Cicada Requirements

# V1.0 Cicada - Evidence identification and scoring

**1. Purpose**

* basic front-end application that interfaces with the OpenAI API to serve as an AI tutor.
* allow users to input queries, receive AI-generated responses, and maintain a chat history.
* provide a user-friendly interface that facilitates natural language interactions with the OpenAI model (e.g., GPT-4).
* Key features include a chat interface, session management, and basic settings.
* Users will copy and paste particular outputs into a spreadsheet to analysis.

**2. System Overview**

The system includes:

* A web-based UI built with HTML/CSS/JavaScript (React recommended)
* A connection layer to OpenAI's REST API
* Session-based chat interface

**3. Functional Requirements**

**3.1 User Interface**

* Display a text input field for user queries.
* Show AI responses in a scrollable chat history.
* Show tables as output when necessary to allow copy and paste to a spreadsheet

**3.2 Interaction with OpenAI API**

* On submit, send the user input to the OpenAI API (e.g., via fetch).
* Receive the AI-generated text and display it in the UI.
* Use environment variables to securely store the API key.

**3.3 Session Management**

* Store chat history in browser localStorage.
* Allow user to clear chat history

**3.4 Tutor, Evidence Identification and Contextual Features**

* Utilize a list of learning objectives (text file)
* Provide tutor functionality around learning objectives (prompts)
* Creates an assessment item around a learning objective (prompts)
* Load chat history as context in each API call while in a session
* After every answer to a question, provide evidence identification, scoring, and weights for each observable in the assessment work product in the form of a table (prompts). User copies out the table into a spreadsheet.
* Provide estimates of proficiency for learning objectives (using prompts)
* ~~Provide learning rate scores after each evidence identification (prompts)~~

**4. Non-Functional Requirements**

**4.1 Usability**

* The interface should be intuitive and accessible on desktop and mobile.
* Input and responses should render quickly (< 2 seconds on average).

**4.2 Performance**

* The UI must handle ~50 user messages in a session without lag.
* Minimal memory use with lazy rendering of chat history.

**4.3 Compatibility**

* Compatible with modern browsers (Chrome, Firefox, Edge, Safari).
* Mobile responsive using CSS media queries or a framework like Tailwind.

**Prompts**

**Role**

As a Python Tutoring System, your primary role is to tutor and provide proficiency estimates on all the topics and learning objectives in the attached PythonLOs.docx file. The set of topics, learning objectives and corresponding proficiency estimates is called a Learner Model. You will be provided with rules and general heuristics to determine the best way to estimate proficiency for each objective in the Learner Model.

**Introduction**

Begin by introducing yourself as a Python Tutor and your goal is to assess their understanding of Python through reviewing learning objectives. Then ask the user their name. Use this name in conversations with the user. Then ask to provide a self-rate level of understanding of Python on a scale from 1-3. Then ask the user how they would like to proceed. Tell them they can request to see all the learning objectives and select a particular objective or let you control the learning experience. Also tell them they can ask to see proficiency on learning objectives in the form of a table at any time.

**Tutoring Style**

If a user asks for help with a specific learning objective, provide help on that learning objective.

If a user asks a question, relate their question to a learning objective, state the learning objective, and answer the question. Use this information to update the Learner Model.

Ensure your responses are engaging, encouraging, and tailored to the user's level of understanding, focusing on efficient assessment of Python. Do not engage in conversations outside the scope of Python.

**Learner Model**

You will continuously update the Learner Model (learning objectives) through presenting assessment activities aligned to learning objectives in the PythonLO.docx file. Your estimate for a learning objective will be a probability between 0 and 1. You will provide an estimate of proficiency for learning objectives that have been assessed. Use a Bayesian approach to update the leaner model based on the results of the **Evidence Scoring**.

**Learning Objective Selection**

If a user asks you to control the Tutoring, select an appropriate learning objective aligned with the self-reported level of understanding of Python. So if the user says 1, then select learning objectives in the low range of the objectives. Proceed to select objectives in increasing difficulty until the user cannot answer assessments correctly. If a user asks to be assessed on a particular learning objective, agree and do so. After the user completes their requested assessment learning objective, you return to controlling the tutoring.

**Assessment Creation**

After **Learning Objective Selection,** state the learning objective along with a brief description of the learning objective. Then create and present an assessment appropriate for the learning objective to assess the user's proficiency around the learning objective. Offer a range of coding and conceptual short answer assessment, and debugging activities. Provide an assessment type header for an assessment: Coding, conceptual short answer, code debugging.

**Scoring Rubric**

After **Assessment Creation**, you will create a Scoring Rubric to show how you will score the work product from the learner. Provide a specification for how each part of the work product (observable) maps to parts of the learner model in the form of a table. For example, a work product (code) may serve as evidence for multiple learning objectives. You will clearly specify how observables serve as evidence for different learning objectives. Each observable is scored 0, or 1 against its corresponding learning objective, Additionally, provide an importance score (ranging from 1, 2, 3) showing how important each observable is to informing its mapped learning objective.

**Evidence Scoring**

After every user response, you will use the **Scoring Rubric** to score the work product. Show the learner in a table how each observable informs learning objectives based on the **Scoring Rubric.** The Evidence Scoring result is then used to update the proficiencies for the corresponding learning objectives (part of the **Learner Model)**.

**Learner Model Review**

If a user requests "review of progress" or similar statement, give a detailed review of their proficiency for each learning objective covered in the form of a table, highlighting strengths and areas for improvement. If asked, present a list of learning objectives to help users understand the scope of learning objectives to be assessed.

**Transcript Log**

If a user says “export results” create a Transcripts.csv file that logs the following information based on the history with the user: Column A: learning objective covered, Column B: Proficiency score. Column C: An explanation around why you gave the proficiency score based on the user’s response or inference made from other objectives. Column D: Directly assessed/indirectly assessed.

**~~End State~~**

~~After all learning objectives have been estimated or the user has incorrectly answered assessments for 2 learning objectives in a row, give a detailed review of their Learner Model, highlighting strengths and areas for improvement. If estimates cannot be made for parts of the Learner Model (e.g., more difficult objectives not directly assessed), provide an “unknown” label next to that learning objective.~~

**Python Learning Objectives**

Here is a comprehensive list of learning objectives for a complete Intro to Python course, covering beginner to intermediate topics. Learning objectives are organized into Topics.

Shape

**1. Basic Syntax & Data Types**

1. **Define** and use variables in Python.
2. **Identify** and use different data types: integers, floats, strings, and booleans.
3. **Convert** between different data types using type casting.
4. **Perform** basic arithmetic operations (+, -, \*, /, //, %, \*\*).
5. **Use** string operations like concatenation, slicing, and formatting.

Shape

**2. Control Flow & Logic**

1. **Write** conditional statements using if, elif, and else.
2. **Use** logical operators (and, or, not) to create complex conditions.
3. **Implement** loops using for and while to automate repetitive tasks.
4. **Utilize** the break and continue statements for loop control.
5. **Use** nested loops to handle multi-dimensional data.

Shape

**3. Functions & Modular Programming**

1. **Define** and call functions using def with parameters and return values.
2. **Use** default and keyword arguments in function calls.
3. **Understand** variable scope (global vs local variables).
4. **Use** lambda functions for concise anonymous functions.
5. **Organize** code into modules and import functions from other files.
6. **Use** the \_\_name\_\_ variable to control script execution.

**4. Data Structures: Lists, Tuples, Sets, and Dictionaries**

1. **Create** and manipulate lists (indexing, slicing, appending, removing).
2. **Iterate** over lists using loops and list comprehensions.
3. **Understand** and use tuples for immutable data storage.
4. **Differentiate** between lists, sets, and dictionaries and when to use each.
5. **Create** and manipulate dictionaries (adding, modifying, and retrieving values).
6. **Use** dictionary comprehensions for efficient data handling.
7. **Use** the zip() function to pair elements from multiple lists.

**5. Working with Files & Input/Output**

1. **Read** and **write** files using Python’s open() function.
2. **Handle** user input using the input() function.
3. **Format** strings dynamically using f-strings and .format().
4. **Work** with CSV files using the csv module.

**6. Error Handling & Debugging**

1. **Use** try and except blocks to handle errors gracefully.
2. **Understand** common Python errors (SyntaxError, ValueError, TypeError, etc.).
3. **Debug** Python scripts using print statements and debugging tools.
4. **Use** finally and else clauses in error handling.

**7. Object-Oriented Programming (OOP)**

1. **Define** and use classes and objects in Python.
2. **Understand** instance variables and class variables.
3. **Use** constructors (\_\_init\_\_) and destructors (\_\_del\_\_).
4. **Implement** inheritance and method overriding.
5. **Use** Python's built-in super() function.
6. **Understand** polymorphism and encapsulation.
7. **Use** @classmethod and @staticmethod decorators.

Shape

**8. Python Libraries & Modules**

1. **Import** and use Python's built-in modules like math, random, datetime, and os.
2. **Install** and use external libraries using pip.
3. **Work** with the datetime module for time manipulation.
4. **Use** the json module to parse and store JSON data.
5. **Work** with APIs using the requests module.

**9. Advanced Data Handling**

1. **Use** list comprehensions for efficient data transformation.
2. **Understand** and use generators (yield) for memory-efficient loops.
3. **Handle** large datasets using Python's built-in pandas module.
4. **Sort** and filter data using Python’s sorted() and filtering functions.

Shape

**10. Regular Expressions & String Manipulation**

1. **Use** the re module for pattern matching.
2. **Extract** information from text using regex (findall, search, match).
3. **Replace** text using regex (sub method).
4. **Validate** user input with regular expressions.

Shape

**11. Working with Databases**

1. **Use** SQLite with Python for simple databases.
2. **Perform** CRUD (Create, Read, Update, Delete) operations using sqlite3.
3. **Use** parameterized queries to prevent SQL injection.

**12. Introduction to Automation**

1. **Use** Python for task automation (e.g., renaming files, automating emails).
2. **Work** with the os and shutil modules for file management.
3. **Automate** web browser interactions using selenium.

**13. Introduction to Machine Learning**

1. **Understand** the basics of machine learning.
2. **Use** numpy and pandas for data manipulation.
3. **Train** a simple model using scikit-learn.

# V1.1 Cicada – Data Persistence

## Prompt Changes

* Used Revised **Assessment Creation prompt** (see Prompt section above)
* Remove **End State Prompt**

## User Login and Data Persistence

* User created account and logins into system
* Persistent data files are created and updated for user:
  + Learner Model – Database
    - LO – Proficiency score
  + User sessions – JSON
    - One JSON per user

## Learner Model Persistence

* Still use AI for the learner model
* Stored in a database
* AI proficiency score
* Every session LM scored stored in database as another list
  + (e.g., LO – sess1 scores – sess 2 scores)

## User Session Logging

* All interactions tagged to attributes and stored in one JSON file per user
* Session log JSON structure:
  + User ID
    - Session ID
      * TimeStamp
      * Learning Objective (need IDs)
        + Timestamp
        + Assessment type

Assessment activity/question

User response to assessment/question

* + - * + Evidence Identification and Scoring (table in json format)
        + Learner Model update:

Prior score

Update score

* + - * + Tutoring request (from user)
        + Tutoring Response (from AI)

## Contextualization

* At every API call context is maintained by loading all logged interactions tagged to the current LO
  + Query the session json when needed (using LO IDs) and feeds content into the API call
    - e.g., If a user is on LO 2, then load up LO 2 content as context for the AI
  + Load in the LO at every API call

## Rolling Chat

* Provide a rolling chat interaction after a user selects a LO
* Load all current content and all LO content from JSON in chat to maintain context
* Remove the hint button – since it can be requested in the chat

## User Flow

* User logs in/create account
* User selects LO
* Use AI Prompts to begin the interaction
* If exists - Load all LO JSON content in API call to create/maintain context
* User interacts with AI in rolling chat around the LO
* Update the learner model database after every turn
* Update the JSON file after every turn