**Task Submission**

****

**Python Screening Task 2: Write a Prompt for an AI Debugging Assistant**

This task evaluates your ability to craft a clear, effective prompt that guides an AI language model to help a student debug their Python code—without revealing the correct solution.

**Task Objective:**

Write a natural-language prompt that will be given to an AI assistant (like ChatGPT). The AI will use your prompt to:

* Analyze a student’s buggy Python code
* Offer helpful suggestions or hints
* Avoid giving away the correct solution

Your prompt should be general enough to work across different coding questions, but specific enough to guide the AI toward constructive, non-revealing feedback.

### **Prompt Description**

This prompt is carefully designed to ensure that any AI clearly understands its role as a Python debugging mentor.

It follows three core principles:

1. **Question-first guidance** – Encourages active learning by questioning the student to think critically before receiving direction, while preventing the AI from simply giving away the full solution.
2. **Concise, evidence-based hints** – It gives step-by-step minimal formatted instructions that mirror effective debugging strategies and established feedback rubrics.
3. **Student-centered feedback** – Ensures responses are non-revealing and focused on guiding learners toward discovery, applicable across a wide range of coding challenges.

### **Wording Approach**

The wording has been structured to make the AI’s role clear, precise, and instructional. To achieve this, the prompt is divided into **four key sections**:

* **Goal Check** – Restates the problem in simple terms and confirms the learner’s understanding of the expected outcome.
* **Reproduce** – Encourages isolating the bug through controlled reproduction steps.
* **Hints** – Provides progressive, non-directive clues that help the learner identify and resolve the issue themselves.
* **Concept Check** – Reinforces the underlying programming concept (e.g., variables, floor vs. true division, while loops) to ensure long-term understanding beyond the immediate fix.

### **How I Ensured It Avoids Giving the Solution**

I designed the prompt with strict guidelines and constraints that prevent the AI from revealing the final fix. Instead of providing direct solutions or corrected code, the AI is required to:

* **Restate the goal** in simple terms to confirm understanding, not to rewrite the solution.
* **Suggest only one reproduction step** that exposes the issue without showing the fix.
* **Offer hints**, phrased as guiding questions or observations about regions of the code rather than explicit line edits.
* **Include a short concept refresher** only when necessary, so the learner builds long-term understanding instead of copying an answer.

### **How it Encourages Helpful, Student-Friendly Feedback**

The prompt is designed to create a supportive and effective learning environment by focusing on a Socratic method of teaching. It prevents the AI from simply giving the answer, forcing learners to actively engage in the debugging process. The structured format and provided guidelines ensures the feedback is consistent, concise, and easy to follow.

### **What tone and style should the AI use when responding?**

The AI's tone should be **encouraging** and **teacherly**, yet **brief**. It should prioritize **curiosity over authority**. This means the AI should avoid a condescending or overly formal tone. The goal is to act as a mentor, guiding the student with well-placed questions rather than telling them what to do. The language should be simple and direct, using a conversational style to make the feedback approachable. The use of questions like, "What happens when..." or "How does the loop handle..." prompts critical thinking without giving away the solution.

### 

### **How should the AI balance between identifying bugs and guiding the student?**

The prompt ensures a balance by requiring the AI to **identify the problem's root cause** but **prohibit providing the solution**. The AI's role is to act as an advanced debugging tool, pointing to the specific problems of the bug (e.g., a variable's value, a specific line, or a data type issue) without providing the solution.

The balance is achieved through the following structure:

* **Reproduce:** This section helps the student confirm the bug's existence with a simple test case. It isolates the problem, preventing the student from getting overwhelmed by complex inputs.
* **Hints:** The hints are the core of the guidance. They don't state "The bug is on line X," but instead ask questions like, "What value does <var\_name> have just before <line/branch>?" This forces the student to investigate the code and the state of their variables, which is a fundamental debugging skill. The hints are **evidence-based**, pointing to specific signals the student should be looking for.
* **Concept check:** This is a final, high-level guide. It addresses the "why" behind the bug without giving away the "how to fix it." It reinforces core programming concepts, which is vital for long-term learning.

### **How would you adapt this prompt for beginner vs. advanced learners?**

**For Beginner Learners:**

* **More explicit hints:** The hints could be more direct, pointing to a very specific line number or variable. Instead of a general question, you might ask, "What is the data type of n on **line 5**?"
* **Simpler conceptual refreshers:** The conceptual checks would be more detailed and use a more simplified language. For instance, explaining **integer vs. float division** with a very simple example (e.g., 5 / 2 = 2.5 vs. 5 // 2 = 2).
* **Focus on a single bug:** If the code has multiple issues, the AI should focus on the first and most critical one, not overwhelming the student with a list of problems.

**For Advanced Learners:**

* **More abstract hints:** The hints would be less prescriptive. Instead of "Check variable X," the hint might be a more conceptual question like, "Consider the **mutation vs. reassignment** within your function."
* **Less frequent conceptual refreshers:** A conceptual refresher would only be included for more complex or subtle issues, not for common misconceptions. The assumption is that they already understand basic concepts.
* **Encourage refactoring:** The hints could subtly guide the student to consider more efficient or idiomatic ways of solving the problem after the bug is fixed, pushing them beyond just finding the solution.
* **Mentioning multiple bugs:** If the code has several independent issues, the AI could point out the existence of other potential problems after the primary one is resolved, prompting the student to find them on their own.

**The Summary**

This task is about creating a prompt for an AI debugging assistant that helps students fix Python code without giving direct solutions. The assistant should restate the goal, suggest one reproduction step, give evidence-based hints as questions, and provide brief concept reminders. The tone must be encouraging and conversational, guiding students through discovery rather than authority. For beginners, hints should be more explicit and simple, while for advanced learners, guidance can be abstract, minimal, and refactoring-oriented.