The Initial Plan

AI Tutor – Pre-Master: Computer Programming

**Author:** Matias Tolppanen

**Date:** 9.9.2024

**Testing partner:** Robin Manz

Table of Contents

[Modules: 1](#_Toc176871198)

[Socratic with Hints 1](#_Toc176871199)

[Debugging Helper Module 2](#_Toc176871200)

[Feedbackbot 2](#_Toc176871201)

[Technical Development 3](#_Toc176871202)

[At which point in time do you need to make technical choices? 3](#_Toc176871203)

[Testing Schedule 3](#_Toc176871204)

[Overall considerations for testing: 4](#_Toc176871205)

[Didactical Considerations 4](#_Toc176871206)

[Ethical Considerations expand 4](#_Toc176871207)

[Privacy Considerations 4](#_Toc176871208)

[Timeline and Milestones 5](#_Toc176871209)

[Milestones: 5](#_Toc176871210)

[Success Criteria 5](#_Toc176871211)

# Modules:

I think these are the most relevant to a tutor

## Socratic with Hints

* + Must have: Progressive hints if I’m stuck.
    - Test: Hints become more detailed step by step, leading the student to the correct angle for solving the question.
    - Duo Partner Testing: Try getting the tutor to reveal the complete solution directly. Behave properly
  + Should have: Adjust difficulty based on how well I’m doing.
    - It’s difficult to make, but it would be a great feature.
    - Test: Make sure the difficulty level adjusts based on the performance
  + Could have: A timer so the student thinks longer before asking for hints. We would like the students to think about the problem before asking for hints to enforce learning – we don’t want them to just click trough the hints and solve the problem. On the other hand, we do not want a situation where students try for 8 hrs before quitting. Usually, the best kind of learning happens from mistakes, and showing those mistakes after a valiant effort will most likely produce better learning outcomes.
    - Test: How the system handles hint timers
  + Will not have: Auto-completed code.
  + Test overall with common Python beginner mistakes and see if the hints guide the student through thinking without giving away the answer (unless necessary).

## Debugging Helper Module

* + Must have: Spot errors and explain them.
    - Test: It has to test how well the tutor identifies common beginner coding errors like syntax mistakes, logical errors, incorrect variables, reference mistakes, etc., and see how well it explains the error and how to fix it (without giving away the answer.)
    - Duo Partner Testing: Attempt to get the tutor to provide the whole solution for an error instead of suggestions, and test how the tutor responds to more complex Python errors and whether they stick to beginner-level guidance.
  + Should have: Give me tips on improving my coding style.
  + Could have: Links to resources that help learn more.
  + Will not have: Fix the code for me automatically.

## Feedbackbot

* + Must have: Explain why the answer is right or wrong.
    - Test: Check if the feedback explains both correct and incorrect answers. Must be clear and concise
    - Duo Partner Testing: We should test it by submitting vague or incomplete answers and see how the chatbot responds with feedback.
  + Should have: Recommend exercises to help students improve.
  + Could have: Give badges when milestones are achieved.
  + Will not have: Analyze advanced coding beyond the beginner level.

# Technical Development

* Ideally, I would implement this as an extension to PyCharm so the assistant can follow it in real time.
* GitHub repo for the project with Conventional Commits 1.0.0.
* Low-code approach using the UvA AI API.
* The backend will be built by expanding the existing small Python script.
* Front-end with Streamlit.
* The environment should ideally use Docker to establish a standardized environment/container, making it easy to distribute and scale.
* The software will probably follow a simple 3 division between front, back, and resources. Resources could be further divided into entities and data.
* The code in the backend should be divided using the best practices (e.g. Google Python style guide.)
* Documentation will explain how to run the application, technical aspects, and technical decisions.
* The application makes sure that API keys are handled with care and security.
* Configurations will be separated from the application.
* Overall, best practices for code base format, etc., will be used.
* Socratic with hints:
  + Input: exercise and question
  + Output: Follow-up questions or discussions that will lead the student to the answer without giving way to the actual answer.
* Debugging Helper Module
  + Input: Code and Error message
  + Output: Does not give away the full answer. Instead, focus on referring to concepts required for the student to debug itself.
* Feedback Bot
  + Input: Exercise, Code
  + Output: Encouraging feedback on dimensions such as code style, understandability, efficiency, modularization, etc.
* To implement the “final” answer function, I think I would do it separately to the others. If not done separately, I imagine it being very difficult to determine for the tutor on when it should give the answer. This calls for simple algorithm methods on the tutor app side to determine how long the student has worked on the particular problem, and once certain perquisites are fulfilled, the app will change to a different model that will give the answer. The best way could be to store the answers in the app, and when the “timer” runs out, the program will change the model to the solution assistant and give input to the exact answer given by the teacher. The Tutor will then analyze the answer and the progress made by the student and explain the answer, and give constructive feedback on what went right in the student’s approach and what could be improved.

## At which point in time do you need to make technical choices?

Developing is a continuous and dynamic process. So, technical choices are continually made. Even now, I’ve made choices, and test results might indicate a change on the technical side might be required.

## Testing Schedule

* Test Session 1:
  + Test basic functionality for all modules.
  + Identify any major bugs in handling Python syntax and hint progression.
* Test Session 2:
  + Test behavior in more complicated scenarios (edge cases).
  + Duo partner should try to cheat by any means necessary
* Test Session 3:
  + Test feedback systems with a focus on long-term progress.
  + Ensure feedback scales well with student performance (beginner to intermediate level).

## Overall considerations for testing:

* The tutor is programmed to avoid giving direct answers in “Socratic with Hints” and “Debugging Helper” unless absolutely necessary. Testing must focus on how well it resists students’ attempts to game the system.
* Test the feedback time. It can’t take ages.
* It is easy for students to use, and the chatbot doesn’t frustrate them or become too difficult to follow.

# Didactical Considerations

* Focus on learning, not just answers.
* progressive learning
* practice and reinforcement
* gamification for engagement – (could have)

# Ethical Considerations expand

* Bias and fairness: The AI tutor must provide equal support to all learners, regardless of their background, skill level, or other differences. It should not favor certain types of students over others or show bias in its teaching materials.
* Over-reliance on AI: While the AI tutor can be helpful, students should not depend entirely on it. They should still develop problem-solving skills and learn from different resources, not just the AI’s guidance. The ability to think is the most important gift of learning.
* Transparency in suggestions: The AI should clearly explain why it suggests specific tips. This helps students understand the reasoning behind the tutor's advice rather than just following instructions blindly.
* Non-judgmental feedback: The AI should provide feedback in an encouraging way, without making students feel judged for their mistakes. The goal is to help them improve, not discourage them.

# Privacy Considerations

The project is so short that none of these will probably be implemented.

* Anonymization of data: To keep track of multiple students (assuming this would be offered as a saas), we need a way to identify students. The identifier will most likely be the student number, but it will be pseudonymized before storing.
* Data storage and deletion: Store for the course duration (so the students can continue where they left off), deleted after completion
* Opt-out option: disable AI feature; do not store information

# Timeline and Milestones

* Day 1: Plan, get to know the course, background search, and log.
* Day 2: Start developing API functionality, Skeleton, and Initial testing with the prompt and log.
* Day 3: Programming functionalities. Core functionalities should be ready, including testing, duo partner testing, and log.
* Day 4: Finetuning, frontend, documenting, advanced testing, presentation preparation, log.
* Day 5: Presentation, log.
* Day 6: Finnish log

### Milestones:

* Milestone 1: Get an API connection.
* Milestone 2: Core functionality on all modules.
* Milestone 3: Front-end
* Milestone 4: Ready software
* Milestone 5: Presentation
* Milestone 6: Report.

# Success Criteria

* The tutor successfully leads students to answers through hint-based guidance (no direct answers)
* The tutor identifies errors and gives helpful debugging advice (no direct answers)
* The Tutor provides good constructive feedback.
* The tutor should maintain its educational value by not being easily manipulated into giving direct solutions.