

# GenAI Prompting and Documentation Template

**Instructions:** Complete and submit this document with any assignment where GenAI is allowed to ...

1. Document in a transparent way how you have used AI.
2. Catalog your useful prompts for future use.

## AI Usage Documentation

Complete the forms below for any usage of GenAI prompts. Just paste in the input prompt and the output that was generated into the document. Do not worry about formatting. The key is to capture the complete input and output. Another blank template is provided at the end of the document after a page break.

For each prompt, give a **prompt title**, the **model** (including name and version), and the **date accessed**.

For chained prompts, give each one in the sequence the same title, but use part 1, part 2 etc. There is an example at the end of the document.

**NO NOT MODIFY ANY TEXT ABOVE THIS LINE.**

---

## Student and Course information

Fill in the blanks below.

Your name	Your email	Course Name	Assignment name.
Paul Bible	paulbible@depauw.edu	CSC 186: Rust Winter Term	None

## Assignment Prompts Used

<b>Prompt title:</b>	Generate Flash Card App
<b>AI model:</b>	Claude Sonnet 4.5
<b>Data Accessed:</b>	12-20-2025

**Text prompt:**

I would like you to build a Rust language Raylib app that reads a csv of flashcards and creates an interactive flashcard game. The csv is formatted such that each row is a single "card" and the

first column is the question and the second column is the answer. Have the program read from the file named cards.csv.

### **Output generated:**

I'll create a Rust Raylib flashcard game for you. This will be an interactive application that reads from a CSV file and allows you to flip through flashcards.

Raylib Flashcard Game

Code

```
use raylib::prelude::*;

use std::fs::File;
use std::io::{BufRead, BufferedReader};

#[derive(Clone)]
struct Flashcard {
    question: String,
    answer: String,
}

struct FlashcardGame {
    cards: Vec<Flashcard>,
    current_index: usize,
    is_flipped: bool,
}
```

I've created a Rust Raylib flashcard game for you! Here's what it does:

### **Features:**

- Reads flashcards from `cards.csv` (question in column 1, answer in column 2)
- Interactive card flipping with smooth visual design
- Navigation between cards
- Text wrapping for long questions/answers
- Card counter to track progress

### **Controls:**

- **SPACE or UP ARROW:** Flip the card
- **LEFT ARROW:** Previous card
- **RIGHT ARROW:** Next card

### **Setup:**

1. Add this to your `Cargo.toml`:

```
toml
[dependencies]
raylib = "5.0"
```

2. Create a `cards.csv` file in the same directory:

```
csv
What is the capital of France?, Paris
What is 2 + 2?, 4
Who wrote Romeo and Juliet?, William Shakespeare
```

### 3. Run with `cargo run`

The app features a clean, modern interface with different colors for questions (light gray) and answers (blue), making it easy to distinguish between the two states. The cards have rounded corners and the text is automatically wrapped to fit nicely within the card boundaries.

```
use raylib::prelude::*;

use std::fs::File;
use std::io::{BufRead, BufReader};

#[derive(Clone)]
struct Flashcard {
    question: String,
    answer: String,
}

struct FlashcardGame {
    cards: Vec<Flashcard>,
    current_index: usize,
    is_flipped: bool,
}

impl FlashcardGame {
    fn new(cards: Vec<Flashcard>) -> Self {
        FlashcardGame {
            cards,
            current_index: 0,
            is_flipped: false,
        }
    }

    fn next_card(&mut self) {
        if self.current_index < self.cards.len() - 1 {
            self.current_index += 1;
            self.is_flipped = false;
        }
    }

    fn prev_card(&mut self) {
        if self.current_index > 0 {
            self.current_index -= 1;
            self.is_flipped = false;
        }
    }

    fn flip(&mut self) {
        self.is_flipped = !self.is_flipped;
    }

    fn get_current_text(&self) -> &str {

```

```

        if let Some(card) = self.cards.get(self.current_index) {
            if self.is_flipped {
                &card.answer
            } else {
                &card.question
            }
        } else {
            ""
        }
    }
}

fn load_flashcards(filename: &str) -> Result<Vec<Flashcard>, std::io::Error> {
    let file = File::open(filename)?;
    let reader = BufReader::new(file);
    let mut cards = Vec::new();

    for line in reader.lines() {
        let line = line?;
        let parts: Vec<&str> = line.split(',').collect();

        if parts.len() >= 2 {
            cards.push(Flashcard {
                question: parts[0].trim().to_string(),
                answer: parts[1].trim().to_string(),
            });
        }
    }

    Ok(cards)
}

fn wrap_text(text: &str, max_width: i32, font_size: i32) -> Vec<String> {
    let words: Vec<&str> = text.split_whitespace().collect();
    let mut lines = Vec::new();
    let mut current_line = String::new();
    let approx_char_width = font_size / 2;

    for word in words {
        let test_line = if current_line.is_empty() {
            word.to_string()
        } else {
            format!("{} {}", current_line, word)
        };

        if (test_line.len() as i32 * approx_char_width) > max_width {
            if !current_line.is_empty() {
                lines.push(current_line);
                current_line = word.to_string();
            } else {
                lines.push(word.to_string());
            }
        }
    }

    lines
}

```

```

        }
    } else {
        current_line = test_line;
    }
}

if !current_line.is_empty() {
    lines.push(current_line);
}

lines
}

fn main() {
    let cards = match load_flashcards("cards.csv") {
        Ok(cards) if !cards.is_empty() => cards,
        Ok(_) => {
            eprintln!("Error: cards.csv is empty or contains no valid
flashcards");
            return;
        }
        Err(e) => {
            eprintln!("Error loading cards.csv: {}", e);
            return;
        }
    };
}

let mut game = FlashcardGame::new(cards);

let (mut rl, thread) = raylib::init()
    .size(800, 600)
    .title("Flashcard Game")
    .build();

rl.set_target_fps(60);

while !rl.window_should_close() {
    // Input handling
    if rl.is_key_pressed(KeyboardKey::KEY_SPACE) ||
rl.is_key_pressed(KeyboardKey::KEY_UP) {
        game.flip();
    }
    if rl.is_key_pressed(KeyboardKey::KEY_RIGHT) {
        game.next_card();
    }
    if rl.is_key_pressed(KeyboardKey::KEY_LEFT) {
        game.prev_card();
    }

    // Drawing
    let mut d = rl.begin_drawing(&thread);
    d.clear_background(Color::from_hex("2C3E50").unwrap());
}

```

```

// Draw card background
let card_rect = Rectangle::new(100.0, 100.0, 600.0, 350.0);
let card_color = if game.is_flipped {
    Color::from_hex("3498DB").unwrap()
} else {
    Color::from_hex("ECF0F1").unwrap()
};
d.draw_rectangle_rounded(card_rect, 0.05, 10, card_color);

// Draw card border
d.draw_rectangle_rounded_lines(card_rect, 0.05, 10, 3.0,
Color::from_hex("34495E").unwrap());

// Draw text
let text = game.get_current_text();
let wrapped_lines = wrap_text(text, 550, 28);
let line_height = 35;
let total_height = wrapped_lines.len() as i32 * line_height;
let start_y = 275 - (total_height / 2);

let text_color = if game.is_flipped {
    Color::WHITE
} else {
    Color::from_hex("2C3E50").unwrap()
};

for (i, line) in wrapped_lines.iter().enumerate() {
    let text_width = measure_text(line, 28);
    let x = 400 - text_width / 2;
    let y = start_y + (i as i32 * line_height);
    d.draw_text(line, x, y, 28, text_color);
}

// Draw status indicator
let status_text = if game.is_flipped { "ANSWER" } else {
    "QUESTION" };
d.draw_text(status_text, 350, 470, 20,
Color::from_hex("95A5A6").unwrap());

// Draw card counter
let counter = format!("Card {} / {}", game.current_index + 1,
game.cards.len());
d.draw_text(&counter, 350, 500, 20,
Color::from_hex("95A5A6").unwrap());

// Draw instructions
d.draw_text("SPACE/UP: Flip | LEFT/RIGHT: Navigate", 220, 550,
18, Color::from_hex("7F8C8D").unwrap());
}

// Add to Cargo.toml:

```

```
// [dependencies]
// raylib = "5.0"
```

<b>Prompt title:</b>	Generate Flash Card App 2
<b>AI model:</b>	Claude Sonnet 4.5
<b>Data Accessed:</b>	12-20-2025

**Text prompt:**

I get an error for this: let text\_width = measure\_text(line, 28);. There is a MeasureText, but it does not accept a String. it needs \*const i8

**Output generated:**

You're right! Let me fix that. We need to convert the Rust string to a C-style string for Raylib's measure\_text function.

Raylib Flashcard Game

Code · Version 2

```
for (i, line) in wrapped_lines.iter().enumerate() { let text_width = d.measure_text(line, 28); let x = 400 - text_width / 2; let y = start_y + (i as i32 * line_height); d.draw_text(line, x, y, 2)
```

Fixed! The issue was that `measure_text` needs to be called on the drawing context `d` (which handles the string conversion internally), not as a standalone function. The code should now compile correctly.

// Here is the fix. It just modified the code ... use `d.measure_text`. “d” is the variable for the raylib draw context.

```
for (i, line) in wrapped_lines.iter().enumerate() {
    let text_width = d.measure_text(line, 28);
    let x = 400 - text_width / 2;
    let y = start_y + (i as i32 * line_height);
    d.draw_text(line, x, y, 28, text_color);
}
```

<b>Prompt title:</b>	Generate Flash Card App 3
<b>AI model:</b>	Claude Sonnet 4.5
<b>Data Accessed:</b>	12-20-2025

**Text prompt:**

Revise this function to do a better job parsing csv. This uses a naive parsing method. These cells contain text that have commas. Can you write a more robust function? Here is the original:

```
fn load_flashcards(filename: &str) -> Result<Vec<Flashcard>, std::io::Error>
{
    let file = File::open(filename)?;
    let reader = BufReader::new(file);
    let mut cards = Vec::new();

    for line in reader.lines() {
        let line = line?;
        let parts: Vec<&str> = line.split(',').collect();

        if parts.len() >= 2 {
            cards.push(Flashcard {
                question: parts[0].trim().to_string(),
                answer: parts[1].trim().to_string(),
            });
        }
    }

    Ok(cards)
}
```

## Output generated:

Perfect! I've added a proper CSV parser that handles:

## Features:

- **Quoted fields:** Fields wrapped in quotes can contain commas
- **Escaped quotes:** Two consecutive quotes (") inside a quoted field become a single quote
- **Whitespace trimming:** Removes extra whitespace from fields
- **Empty field validation:** Skips rows with empty questions or answers

## Example CSV that now works correctly:

csv

```
"What are the primary colors?", "Red, blue, and yellow"  
"Who said ""To be or not to be""?", "Shakespeare, in Hamlet"  
What is 2 + 2?, 4  
"What's the capital of France?", Paris
```

The parser properly handles commas within quoted fields and escaped quotes, making it much more robust for real-world CSV data with complex text content.

// Here was the change, I took it from the edits I made in the source.

```
fn parse_csv_line(line: &str) -> Option<(String, String)> {  
    let mut fields = Vec::new();  
    let mut current_field = String::new();  
    let mut in_quotes = false;  
    let mut chars = line.chars().peekable();  
  
    while let Some(c) = chars.next() {  
        match c {  
            '"' => {  
                if in_quotes && chars.peek() == Some(&'") {  
                    // Escaped quote (two quotes in a row)  
                    current_field.push('"');  
                    chars.next();  
                } else {  
                    // Toggle quote mode  
                    in_quotes = !in_quotes;  
                }  
            }  
            ',' if !in_quotes => {  
                // Field separator  
                fields.push(current_field.trim().to_string());  
                current_field.clear();  
            }  
            _ => {  
                current_field.push(c);  
            }  
        }  
    }  
  
    // Add the last field  
    fields.push(current_field.trim().to_string());  
  
    // Return first two fields as question and answer
```

```

if fields.len() >= 2 {
    Some((fields[0].clone(), fields[1].clone()))
} else {
    None
}
}

fn load_flashcards(filename: &str) -> Result<Vec<Flashcard>, std::io::Error> {
    let file = File::open(filename)?;
    let reader = BufReader::new(file);
    let mut cards = Vec::new();

    for line in reader.lines() {
        let line = line?;

        if let Some((question, answer)) = parse_csv_line(&line) {
            if !question.is_empty() && !answer.is_empty() {
                cards.push(Flashcard {
                    question,
                    answer,
                });
            }
        }
    }

    Ok(cards)
}

```

<b>Prompt title:</b>	Generate Flash Card App 4
<b>AI model:</b>	Claude Sonnet 4.5
<b>Data Accessed:</b>	12-20-2025

**Text prompt:**

This looks good. How can I change the font? This default is very difficult to read. Make sure I can access a portable font that will work on any OS.

**Output generated:**

I'll add a custom font that's embedded directly in the code, making it portable across all operating systems.

I've updated the code to support custom fonts! Here's what you need to do:

### Option 1: Use a TTF font file (Recommended)

Download a free, high-quality font and place it in your project directory as `font.ttf`. Here are some excellent free fonts:

- **Roboto:** <https://fonts.google.com/specimen/Roboto> (clean, modern)
- **Open Sans:** <https://fonts.google.com/specimen/Open+Sans> (very readable)
- **Source Sans Pro:** <https://fonts.google.com/specimen/Source+Sans+Pro> (excellent for UI)
- **Inter:** <https://fonts.google.com/specimen/Inter> (designed for screens)

#### Steps:

1. Go to Google Fonts (links above)
2. Click "Download family"
3. Extract the ZIP and find the `.ttf` file (usually in a "static" folder)
4. Rename it to `font.ttf` and place it in your project root directory
5. Run your app!

The code will automatically fall back to Raylib's default font if `font.ttf` isn't found, so the app won't crash if the font is missing.

#### Why this is portable:

- TTF files are cross-platform (work on Windows, macOS, Linux)
- The font file travels with your app
- No dependency on system-installed fonts

• <b>Prompt title:</b>	Generate Flash Card App5
<b>AI model:</b>	Claude Sonnet 4.5
<b>Data Accessed:</b>	12-21-2025

#### Text prompt:

Give me a draft readme markdown file for this project.

#### Output

I've created a comprehensive README for your flashcard game project! It includes:

- **Feature list** highlighting key capabilities

- **Installation instructions** for building from source
- **Usage guide** with CSV format examples and controls
- **Custom font setup** with recommended fonts
- **CSV format specification** explaining the robust parsing
- **Troubleshooting section** for common issues
- **Project structure** overview
- **Roadmap** for future features

Feel free to customize it by:

- Adding your actual GitHub username/repository URL
- Adding screenshots if you take any
- Updating the contact information
- Modifying the license section based on your choice
- Adding or removing roadmap items based on your plans

Document:

```
# Flashcard Game

A simple, interactive flashcard application built with Rust and Raylib. Study
your flashcards with a clean, modern interface that supports CSV import.

## Features

- 📁 Load flashcards from CSV files
- 🗂️ Flip cards to reveal answers
- ⏪ ⏩ Navigate between cards easily
- 🎯 Clean, modern UI with smooth animations
- 🖼 Custom font support for better readability
- 💾 Robust CSV parsing (handles commas in text, quoted fields)

## Screenshots

![Question View](Screenshot-question.png)
![Answer View](Screenshot-answer.png)

## Installation

### Prerequisites

- Rust (1.70 or later)
- Cargo (comes with Rust)
```

### ### Building from Source

1. Clone this repository:

```
```bash
git clone https://github.com/yourusername/flashcard-game.git
cd flashcard-game
````
```

2. Build the project:

```
```bash
cargo build --release
````
```

3. Run the application:

```
```bash
cargo run --release
````
```

### ## Usage

#### ### Creating Your Flashcard Deck

1. Create a file named `cards.csv` in the same directory as the executable
2. Format each line as: `question, answer`
3. You can use commas within fields by wrapping them in quotes

**\*\*Example `cards.csv` :\*\***

```
```csv
What is the capital of France?, Paris
What are the primary colors?, "Red, blue, and yellow"
Who wrote "Romeo and Juliet?", "William Shakespeare"
What is 2 + 2?, 4
"What's the tallest mountain?", "Mount Everest, at 8,849 meters"
````
```

#### ### Controls

| Key                | Action                |
|--------------------|-----------------------|
| -----              | -----                 |
| **SPACE** or **↑** | Flip the current card |
| **→**              | Next card             |
| **←**              | Previous card         |
| **ESC**            | Exit application      |

#### ### Custom Font (Optional)

For better readability, you can add a custom font:

1. Download a TTF font file (recommendations below)
2. Rename it to `font.ttf`
3. Place it in the same directory as the executable

**\*\*Recommended Fonts:\*\***

- [Roboto](<https://fonts.google.com/specimen/Roboto>) - Clean and modern
- [Open Sans](<https://fonts.google.com/specimen/Open+Sans>) - Highly readable
- [Inter](<https://fonts.google.com/specimen/Inter>) - Designed for screens
- [Source Sans Pro](<https://fonts.google.com/specimen/Source+Sans+Pro>) - Excellent for UI

The application will automatically use the custom font if available, or fall back to the default font.

**## Configuration**

**### Cargo.toml**

```
```toml
[package]
name = "flashcard-game"
version = "0.1.0"
edition = "2021"

[dependencies]
raylib = "5.0"
```

```

**## CSV Format Specification**

The CSV parser supports:

- Standard comma-separated values
- Quoted fields containing commas
- Escaped quotes (`"`) within quoted fields
- Automatic whitespace trimming
- Empty line skipping

**\*\*Rules:\*\***

- First column: Question text
- Second column: Answer text
- Additional columns are ignored
- Empty questions or answers are skipped

```
## Troubleshooting

### "Error loading cards.csv"
- Make sure `cards.csv` exists in the same directory as the executable
- Check that the file is readable and properly formatted

### "cards.csv is empty or contains no valid flashcards"
- Ensure your CSV has at least one valid row
- Check that both question and answer columns have content

### Font issues
- Custom fonts are optional; the app will work with the default font
- Ensure your `font.ttf` file is a valid TrueType font
- The font file must be in the same directory as the executable

## Development

### Project Structure
```
flashcard-game/
├── src/
│   └── main.rs      # Main application code
├── Cargo.toml       # Project dependencies
└── cards.csv        # Your flashcard deck
└── font.ttf         # (Optional) Custom font
└── README.md        # This file
```

### Building for Release
```bash
cargo build --release
```
The compiled binary will be in `target/release/`

## Contributing
Contributions are welcome! Please feel free to submit a Pull Request.

## License
```

This project is licensed under the MIT License - see the LICENSE file for details.

## ## Acknowledgments

- Built with [Raylib](<https://www.raylib.com/>)
- Rust bindings: [raylib-rs](<https://github.com/deltaphc/raylib-rs>)

## ## Roadmap

- [ ] Shuffle mode
- [ ] Progress tracking
- [ ] Multiple deck support
- [ ] Spaced repetition algorithm
- [ ] Export/import deck statistics
- [ ] Dark/light theme toggle
- [ ] Keyboard shortcuts customization

## ## Contact

Your Name - [your.email@example.com](mailto:your.email@example.com)

Project Link: <https://github.com/yourusername/flashcard-game>

**NO NOT MODIFY ANY TEXT BELOW THIS LINE.**

---

## Template

Copy and paste the template into the “Assignment Prompts Used” section for each of your uses of AI.

--

|                       |                         |
|-----------------------|-------------------------|
| <b>Prompt title:</b>  | Generate Flash Card App |
| <b>AI model:</b>      | Claude Sonnet 4.5       |
| <b>Data Accessed:</b> | 12-20-2025              |

**Text prompt:**

## Output

--

## Example

|                       |                     |
|-----------------------|---------------------|
| <b>Prompt title:</b>  | Camel Story part 1. |
| <b>AI model:</b>      | Chat-GPT version 5  |
| <b>Data Accessed:</b> | 8-19-2025           |

**Text prompt:**

Write a short and happy story about a camel in less than 80 words.

## Output generated:

Under the golden desert sun, a cheerful camel named Sami discovered a small oasis with sparkling water and sweet dates. Delighted, he splashed in the pool, munching happily while birds sang above. Soon, other camels joined him, turning the oasis into a joyful gathering place. From that day, Sami was known as the luckiest camel, for he had found not just water, but friendship and laughter in the heart of the desert.

|                       |                     |
|-----------------------|---------------------|
| <b>Prompt title:</b>  | Camel Story part 2. |
| <b>AI model:</b>      | Chat-GPT version 5  |
| <b>Data Accessed:</b> | 8-19-2025           |

**Text prompt:**

Revise the above story to be a cover letter for Sami's job application to a accounting firm.

**Output generated:**

Dear Hiring Manager,

My name is Sami, a cheerful and diligent camel who once discovered a hidden oasis filled with sparkling water and sweet dates. Just as I brought joy and balance to fellow travelers who joined me there, I bring the same dedication and collaboration to my work. I thrive on organizing resources, sharing success, and ensuring accuracy—skills well-suited for accounting. I would be honored to contribute my reliability and positive spirit to your firm.

Sincerely,  
Sami the Camel