NextWave Assignment By Anudeep Konagala

Masterclass Plan: Building AI Teammates That Fix Your Code

Hey everyone The goal for this session is to take you on a journey. We'll start by looking at how AI can help us fix code, and by the end, we'll have built a fully autonomous AI agent that can find and squash bugs all by itself. We'll even look at a wild "meta" concept: using AI to write our AI agent *for* us.

Part 1: Outline of the project

- **The Concept:** We all spend way too much time debugging. What if we could teach an AI to do it for us We're not just talking about simple syntax errors. We're talking about an AI that can understand why a test is failing and logically fix the code.
- **Our Game Plan:** We'll build two different tools from the project:
    1. **The AI Assistant:** A simple web app where you're still the boss. You paste in broken code, and the AI gives you a fix.
    2. **The AI Agent:** This is the main event. A fire and forget script that runs your tests, finds the bugs, and fixes them on its own while you grab a coffee.
- **Our Toolkit:** The main tools for today will be **Python**, Google's **Gemini** AI, **Flask** for our web app, and **Pytest** to act as our bug-finder.

Part 2: Our First Build - The AI Code Assistant

- **The Concept:** We'll start simple , We're building a tool that has a human in the loop This is perfect for getting comfortable with the Gemini API and understanding the most important part: prompts. We'll learn how to prime the AI with a system_instruction to make it act like an expert developer.
- **Let's Look at the Code:**
    - **The Backend (app.py):** We'll walk through the Flask server. It's surprisingly simple! We'll see how it securely grabs the API key, sets up an API endpoint, and passes our code and our special prompt to the AI.
    - **The Frontend (templates/index.html):** We'll check out the webpage and the JavaScript behind it. We'll see how it fetch-es data from our own backend, handles loading states, and even has a cool "retry" logic in case the API call stumbles.
- **Our Toolkit:** Flask, google-generativeai, python-dotenv, and a bit of HTML/JS.

Part 3: The Main Event - Building an Autonomous Agent

- **The Concept:** Alright, training wheels off. Now we build the "autonomous" agent. The core idea is a simple, powerful loop: **Test -> Fail -> Analyze -> Fix -> Repeat**.
    - We'll use pytest as the agent's "eyes and ears."
    - We'll use subprocess to let our Python script run pytest in the terminal and, most importantly, *read the error messages*.
    - We'll use Regular Expressions (re) to "read" those errors and figure out *which file* crashed.
    - And to be safe, we'll use shutil to backup **the project** before our agent touches anything. If it messes up, it can just restore from backup.
- **Let's Look at the Code (generated_agent/agent.py):**
    - This is the heart of the project. We'll trace the logic flow step-by-step.
    - backup_project() / restore_project(): How to be safe and not sorry.
    - run_tests(): The function that runs pytest and captures the output.
    - generate_fix(): **This is the brain**. We'll see how it builds a *super-detailed prompt* for the AI, complete with the error message and the full source code of the broken file.
    - main(): We'll see how the main() function ties this all together in a for loop, giving the agent 3 attempts to fix the build.
- **Our Toolkit:** google-generativeai, subprocess, pytest, shutil, and re.

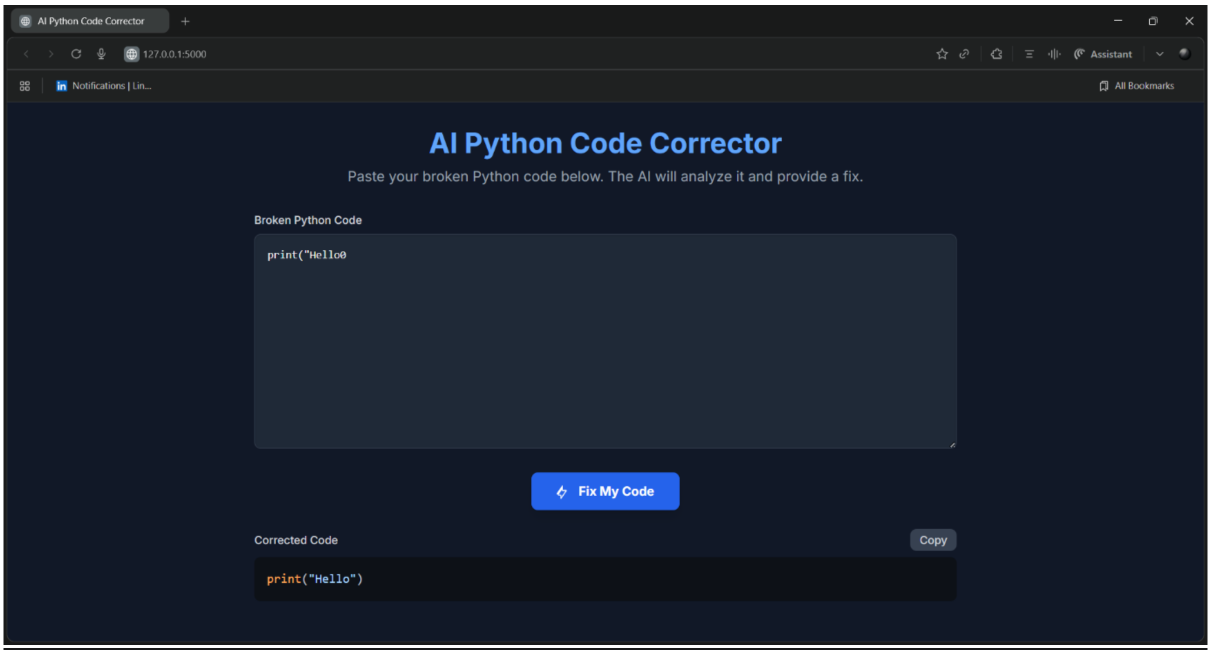
Part 4: Next-Level Stuff - "Prompt-Driven Development" (30 Mins)

- **The Concept:** This part is super cool. That agent.py file from Part 3 is pretty complex, right? What if we didn't write it? What if we just wrote a "blueprint" and had the AI write the agent *for* us?
- **Let's Look at the Code:**
    - **The Blueprint (main.prompt):** This is our "source code." We'll look at this plain text file and see how it's a detailed set of instructions. It tells the AI *exactly* what functions to write, what libraries to use, and what the goal is.
    - **The "Builder" (smol_dev.py):** This script is tiny. All it does is read the main.prompt blueprint, send it to the Gemini API, and save the (massive) code response as agent.py.
- **Live Demo:** We'll run python smol_dev.py right in the session and watch it generate our entire autonomous agent from that one prompt file. It's a bit mind-blowing.
- **Our Toolkit:** main.prompt (our blueprint) and smol_dev.py (our builder).

Part 5: Execution

- **Putting It All Together:** We'll do a quick review of the setup: getting the requirements.txt.txt installed, and most importantly, creating that .env file with your GOOGLE_API_KEY.
- **The Big Demo:**
    1. We'll launch the **web app** (flask run) and fix some code live.
    2. We'll run the **autonomous agent** (python agent.py) against the broken target_project and watch its log file (agent.log) to see it think, fail, and (hopefully) succeed in fixing the code.
- **Open Discussion & Q&A:** After that, I'll open the floor. We can talk about anything—this project, prompt engineering, or where this AI-driven development stuff is headed.

Test Cases :



AI Python Code Corrector

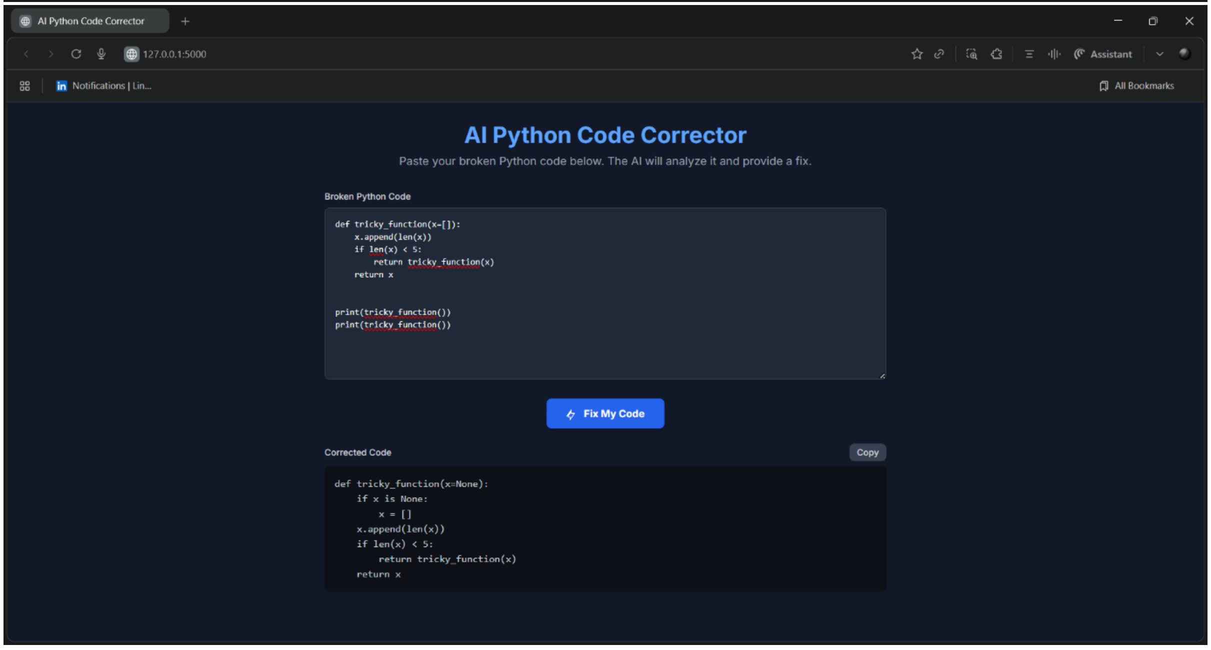
Paste your broken Python code below. The AI will analyze it and provide a fix.

Broken Python Code

```
print("Hello0
```

Corrected Code

```
print("Hello")
```



AI Python Code Corrector

Paste your broken Python code below. The AI will analyze it and provide a fix.

Broken Python Code

```
def tricky_function(x=[]):
    x.append(len(x))
    if len(x) < 5:
        return tricky_function(x)
    return x


print(tricky_function())
print(tricky_function())
```

Corrected Code

```
def tricky_function(x=None):
    if x is None:
        x = []
    x.append(len(x))
    if len(x) < 5:
        return tricky_function(x)
    return x
```