Complete Guide to PRD Generation System

A Beginner's Guide to Python Programming and AI Integration

From the July AI Team

June 26, 2025

Contents

1 Homework Assignment **3**

2 Introduction: What is This System? **3**
2.1 How the System Works (Overview) . . . 3

3 Understanding Python Basics in This Code **4**
3.1 What are Imports? . . . 4
3.2 What are Classes and Methods? . . . 4
3.3 Understanding the Main Function . . . 5

4 Breaking Down the Code: Component by Component **5**
4.1 The PRDGenerationSystem Class Setup . . . 5
4.2 File Processing: The encode_pdf_to_base64 Method . . . 6
4.3 The PRD Template: get_prd_generation_prompt Method . . . 6
4.4 AI Integration: The ChatGPT Generation Method . . . 7
4.4.1 Understanding the For Loop Design . . . 8
4.5 User Interface Methods . . . 9
4.5.1 Displaying Results . . . 9
4.5.2 Getting User Input . . . 9

5 The Main Process Flow **10**
5.1 Deep Dive: Understanding process_prd_generation() . . . 10
5.1.1 Part 1: Setting Up and Getting User Input . . . 10
5.1.2 Part 2: Collecting Context Files . . . 11
5.1.3 Part 3: Configuration Summary and AI Generation . . . 12
5.1.4 Part 4: User Choice and File Saving Logic . . . 13
5.1.5 Understanding the Control Flow . . . 15
5.1.6 Error Handling Patterns . . . 16
5.1.7 Variable Naming and Code Organization . . . 17
5.2 The Main Function Structure . . . 17

6 Error Handling and Robustness **18**
6.1 Try-Except Blocks . . . 18
6.2 File Existence Checking . . . 18

7 Advanced Concepts and Design Decisions **18**
7.1 Why Use Classes Instead of Simple Functions? . . . 19
7.2 API Design Patterns . . . 19
7.3 Data Structure Choices . . . 19

8 Practical Usage Guide **20**
8.1 Setting Up the System . . . 20
8.2 Best Practices for Input Files . . . 20
8.3 Troubleshooting Common Issues . . . 21

9 Extending and Customizing the System **21**
9.1 Adding New AI Providers . . . 21
9.2 Customizing the PRD Template . . . 21

10 Step-by-Step Guide: Actually Using the System **21**
10.1 Preparing Your Files . . . 22
10.1.1 Finding Files on Your Computer . . . 22
10.1.2 Organizing Your Project Files . . . 23
10.2 Running the System: Complete Walkthrough . . . 23
10.2.1 Step 1: Starting the Program . . . 23
10.2.2 Step 2: Providing Your Main Information File . . . 23
10.2.3 Step 3: Adding Context PRDs (Optional) . . . 24
10.2.4 Step 4: The Generation Process . . . 24
10.2.5 Step 5: Reviewing and Choosing . . . 25
10.3 Real-World Usage Examples . . . 25
10.3.1 Example 1: Startup Founder . . . 25
10.3.2 Example 2: Product Manager . . . 26
10.4 Understanding the Output . . . 26
10.4.1 Where Files Are Saved . . . 26
10.4.2 Working with the Generated PRDs . . . 26
10.5 Troubleshooting Common Issues . . . 27
10.5.1 File Path Problems . . . 27
10.5.2 API and Network Issues . . . 27
10.6 Pro Tips for Better Results . . . 28

11 Conclusion **28**

1 Homework Assignment

Programming Assignment: Create Your Own AI Integration

Objective: Using this guide and the existing codebase as your reference, create a new function called `generate_gemini_prd()` that integrates Google's Gemini AI into the PRD generation system.

What You Need to Create:

1. A new method `generate_gemini_prd()` that follows the same pattern as the existing ChatGPT and Claude functions
2. Integration into the main user choice system (add Gemini as a third option)
3. Proper error handling following the established patterns

Learning Goals:

- Understand how API integration works by implementing one yourself
- Practice reading and adapting existing code patterns
- Learn to extend object-oriented systems
- Gain experience with error handling and user interface design

2 Introduction: What is This System?

This document will teach you how to understand and use a Python program that automatically creates **Product Requirements Documents (PRDs)** using artificial intelligence. Think of it as having two expert consultants (ChatGPT and Claude AI) work together to transform your ideas into professional business documents.

What's a PRD?

A **Product Requirements Document** is like a detailed blueprint for building a product. It explains:

- What problem you're solving
- Who will use the product
- What features it needs
- How success will be measured

2.1 How the System Works (Overview)

Your Input → AI Processing → Two PRDs → You Choose Best → Final Document

3 Understanding Python Basics in This Code

Before diving into the specific functions, let's understand the fundamental Python concepts used in this program.

3.1 What are Imports?

At the very beginning of our code, you'll see several `import` statements:

```
from pathlib import Path
import openai
import anthropic
from typing import List
import base64
from google import genai
```

Understanding Imports

Think of imports like borrowing tools from a toolbox. Each import statement tells Python: "I want to use tools from this specific toolbox."

- `from pathlib import Path` - Tools for working with file paths
- `import openai` - Tools to communicate with ChatGPT
- `import anthropic` - Tools to communicate with Claude AI
- `from typing import List` - Tools to specify data types
- `import base64` - Tools to convert files to text format
- `from google import genai` - Tools for Google's AI (not used in this version)

3.2 What are Classes and Methods?

```
class PRDGenerationSystem:
    def __init__(self):
        # Setup code here

    def encode_pdf_to_base64(self, file_path: str) -> str:
        # Function code here
```

Classes and Methods Explained

Class: Think of a class like a blueprint for building a house. It defines what the house will have (rooms, doors, windows) but isn't the actual house yet.
Methods: These are like rooms in the house - each has a specific purpose:

- `__init__` - The "foundation" method that sets up the class when created
- `encode_pdf_to_base64` - A "utility room" that converts PDF files
- `generate_chatgpt_prd` - The "ChatGPT workshop" that creates PRDs

The `self` parameter is like saying "this specific house" - it refers to the particular instance of the class you're working with.

3.3 Understanding the Main Function

At the bottom of our code, you'll see:

```
def main():
    """Main function to run the PRD generation system"""
    system = PRDGenerationSystem()
    # More code here...

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

The Main Function

The `main()` function is like the "front door" of your program. It's where everything starts when you run the code.
The strange looking `if __name__ == "__main__":` is Python's way of saying "only run this code if someone directly runs this file" (not if it's imported into another program).
Think of it like: "If someone rings the front doorbell, start the house tour."

4 Breaking Down the Code: Component by Component

4.1 The PRDGenerationSystem Class Setup

```
class PRDGenerationSystem:
    def __init__(self):
        self.chatgpt_api_key = "your_chatgpt_api_key_here"
        self.claude_api_key = "sk-ant-api03-oQepAG71jee8--XikuEPtVlI...
    "
        self.gemini_api_key = "your_chatgpt_api_key_here"
```

The __init__ Method

This is called a "constructor" - it runs automatically when you create a new instance of the class.
Real-world analogy: When you buy a new phone, it comes pre-loaded with certain apps and settings. The `__init__` method is like that initial setup.
The `self.` prefix means "this belongs to this specific instance of the class." It's like putting your name on your belongings.

4.2 File Processing: The encode_pdf_to_base64 Method

```python
def encode_pdf_to_base64(self, file_path: str) -> str:
    """Encode PDF file to base64 for API upload"""
    with open(file_path, 'rb') as file:
        return base64.b64encode(file.read()).decode('utf-8')
```

Understanding This Method

Purpose: Converts PDF files into a format that AI can understand.
Parameter Types:

- `file_path: str` - This tells Python "expect a string (text) for the file path"
- `-> str` - This tells Python "this method will return a string"

The 'with' statement: This is Python's safe way to open files. It automatically closes the file when done, even if something goes wrong. Like automatically turning off the lights when you leave a room.
Why 'rb' mode?: The 'r' means "read" and 'b' means "binary." PDFs aren't text files - they're binary files (like images), so we need to read them in binary mode.

4.3 The PRD Template: get_prd_generation_prompt Method

```python
def get_prd_generation_prompt(self) -> str:
    """Get the core PRD generation prompt"""
    return """
    You are an expert Product Manager. Your task is to create a
   comprehensive Product Requirements Document (PRD) based on the
   provided information and context.

    A great PRD should include:

    ## 1. EXECUTIVE SUMMARY
    - Brief overview of the product/feature
    - Key business objectives
    - Success metrics
    # ... more sections
    """
```

Design Choice: Why a Separate Method?

Question: Why not just write this text directly in the AI functions?
Answer: This follows the "Don't Repeat Yourself" (DRY) principle:

- Both ChatGPT and Claude use the same prompt
- If we want to change the PRD structure, we only change it in one place
- Makes the code easier to maintain and modify

It's like having a master recipe that multiple chefs can follow, rather than each chef memorizing their own version.

4.4 AI Integration: The ChatGPT Generation Method

Let's examine the most complex method in detail:

```
def generate_chatgpt_prd(self, new_info_file: str, context_files: List[str] = None) -> str:
    """Generate PRD using ChatGPT with PDF context"""
    try:
        # Initialize OpenAI client with minimal parameters
        client = openai.OpenAI(
            api_key=self.chatgpt_api_key
        )

        # Build the message content
        message_content = [
            {
                "type": "text",
                "text": self.get_prd_generation_prompt()
            }
        ]

        # Add context PRDs if provided
        if context_files:
            message_content.append({
                "type": "text",
                "text": f"\n\nCONTEXT: Here are {len(context_files)} example PRD(s) for reference on style, format, and structure:"
            })

            for i, context_file in enumerate(context_files, 1):
                if Path(context_file).suffix.lower() == '.pdf':
                    message_content.append({
                        "type": "text",
                        "text": f"\n--- CONTEXT PRD {i}: {Path(context_file).name} ---"
                    })
                    message_content.append({
                        "type": "image_url",
                        "image_url": {
                            "url": f"data:application/pdf;base64,{self.encode_pdf_to_base64(context_file)}"
                        }
                    })
```

```
                else:
                    # For non-PDF files, read as text
                    with open(context_file, 'r', encoding='utf-8') as f:
                        content = f.read()
                    message_content.append({
                        "type": "text",
                        "text": f"\n--- CONTEXT PRD {i}: {Path(context_file).name} ---\n{content}"
                    })

        # ... rest of method
```

Breaking Down Complex Logic

The try-except pattern: This is Python's way of handling errors gracefully. It says "try to do this, but if something goes wrong, don't crash - handle it nicely."
List parameter with default: `context_files: List[str] = None` means:

- Expect a list of strings (file paths)
- If no list is provided, use None as default
- This makes the parameter optional

The message_content list: This builds a conversation with ChatGPT step by step, like preparing talking points for a meeting.

4.4.1 Understanding the For Loop Design

```
for i, context_file in enumerate(context_files, 1):
    if Path(context_file).suffix.lower() == '.pdf':
        # Handle PDF files
    else:
        # Handle text files
```

For Loop Design Choices

enumerate(context_files, 1):

- `enumerate` gives us both the item and its position in the list
- The 1 means "start counting from 1" (instead of 0)
- This lets us say "Context PRD 1", "Context PRD 2", etc.

Path(context_file).suffix.lower():

- `Path(context_file)` creates a path object from the file name
- `.suffix` gets the file extension (.pdf, .txt, etc.)
- `.lower()` converts to lowercase (so .PDF becomes .pdf)

Why this design?: The system needs to handle different file types differently. PDFs need special encoding, while text files can be read directly.

4.5 User Interface Methods

4.5.1 Displaying Results

```
def display_prds(self, chatgpt_prd: str, claude_prd: str):
    """Display both PRDs for comparison"""
    print("\n" + "="*80)
    print("PRD COMPARISON")
    print("="*80)

    print("\n ChatGPT PRD:")
    print("-" * 40)
    print(chatgpt_prd)

    print("\n Claude PRD:")
    print("-" * 40)
    print(claude_prd)
```

String Multiplication

`"="*80` is a Python shortcut that repeats the "=" character 80 times. It's like using a rubber stamp 80 times to create a decorative border.
This makes the output more readable by creating visual separation between sections.

4.5.2 Getting User Input

```
def get_user_choice(self) -> str:
    """Get user's choice of which PRD is better"""
    while True:
        print("\nWhich PRD do you prefer?")
        print("1. ChatGPT PRD")
```

```
        print("2. Claude PRD")
        print("3. Show PRDs again")
        print("4. Save both PRDs to files")

        choice = input("Enter your choice (1-4): ").strip()

        if choice == "1":
            return "chatgpt"
        elif choice == "2":
            return "claude"
        elif choice == "3":
            return "show_again"
        elif choice == "4":
            return "save_both"
        else:
            print(" Invalid choice. Please enter 1-4.")
```

While Loop Design

while True: This creates an "infinite loop" that keeps asking until the user gives a valid answer.
Why this design?:

- Users might make typos or enter invalid options
- Rather than crashing, the program politely asks again
- The loop only exits when a valid choice is made (via `return`)

.strip() method: Removes extra spaces from user input. If someone types " 1 " (with spaces), it becomes "1".

5 The Main Process Flow

5.1 Deep Dive: Understanding process_prd_generation()

This is the "conductor" method that orchestrates everything. Let's examine every line of code to understand exactly how it works:

5.1.1 Part 1: Setting Up and Getting User Input

```
def process_prd_generation(self):
    """Main method to generate PRDs"""
    print("\n PRD GENERATION SYSTEM")
    print("="*50)

    # Get new information file
    new_info_file = input("Enter path to new information file (research
   , specs, etc.): ").strip()
    if not Path(new_info_file).exists():
        print(f" File not found: {new_info_file}")
        return
```

Line-by-Line Breakdown

Line 1: `def process_prd_generation(self):`

- Defines a method that belongs to the PRDGenerationSystem class
- `self` refers to the specific instance of the class being used

Line 2: `"""Main method to generate PRDs"""`

- This is a "docstring" - documentation that explains what the function does
- Python ignores this when running, but it helps programmers understand the code

Lines 3-4: `print` statements

- `\n` creates a new line for spacing
- `"="*50` creates a decorative line of 50 equal signs
- This creates a visual header for the user

Line 6: `new_info_file = input(...).strip()`

- `input()` displays the text and waits for user to type something
- `.strip()` removes any extra spaces the user might have typed
- The result is stored in the variable `new_info_file`

Lines 7-9: File existence check

- `Path(new_info_file)` creates a path object from the user's input
- `.exists()` checks if the file actually exists on the computer
- `if not` means "if the file does NOT exist"
- `return` exits the function immediately if there's a problem

5.1.2 Part 2: Collecting Context Files

```
    # Get context PRD files (optional)
    context_files = []
    print("\n Context PRDs (optional - helps with style/format)")
    while True:
        context_file = input("Enter path to context PRD (or press Enter
    to skip): ").strip()
        if not context_file:
            break
        if Path(context_file).exists():
            context_files.append(context_file)
            print(f" Added context: {Path(context_file).name}")
        else:
            print(f" File not found: {context_file}")
```

Understanding the Context File Loop

Line 1: `context_files = []`

- Creates an empty list to store context file paths
- Think of it as an empty shopping basket that we'll fill with file names

Line 3: `while True:`

- Creates an infinite loop that keeps asking for files
- "While True" means "keep doing this forever" (until we tell it to stop)

Line 4: User input collection

- Same pattern as before: ask user, get input, remove extra spaces

Lines 5-6: Empty input check

- `if not context_file:` checks if user pressed Enter without typing anything
- `break` exits the while loop (stops asking for more files)
- This is how users signal "I'm done adding files"

Lines 7-11: File validation and storage

- Check if the file exists (same pattern as before)
- If it exists: add to our list and show success message
- If it doesn't exist: show error message but keep asking for more files
- `.append()` adds the file path to our list
- `Path(context_file).name` gets just the filename (not the full path) for display

5.1.3 Part 3: Configuration Summary and AI Generation

```
1     print(f"\  n      Configuration:")
2     print(f"  New info: {Path(new_info_file).name}")
3     print(f"  Context PRDs: {len(context_files)} files")
4
5     # Generate PRDs
6     print("\n Generating ChatGPT PRD...")
7     chatgpt_prd = self.generate_chatgpt_prd(new_info_file,
     context_files)
8
9     print(" Generating Claude PRD...")
10    claude_prd = self.generate_claude_prd(new_info_file, context_files)
```

Configuration Display and AI Calls

Lines 1-3: Configuration summary

- Shows user what files will be processed
- `Path(new_info_file).name` extracts just the filename for cleaner display
- `len(context_files)` counts how many context files were added
- This gives users a chance to verify their input before processing

Lines 5-6: ChatGPT generation

- `self.generate_chatgpt_prd(...)` calls the ChatGPT method we defined earlier
- Passes both the main file and list of context files as parameters
- Stores the result in `chatgpt_prd` variable

Lines 8-9: Claude generation

- Same pattern as ChatGPT, but calls the Claude method
- Both AI services process the same input files
- Now we have two different PRD versions to compare

5.1.4 Part 4: User Choice and File Saving Logic

```
1     # Display and get user choice
2     while True:
3         self.display_prds(chatgpt_prd, claude_prd)
4         choice = self.get_user_choice()
5
6         if choice == "show_again":
7             continue
8         elif choice == "save_both":
9             # Save both PRDs
10            base_name = Path(new_info_file).stem
11            chatgpt_file = self.save_prd_to_file(chatgpt_prd, f"{
    base_name}_chatgpt_prd.md")
12            claude_file = self.save_prd_to_file(claude_prd, f"{
    base_name}_claude_prd.md")
13
14            print("\n Both PRDs saved! You can:")
15            print("1. Review them in your file manager")
16            print("2. Edit them further")
17            break
18        else:
19            # Save chosen PRD
20            chosen_prd = chatgpt_prd if choice == "chatgpt" else
    claude_prd
21            model_name = choice
22
```

```
23              base_name = Path(new_info_file).stem
24              saved_file = self.save_prd_to_file(chosen_prd, f"{base_name
    }_final_prd.md")
25
26              if saved_file:
27                  print(f"\n PRD generation complete!")
28                  print(f" Final PRD ({model_name}): {saved_file}")
29              break
```

Understanding the Choice Logic

Another while True loop:

- Keeps showing options until user makes a final decision
- User might want to review PRDs multiple times before choosing

Lines 2-3: Display and get choice

- `self.display_prds(...)` shows both PRDs side by side
- `self.get_user_choice()` presents menu and gets user's selection

Lines 5-6: "Show again" option

- `continue` skips the rest of the loop and starts over
- Useful if PRDs are long and user wants to review again

Lines 7-15: "Save both" option

- `Path(new_info_file).stem` gets filename without extension
- Example: "market_research.pdf" becomes "market_research"
- `f"{base_name}_chatgpt_prd.md"` creates descriptive filenames
- Both PRDs are saved with clear naming convention
- `break` exits the loop after saving

Lines 16-25: "Choose one" option

- `chosen_prd = chatgpt_prd if choice == "chatgpt" else claude_prd`
- This is a "ternary operator" - a shortcut for if/else
- Reads as: "If choice is chatgpt, use chatgpt_prd, otherwise use claude_prd"
- Saves the chosen PRD with "final" in the filename
- Shows completion message with file location

5.1.5 Understanding the Control Flow

How the Function Flows

The method follows this logical sequence:

1. **Setup**: Display header and get main input file
2. **Validation**: Check if main file exists (exit if not)
3. **Context Collection**: Loop to gather optional context files
4. **Summary**: Show user what will be processed
5. **AI Generation**: Create two PRD versions simultaneously
6. **User Decision Loop**:
    - Show both PRDs
    - Get user choice
    - Handle choice (save files and exit, or show again)

Key Design Principles:

- **Fail Fast**: Check for problems early (file existence)
- **User Control**: Multiple opportunities to review and choose
- **Clear Feedback**: Status messages at each step
- **Flexible Input**: Optional context files, multiple viewing options

5.1.6 Error Handling Patterns

Defensive Programming in Action

Notice how the code handles potential problems:
File Not Found:

```
if not Path(new_info_file).exists():
    print(f" File not found: {new_info_file}")
    return
```

- Checks before trying to use the file
- Gives helpful error message
- Exits gracefully instead of crashing

Empty User Input:

```
if not context_file:
    break
```

- Handles when user presses Enter without typing
- Uses this as a signal to stop asking for more files

Invalid Context Files:

```
if Path(context_file).exists():
    context_files.append(context_file)
    print(f" Added context: {Path(context_file).name}")
else:
    print(f" File not found: {context_file}")
```

- Doesn't crash if context file doesn't exist
- Continues asking for more files
- Only adds valid files to the list

5.1.7 Variable Naming and Code Organization

Why These Variable Names?

Descriptive Names:

- `new_info_file` - clearly indicates this is the main input
- `context_files` - shows this is a collection of reference files
- `chatgpt_prd` and `claude_prd` - distinguish between AI outputs
- `base_name` - indicates filename without extension

Consistent Patterns:

- All file-related variables end with "file" or "files"
- All PRD-related variables end with "prd"
- Status messages use emoji for visual clarity

This makes the code self-documenting - you can understand what each variable does just from its name.

5.2 The Main Function Structure

```
def main():
    """Main function to run the PRD generation system"""
    system = PRDGenerationSystem()

    while True:
        print(" PRD GENERATION SYSTEM")
        print("1. Generate new PRD")
        print("2. Exit")

        choice = input("\nEnter your choice (1-2): ").strip()

        if choice == "1":
            system.process_prd_generation()
        elif choice == "2":
            print(" Goodbye!")
            break
        else:
            print(" Invalid choice. Please enter 1 or 2.")
```

Main Function Design

Why create a system object?: `system = PRDGenerationSystem()` creates an instance of our class. This gives us access to all the methods and stored data (like API keys).
Menu-driven interface: The while loop creates a simple menu system that keeps running until the user chooses to exit.
Method calling: `system.process_prd_generation()` calls the method on our specific system instance.

6 Error Handling and Robustness

6.1 Try-Except Blocks

Throughout the code, you'll see this pattern:

```
try:
    # Attempt to do something that might fail
    response = client.chat.completions.create(...)
    return response.choices[0].message.content
except Exception as e:
    return f" ChatGPT Error: {str(e)}"
```

Why Error Handling Matters

Without error handling: If the internet connection fails or the API key is wrong, the program would crash with a confusing error message.
With error handling: The program catches the error and returns a user-friendly message explaining what went wrong.
This makes the program more professional and easier to debug.

6.2 File Existence Checking

```
if not Path(new_info_file).exists():
    print(f" File not found: {new_info_file}")
    return
```

Defensive Programming

This is called "defensive programming" - anticipating what could go wrong and handling it gracefully.
Instead of trying to read a file that doesn't exist (which would cause an error), we check first and give a helpful message.

7 Advanced Concepts and Design Decisions

7.1 Why Use Classes Instead of Simple Functions?

Class vs Function Design

Alternative approach: We could have written separate functions for each task.
Why we chose a class:

- **Data organization**: API keys are stored once and used by multiple methods
- **Related functionality**: All PRD-related functions are grouped together
- **State management**: The class can remember settings and preferences
- **Extensibility**: Easy to add new AI providers or features

Think of it like organizing tools in a toolbox rather than leaving them scattered around.

7.2 API Design Patterns

```
response = client.chat.completions.create(
    model="gpt-4o",
    messages=[...],
    max_tokens=4000,
    temperature=0.3
)
```

Understanding API Parameters

model: Specifies which AI model to use (like choosing which expert to consult)
messages: The conversation history and current request
max_tokens: Limits how long the response can be (tokens are roughly equivalent to words)
temperature: Controls creativity (0.0 = very focused, 1.0 = very creative). We use 0.3 for consistent, professional output.

7.3 Data Structure Choices

```
message_content = [
    {
        "type": "text",
        "text": self.get_prd_generation_prompt()
    }
]
```

Why Lists of Dictionaries?

This structure matches what the AI APIs expect. Each item in the list represents a different part of the message:

- Text content
- Images or documents
- Special formatting instructions

It's like organizing a presentation with different types of slides (text slides, image slides, etc.).

8 Practical Usage Guide

8.1 Setting Up the System

1. **Install Python**: Download from python.org
2. **Install required packages**:

```
pip install openai anthropic pathlib

```

3. **Get API keys**:
   - ChatGPT: Sign up at platform.openai.com
   - Claude: Sign up at console.anthropic.com
4. **Update the code**: Replace the placeholder API keys with your real ones

8.2 Best Practices for Input Files

Creating Good Input Files

For new information files:

- Be specific about your product vision
- Include user research or market analysis
- Mention technical constraints or requirements
- Describe the target audience clearly

For context PRD files:

- Choose PRDs from similar products or industries
- Include examples with good structure and detail
- Vary the complexity to show different styles

8.3 Troubleshooting Common Issues

Common Problems and Solutions

Problem: "Module not found" error
Solution: Install missing packages with pip
Problem: API authentication errors
Solution: Check that your API keys are correct and active
Problem: Poor quality PRDs
Solution: Provide more detailed input files or better context examples
Problem: File path errors
Solution: Use full file paths or ensure files are in the same directory as the script

9 Extending and Customizing the System

9.1 Adding New AI Providers

To add a new AI service, you would:

1. Add the import statement
2. Add the API key to `__init__`
3. Create a new `generate_[service]_prd` method
4. Update the main process to include the new option

9.2 Customizing the PRD Template

Modify the `get_prd_generation_prompt` method to change:

- Section structure
- Writing style
- Specific requirements
- Industry focus

10 Step-by-Step Guide: Actually Using the System

Now that you understand how the code works, let's walk through exactly how to use it in real life.

10.1 Preparing Your Files

10.1.1 Finding Files on Your Computer

Before running the system, you need to know where your files are located. Here's how to find file paths on different operating systems:

Finding File Paths

On Windows:

1. Open File Explorer
2. Navigate to your file
3. Right-click on the file
4. Select "Properties"
5. Copy the path from the "Location" field
6. Add the filename to the end

Example: `C:\Users\YourName\Documents\my_product_research.pdf`

On Mac:

1. Open Finder
2. Navigate to your file
3. Right-click (or Control+click) on the file
4. Hold the Option key and select "Copy as Pathname"

Example: `/Users/YourName/Documents/my_product_research.pdf`

On Linux:

1. Open your file manager
2. Navigate to your file
3. Right-click and select "Properties" or "Copy Path"

Example: `/home/username/Documents/my_product_research.pdf`

10.1.2 Organizing Your Project Files

Recommended File Organization

Create a dedicated folder for your PRD project:

```
PRD_Project/
          prd_gen.py                # The main Python script
          input_files/              # Your source materials
                market_research.pdf
                user_interviews.docx
                technical_specs.txt
          context_prds/             # Example PRDs for reference
                competitor_prd.pdf
                previous_product_prd.pdf
          generated_prds/           # Output folder (created
   automatically)
```

This organization makes it easier to find files and reduces the chance of path errors.

10.2 Running the System: Complete Walkthrough

10.2.1 Step 1: Starting the Program

1. Open your command prompt/terminal
2. Navigate to the folder containing `prd_gen.py`
3. Run the command: `python prd_gen.py`

```
# Example commands:
cd /path/to/your/PRD_Project
python prd_gen.py

# Or on some systems:
python3 prd_gen.py
```

You should see:

```
 PRD GENERATION SYSTEM
1. Generate new PRD
2. Exit

Enter your choice (1-2):
```

10.2.2 Step 2: Providing Your Main Information File

When you select option 1, the system will ask:

```
Enter path to new information file (research, specs, etc.):
```

Example Responses

If your file is in the same folder as the script:

```
market_research.pdf
```

If your file is in a subfolder:

```
input_files/market_research.pdf
```

If your file is elsewhere on your computer:

```
C:\Users\YourName\Documents\my_research.pdf
```

Pro tip: You can often drag and drop the file into your terminal window to automatically paste the full path!

10.2.3 Step 3: Adding Context PRDs (Optional)

The system will then ask:

```
 Context PRDs (optional - helps with style/format)
Enter path to context PRD (or press Enter to skip):
```

Context PRD Strategy

What are good context PRDs?:

- PRDs from similar products in your industry
- Internal PRDs from your company (if you have them)
- Well-structured PRDs you found online
- PRDs with a style/format you want to emulate

You can add multiple context PRDs:

```
Enter path to context PRD (or press Enter to skip): context_prds/
    example1.pdf
 Added context: example1.pdf
Enter path to context PRD (or press Enter to skip): context_prds/
    example2.pdf
 Added context: example2.pdf
Enter path to context PRD (or press Enter to skip): [Press Enter
    to finish]
```

10.2.4 Step 4: The Generation Process

After providing your files, you'll see:

```
Configuration:
 New info: market_research.pdf
 Context PRDs: 2 files

Generating ChatGPT PRD...
Generating Claude PRD...
```

This process typically takes 30-60 seconds, depending on file sizes and internet speed.

10.2.5 Step 5: Reviewing and Choosing

The system will display both PRDs and ask:

```
================================================================================

 PRD COMPARISON
================================================================================


 ChatGPT PRD:
----------------------------------------
[Full ChatGPT PRD content here...]

 Claude PRD:
----------------------------------------
[Full Claude PRD content here...]

Which PRD do you prefer?
1. ChatGPT PRD
2. Claude PRD
3. Show PRDs again
4. Save both PRDs to files

Enter your choice (1-4):
```

Making Your Choice

Option 1 or 2: Choose the PRD you prefer. The system will save it as the final version.
Option 3: If the PRDs are long and you want to review them again.
Option 4: Save both versions so you can compare them later or combine elements from each.

10.3 Real-World Usage Examples

10.3.1 Example 1: Startup Founder

Scenario: You're launching a food delivery app and have market research data.

```
# Your file structure:
FoodApp_PRD/
          prd_gen.py
          market_research.pdf          # Your main input
          user_surveys.docx           # Additional input
          context_prds/
              uber_eats_prd.pdf       # Context example

# Usage:
python prd_gen.py
# Choose option 1
# Input file: market_research.pdf
# Context PRD: context_prds/uber_eats_prd.pdf
# Review results and choose best version
```

10.3.2 Example 2: Product Manager

Scenario: You need to create a PRD for a new feature based on user feedback.

```
# Your file structure:
FeaturePRD/
          prd_gen.py
          user_feedback_analysis.pdf    # Your main input
          technical_requirements.txt    # Additional details
          context_prds/
              previous_feature_prd.pdf # Your company's style
              competitor_feature.pdf    # Market reference

# Usage process:
python prd_gen.py
# Input: user_feedback_analysis.pdf
# Context 1: context_prds/previous_feature_prd.pdf
# Context 2: context_prds/competitor_feature.pdf
# Generate and compare results
```

10.4 Understanding the Output

10.4.1 Where Files Are Saved

The system automatically creates a `generated_prds/` folder and saves your PRDs there:

```
generated_prds/
          market_research_final_prd.md      # Your chosen PRD
          market_research_chatgpt_prd.md    # ChatGPT version (if you
   saved both)
          market_research_claude_prd.md     # Claude version (if you
   saved both)
```

10.4.2 Working with the Generated PRDs

Next Steps with Your PRD

The generated files are in Markdown format (.md):

- Can be opened in any text editor
- Easily converted to Word, PDF, or HTML
- Great for version control and collaboration
- Supports formatting like headers, lists, and tables

Recommended tools for editing:

- **Typora**: Beautiful Markdown editor
- **Visual Studio Code**: Free, powerful editor
- **Notion**: Convert to Notion pages
- **Google Docs**: Copy and paste content

10.5 Troubleshooting Common Issues

10.5.1 File Path Problems

Common File Path Errors

Error: "File not found"

- Check that the file path is correct
- Use forward slashes (/) or double backslashes (\\) on Windows
- Ensure there are no extra spaces in the path
- Try using the full absolute path

Error: "Permission denied"

- Make sure the file isn't open in another program
- Check that you have read permissions for the file
- Try copying the file to the same folder as the script

10.5.2 API and Network Issues

API Troubleshooting

Error: "API key invalid"

- Verify your API keys are correctly copied
- Check that your API accounts have sufficient credits
- Ensure there are no extra spaces or characters in the keys

Error: "Connection timeout"

- Check your internet connection
- Try running the script again (temporary network issues)
- Large files may take longer to process

10.6 Pro Tips for Better Results

Maximizing PRD Quality

Input File Best Practices:

- Include specific user personas and use cases
- Provide quantitative data where possible
- Mention technical constraints or requirements
- Include competitive analysis
- Be specific about success metrics

Context PRD Selection:

- Choose PRDs from similar complexity products
- Include both internal and external examples
- Select PRDs with clear structure and good detail
- Avoid PRDs that are too different from your product

Review and Editing:

- Always review the generated PRD for accuracy
- Add domain-specific details the AI might have missed
- Customize the success metrics for your specific context
- Consider combining elements from both AI versions

11 Conclusion

This PRD generation system demonstrates several important programming concepts:

- **Object-oriented programming**: Using classes to organize related functionality
- **API integration**: Communicating with external services
- **Error handling**: Making programs robust and user-friendly
- **File processing**: Working with different file types
- **User interface design**: Creating intuitive command-line interactions

The system provides a foundation that you can build upon, whether you're interested in AI integration, business process automation, or Python programming in general.

Key Takeaways

- **Modular design**: Each function has a specific, well-defined purpose
- **User-centered approach**: The interface prioritizes ease of use over technical complexity
- **Extensibility**: The structure makes it easy to add new features or AI providers
- **Robustness**: Error handling ensures the system fails gracefully
- **Practical application**: Understanding code is only the first step - knowing how to use it effectively is equally important

Remember: The goal isn't just to use this system, but to understand the thinking behind its design. These patterns and principles apply to many other programming projects you might encounter or create. More importantly, understanding both the technical implementation and practical usage gives you the complete picture of how software systems work in the real world.