5 Beginner AI Projects - Implementation Guide

Here is a comprehensive guide for AI Engineering Fellowship students to build their first AI applications using Python and AI APIs (Grok/Google AI).

Project Overview

All projects follow a similar structure but teach different aspects of AI application development:

- **Input Processing:** How to handle different types of user data
- **Prompt Engineering:** Crafting effective prompts for different use cases
- **Output Processing:** Formatting and presenting AI responses
- **User Experience:** Creating intuitive interfaces

Project 1: AI Story Generator

Goal: Generate creative stories based on user inputs
Time Estimate: 1-1.5 hours
Difficulty: Beginner

Core Components

Input Processing

```
# Simple text inputs
character = input("Enter a character name: ")
setting = input("Enter a setting: ")
genre = input("Enter a genre (fantasy, sci-fi, mystery): ")
```

Prompt Engineering Strategy

- Use structured prompts with clear instructions
- Include examples of desired output format

- Set creative constraints (word count, style)

```
prompt = f"""
Write a {genre} story with the following elements:
- Main character: {character}
- Setting: {setting}
- Length: 200-300 words
- Include dialogue and descriptive language
- End with a twist or cliffhanger
"""
```

Output Processing

- Clean up formatting
- Add title generation
- Optional: Save stories to file

Enhancement Ideas

- Story length options (short, medium, long)
- Multiple genre combinations
- Character personality traits
- Story continuation feature

Project 2: Meeting Notes Summarizer

Goal: Transform messy meeting notes into organized summaries
Time Estimate: 1.5-2 hours
Difficulty: Beginner-Intermediate

Core Components

Input Processing

```python
# Handle large text blocks
notes = input("Paste your meeting notes here:\n")

# Optional: Read from file
def read_notes_from_file(filename):
    with open(filename, 'r') as file:
        return file.read()
```

Prompt Engineering Strategy

- Use extraction-focused prompts
- Request structured output
- Handle different meeting types

```python
prompt = f"""
Analyze these meeting notes and provide:
1. SUMMARY: 2-3 sentence overview
2. KEY POINTS: Main discussion topics (bullet points)
3. ACTION ITEMS: Who needs to do what by when
4. DECISIONS MADE: Any conclusions or agreements

Meeting Notes:
{notes}

Format your response clearly with headers.
"""
```

Output Processing

```python
def format_summary(ai_response):
    # Split response into sections
    # Add consistent formatting
    # Optional: Export to different formats
    pass
```

Enhancement Ideas

- Meeting type detection (standup, planning, review)
- Participant tracking
- Priority levels for action items
- Integration with calendar/task apps

Project 3: Personal Learning Tutor

Goal: Create adaptive explanations and practice questions
Time Estimate: 1.5-2 hours
Difficulty: Intermediate

Core Components

Input Processing

```
# Structured input collection
topic = input("What topic would you like to learn about? ")
current_level = input("Your current level (beginner/intermediate/advanced): ")
learning_style = input("Preferred style (visual/examples/step-by-step): ")
```

Prompt Engineering Strategy

- Adaptive complexity based on level
- Multiple explanation approaches
- Interactive Q&A generation

```python
def create_tutor_prompt(topic, level, style):
    base_prompt = f"""
    You are a patient tutor explaining {topic} to a {level} student.

    Provide:
    1. Simple explanation using {style} approach
    2. Real-world analogy
    3. 2-3 practice questions with answers
    4. Common mistakes to avoid

    Keep language appropriate for {level} level.
    """
    return base_prompt
```

Output Processing

```python
def interactive_learning_session():
    # Present explanation
    # Ask follow-up questions
    # Provide feedback on answers
    # Suggest next topics
    pass
```

Enhancement Ideas

- Progress tracking
- Difficulty adjustment based on performance
- Multi-subject support
- Spaced repetition for practice questions

Project 4: Recipe Remix Chef

Goal: Generate recipes from available ingredients
Time Estimate: 1.5-2 hours
Difficulty: Intermediate

Core Components

Input Processing

```
# Handle lists and constraints
def get_recipe_inputs():
    ingredients = input("Enter available ingredients (comma-separated): ").split(',')
    ingredients = [ingredient.strip() for ingredient in ingredients]

    dietary_restrictions = input("Any dietary restrictions? (vegetarian, vegan, gluten-free, et
    cooking_time = input("How much time do you have? (15 min, 30 min, 1 hour): ")
    skill_level = input("Cooking skill level (beginner, intermediate, expert): ")

    return ingredients, dietary_restrictions, cooking_time, skill_level
```

Prompt Engineering Strategy

- Constraint-based recipe generation
- Ingredient substitution suggestions
- Difficulty-appropriate techniques

```
def create_recipe_prompt(ingredients, restrictions, time, skill):
    prompt = f"""
    Create a recipe using these ingredients: {', '.join(ingredients)}

    Constraints:
    - Dietary restrictions: {restrictions}
    - Cooking time: {time}
    - Skill level: {skill}

    Provide:
    1. Recipe name
    2. Ingredients list (with quantities)
    3. Step-by-step instructions
    4. Cooking tips for {skill} level
    5. Possible substitutions for missing ingredients
    """
    return prompt
```

Output Processing

```python
def format_recipe(ai_response):
    # Parse into structured format
    # Add nutrition estimates (optional)
    # Generate shopping list for missing ingredients
    # Save favorite recipes
    pass
```

Enhancement Ideas

- Cuisine type preferences
- Nutritional information
- Cooking technique tutorials
- Recipe rating and favorites system

Project 5: Career Advice Counselor

Goal: Provide personalized career guidance
Time Estimate: 2 hours
Difficulty: Intermediate-Advanced

Core Components

Input Processing

```python
def collect_career_info():
    current_role = input("Current role/student status: ")
    skills = input("Current skills (comma-separated): ").split(',')
    interests = input("What interests you? ")
    career_goals = input("Career goals (short-term and long-term): ")
    experience_level = input("Years of experience: ")

    return {
        'current_role': current_role,
        'skills': [skill.strip() for skill in skills],
        'interests': interests,
        'career_goals': career_goals,
        'experience_level': experience_level
    }
```

Prompt Engineering Strategy

- Multi-faceted analysis
- Actionable advice generation
- Resource recommendations

```python
def create_career_prompt(profile):
    prompt = f"""
    You are a career counselor analyzing this profile:

    Current Role: {profile['current_role']}
    Skills: {', '.join(profile['skills'])}
    Interests: {profile['interests']}
    Goals: {profile['career_goals']}
    Experience: {profile['experience_level']} years

    Provide:
    1. STRENGTHS: Key strengths based on current profile
    2. SKILL GAPS: What skills to develop for their goals
    3. NEXT STEPS: 3-5 actionable steps for next 6 months
    4. LEARNING RESOURCES: Specific courses, books, or certifications
    5. NETWORKING: How to connect with people in their target field
    6. TIMELINE: Realistic timeline for achieving goals

    Be specific and actionable in your advice.
    """
    return prompt
```

Output Processing

```python
def create_action_plan(ai_response):
    # Parse advice into categories
    # Create timeline with milestones
    # Generate learning checklist
    # Save plan for progress tracking
    pass
```

Enhancement Ideas

- Industry-specific advice
- Salary information integration
- Job market analysis
- Progress tracking and updates

Common Implementation Patterns

Basic Project Structure

```python
# main.py
import os
from api_client import get_ai_response

def main():
    print("Welcome to [Project Name]!")

    # 1. Collect user input
    user_input = collect_input()

    # 2. Create prompt
    prompt = create_prompt(user_input)

    # 3. Get AI response
    response = get_ai_response(prompt)

    # 4. Process and display output
    formatted_output = process_output(response)
    display_output(formatted_output)

if __name__ == "__main__":
    main()
```

API Client Setup

```python
# api_client.py
import requests
import os

def get_ai_response(prompt, api_choice="grok"):
    if api_choice == "grok":
        return call_grok_api(prompt)
    elif api_choice == "google":
        return call_google_ai_api(prompt)

def call_grok_api(prompt):
    # API implementation
    headers = {
        "Authorization": f"Bearer {os.getenv('GROK_API_KEY')}",
        "Content-Type": "application/json"
    }

    data = {
        "messages": [{"role": "user", "content": prompt}],
        "model": "grok-beta"
    }

    # Make API call and return response
    pass

def call_google_ai_api(prompt):
    # Google AI implementation
    pass
```

Error Handling Template

```python
def safe_api_call(prompt):
    try:
        response = get_ai_response(prompt)
        return response
    except Exception as e:
        print(f"Error: {e}")
        return "Sorry, there was an issue generating a response. Please try again."
```

Getting Started Checklist

Prerequisites

- [ ] Python 3.7+ installed
- [ ] API key for chosen service (Grok or Google AI)
- [ ] Basic understanding of Python variables and functions

Setup Steps

1. **Environment Setup**

   ```
   pip install requests python-dotenv
   ```

2. **API Key Configuration**

   ```
   # Create .env file
   echo "GROK_API_KEY=your_api_key_here" > .env
   ```

3. **Choose Your First Project**
   - Start with AI Story Generator (simplest)
   - Move to Meeting Notes Summarizer
   - Progress to more complex projects
4. **Development Process**
   - Build basic version first
   - Test with simple inputs
   - Add enhancements incrementally
   - Get feedback from fellow students

Learning Progression

Day 1: Build Projects 1-2 (Story Generator, Meeting Notes)
Day 2: Build Projects 3-4 (Learning Tutor, Recipe Chef)
Day 3: Build Project 5 (Career Counselor) + Add enhancements to previous projects

Beyond Basics: Connect projects with databases, add web interfaces, or create mobile apps

Resources for Further Learning

- **API Documentation:** Check official docs for your chosen API
- **Prompt Engineering:** Learn advanced prompting techniques
- **Python Libraries:** Explore `streamlit` for web interfaces, `sqlite3` for data storage
- **Deployment:** Consider `Heroku`, `Replit`, or `GitHub Pages` for sharing projects

Remember: The goal isn't perfect code on the first try. Focus on learning, experimenting, and building something that works. Each project teaches different skills that build upon each other!