Egg Timer Project

A Python-Based CLI Cooking Assistant

Automating the Art of Breakfast

Introduction

The **Egg Timer** is a user-friendly, command-line interface (CLI) application developed in Python.

Its primary purpose is to assist users in cooking eggs to perfection by providing precise, automated timers for various preparation methods.

By simulating a digital kitchen assistant, the project combines functional utility with engaging ASCII art visuals to enhance the user experience.

The design emphasizes simplicity and rapid operation for kitchen use.

Problem Statement

The Challenge

- Cooking eggs requires precise timing to achieve the desired consistency (soft vs. hard boiled).
- Manual tracking is prone to human error, often resulting in undercooked or overcooked meals.
- Many existing tools lack a simple, lightweight interface for quick access on desktop environments.

Our Solution: An automated, dedicated script that standardizes cooking times.

Functional Requirements

Menu Selection

Users must be able to select from predefined egg types: Soft Boiled, Hard Boiled, Omelet, or Sunny Side Up.

Automated Timer

The system must calculate and execute a countdown based on the selected egg type.

Visual Feedback

The application must display relevant ASCII art and progress indicators during the process.

Non-functional Requirements

Performance

The application should load instantly and respond to user inputs without noticeable latency.

Portability

The script should run on any system with a standard Python environment (Windows, macOS, Linux).

Usability

The interface should be intuitive, using clear text prompts to guide the user through the workflow.

System Architecture

The system follows a linear procedural flow utilizing standard Python libraries.

- **Input Module:** Captures user choice via `input()` and `sys`.
- **Logic Core:** Determines duration based on selection statements (`if/elif`).
- **Timing Engine:** Uses `time.sleep()` for delays and `datetime` for real-time calculation.
- **Display Module:** Renders text and ASCII art to the console.

Design Diagrams

Workflow

1. **Start:** Display Intro Art.

2. **Input:** User selects Egg Type & Quantity.

3. **Process:** Calculate End Time.

4. **Action:** Run Countdown loop.

5. **End:** Alert user & Show Final Art.

Use Case

Actor: Home Cook

Goal: Boil an Egg

Pre-condition: Python installed.

Scenario: User launches app, selects "Soft Boiled", waits for timer, removes egg.

Design Decisions & Rationale

Why Python?

Chosen for its readability and powerful standard libraries like `datetime`, allowing for rapid prototyping.

Why CLI?

A Command Line Interface ensures the tool is lightweight and distraction-free, focusing purely on utility.

Why ASCII Art?

Provides a visual "fun factor" without the overhead of complex GUI libraries (like Tkinter) for a simple tool.

Implementation Details

The core functionality relies on the **`time`** and **`datetime`** modules.

Key Functions:

- **`intro()`:** Handles the visual startup sequence using `time.sleep` for a "typing" effect.
- **`body()`:** The main logic engine that accepts user inputs (`choi`, `chi`) and executes the specific timer loop.

The progress bar is simulated by printing dots iteratively to the standard output buffer.

Screenshots & Output

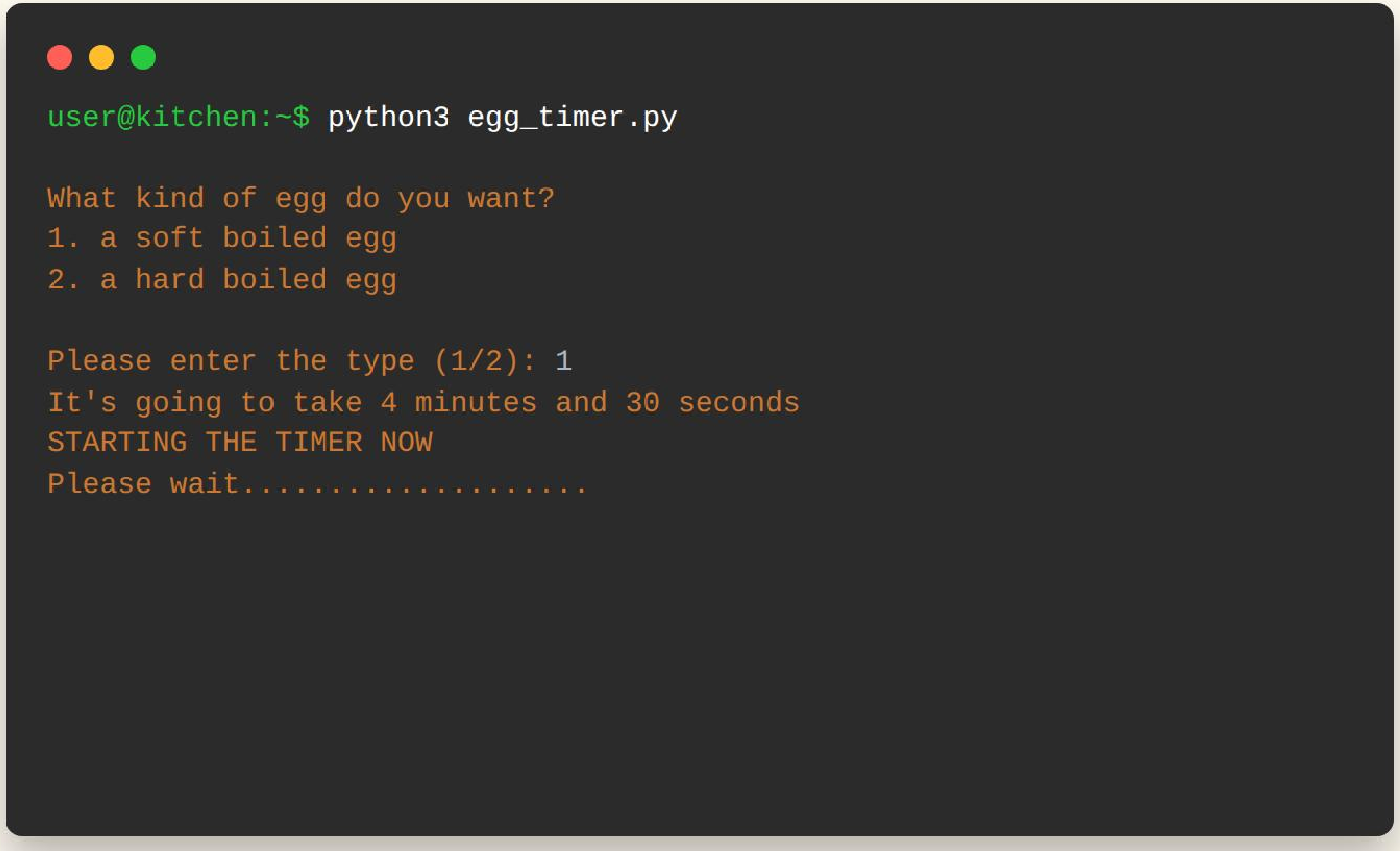
Simulated Execution

The interface guides the user clearly through the process.

1. Type selection is typed out for effect.

2. The estimated time is calculated instantly.

3. A real-time visual indicator shows activity.



```
user@kitchen:~$ python3 egg_timer.py

What kind of egg do you want?
1. a soft boiled egg
2. a hard boiled egg

Please enter the type (1/2): 1
It's going to take 4 minutes and 30 seconds
STARTING THE TIMER NOW
Please wait....................
```

Testing Approach

- **Unit Testing:** Verified that input `1` triggers the correct 4:30 minute timer and input `2` triggers the 13:30 minute timer.
- **Boundary Testing:** Tested input validation by entering numbers outside the 1-6 range for egg quantity.
- **Visual Inspection:** Checked ASCII art alignment on standard 80-column terminal windows to ensure no wrapping issues.
- **User Acceptance:** Confirmed the "typing effect" speed was readable and not too slow.

Challenges Faced

ASCII Alignment

Getting the ASCII art (the eggs and titles) to look consistent across different screen sizes was difficult. We had to manually adjust whitespace and escape characters.

Timer Logic

Creating a blocking timer using `time.sleep` halts the entire program. While simple, it prevents other interactions. Balancing the delay intervals for the "typing effect" was also tricky.

Learnings & Key Takeaways

Python Modules

Gained deep practical knowledge of the `datetime` module for manipulating timestamps and the `sys` module for output flushing.

CLI UX Design

Learned that even text-based interfaces need good UX. Pacing the text output makes the app feel more "alive" and responsive.

Control Flow

Mastered the use of conditional logic (`if/elif/else`) to direct program execution based on complex user inputs.

Future Enhancements

Roadmap V2.0

- **Graphical User Interface (GUI):** Implement a frontend using Tkinter or PyQt for buttons and visual progress bars.
- **Audio Alerts:** Integrate the `playsound` library to ring an alarm when the eggs are ready.
- **Recipe Database:** Expand the menu to include poaching instructions and scrambled egg timers.
- **Mobile App:** Port the logic to Kivy for Android deployment.

References

- Python 3 Documentation - *docs.python.org*
- GeeksforGeeks Python Tutorials - *geeksforgeeks.org*
- ASCII Art Archive - *asciiart.eu*
- "The Science of Boiling an Egg" - *Serious Eats*
- StackOverflow Community Forums