2):
 """
 Finds the longest common subsequence of two strings (BUGGY!).
 Args:
  str1: The first string.
  str2: The second string.
 Returns:
  The longest common subsequence as a string.
 “”"
 m = len(str1)
 n = len(str2)
 # Initialize a 2D array with INCORRECT dimensions
 dp = [[0] * n for _ in range(m)]
 for i in range(m): # Looping up to m instead of m+1
  for j in range(n): # Looping up to n instead of n+1
   if str1[i] == str2[j]: # Comparing characters at incorrect indices
    dp[i][j] = dp[i-1][j-1] + 1 # Potential IndexError due to incorrect loop bounds
   else:
    dp[i][j] = max(dp[i-1][j], dp[i][j-1]) # Potential IndexError here as well
 # Backtracking with incorrect indices and logic
 i = m - 1
 j = n - 1
 lcs = “”
 while i >= 0 and j >= 0: # Incorrect loop condition
  if str1[i] == str2[j]: # Comparing characters at incorrect indices
   lcs = str1[i] + lcs
   i -= 1
```

```
   j -= 1
  else:
   if dp[i-1][j] > dp[i][j-1]: # Incorrect comparison and potential IndexError
    i -= 1
   else:
    j -= 1
 return lcs
```

Test Generation:

"**Write a comprehensive set of unit tests in Python for a function that validates email addresses.** The function should check for the presence of the @ symbol, a valid domain name, and optional top-level domain. The tests should cover various scenarios, including valid email formats, invalid formats (missing @, invalid characters, etc.), and edge cases (e.g., very long email addresses, unusual domain names). Aim for high test coverage to ensure the email validation function works reliably."

"**Generate test cases for a C++ program that controls a robotic arm in a manufacturing environment**. The program receives commands to move the arm to specific coordinates and perform actions like gripping and releasing objects. The test cases should cover a range of scenarios, including valid movement commands, invalid commands (e.g., out-of-bounds coordinates), and sequences of commands that test the robot's ability to perform complex tasks. Consider edge cases like obstacle avoidance and error handling for unexpected situations."

Solutions Reasoning:

"You are given a JavaScript implementation of a sorting algorithm called Merge Sort. Explain step-by-step how this algorithm works to sort an array of numbers. Analyze its time and space complexity in comparison to other sorting algorithms, such as Bubble Sort and Quick Sort. Discuss the advantages and disadvantages of Merge Sort, including its suitability for different types of data and use cases."

```
function mergeSort(arr) {
 if (arr.length <= 1) {
  return arr;
 }
 const mid = Math.floor(arr.length / 2);
 const left = arr.slice(0, mid);
 const right = arr.slice(mid);
 return merge(mergeSort(left), mergeSort(right));
}
function merge(left, right) {
 let result = [];
 let leftIndex = 0;
 let rightIndex = 0;
 while (leftIndex < left.length && rightIndex < right.length) {
  if (left[leftIndex] < right[rightIndex]) {
   result.push(left[leftIndex]);
   leftIndex++;
```

```
  } else {
   result.push(right[rightIndex]);
   rightIndex++;
  }
 }
 return result.concat(left.slice(leftIndex)).concat(right.slice(rightIndex));
}
// Example usage:
const unsortedArray = [5, 2, 4, 6, 1, 3];
const sortedArray = mergeSort(unsortedArray);
console.log(sortedArray); // Output: [1, 2, 3, 4, 5, 6]
```

Example 2 ++++++++++++++++++++

“Analyze the following C# code that implements a data structure called a Trie. Explain how this data structure efficiently stores and retrieves a large collection of strings. Discuss its use cases, such as autocomplete functionality and spell checking. Compare the performance of a Trie with other data structures, like hash tables and balanced trees, for storing and searching strings. What are the strengths and weaknesses of using a Trie in different scenarios?”

```
public class TrieNode
{
  public char Value { get; set; }
  public bool IsEndOfWord { get; set; }
  public Dictionary<char, TrieNode> Children { get; set; }
  public TrieNode(char value)
  {
    Value = value;
    IsEndOfWord = false;
    Children = new Dictionary<char, TrieNode>();
  }
}
public class Trie
{
  private TrieNode root;
  public Trie()
  {
    root = new TrieNode('\0'); // Root node with null character
  }
  public void Insert(string word)
  {
    TrieNode currentNode = root;
    foreach (char c in word)
    {
      if (!currentNode.Children.ContainsKey©)
      {
        currentNode.Children.Add(c, new TrieNode©);
      }
      currentNode = currentNode.Children[c];
    }
    currentNode.IsEndOfWord = true;
  }
  public bool Search(string word)
  {
    TrieNode currentNode = root;
    foreach (char c in word)
    {
```

```
      if (!currentNode.Children.ContainsKey©)
      {
        return false;
      }
      currentNode = currentNode.Children[c];
    }
    return currentNode.IsEndOfWord;
  }
}
// Example usage:
Trie trie = new Trie();
trie.Insert(“apple”);
trie.Insert(“app”);
trie.Insert(“banana”);
Console.WriteLine(trie.Search(“apple”)); // Output: True
Console.WriteLine(trie.Search(“app”));   // Output: True
Console.WriteLine(trie.Search(“ban”));   // Output: False
```