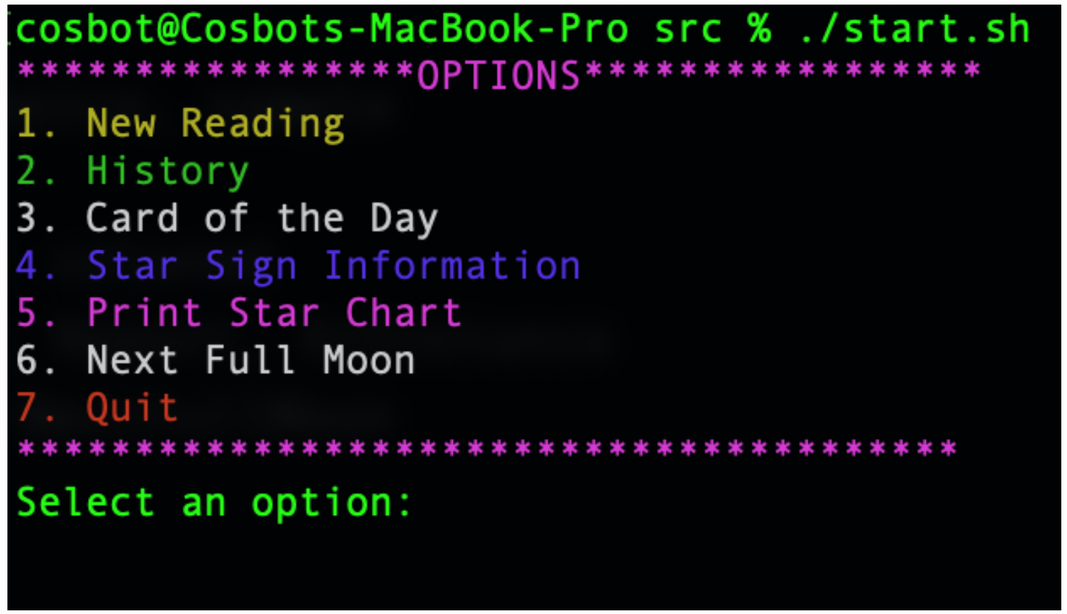
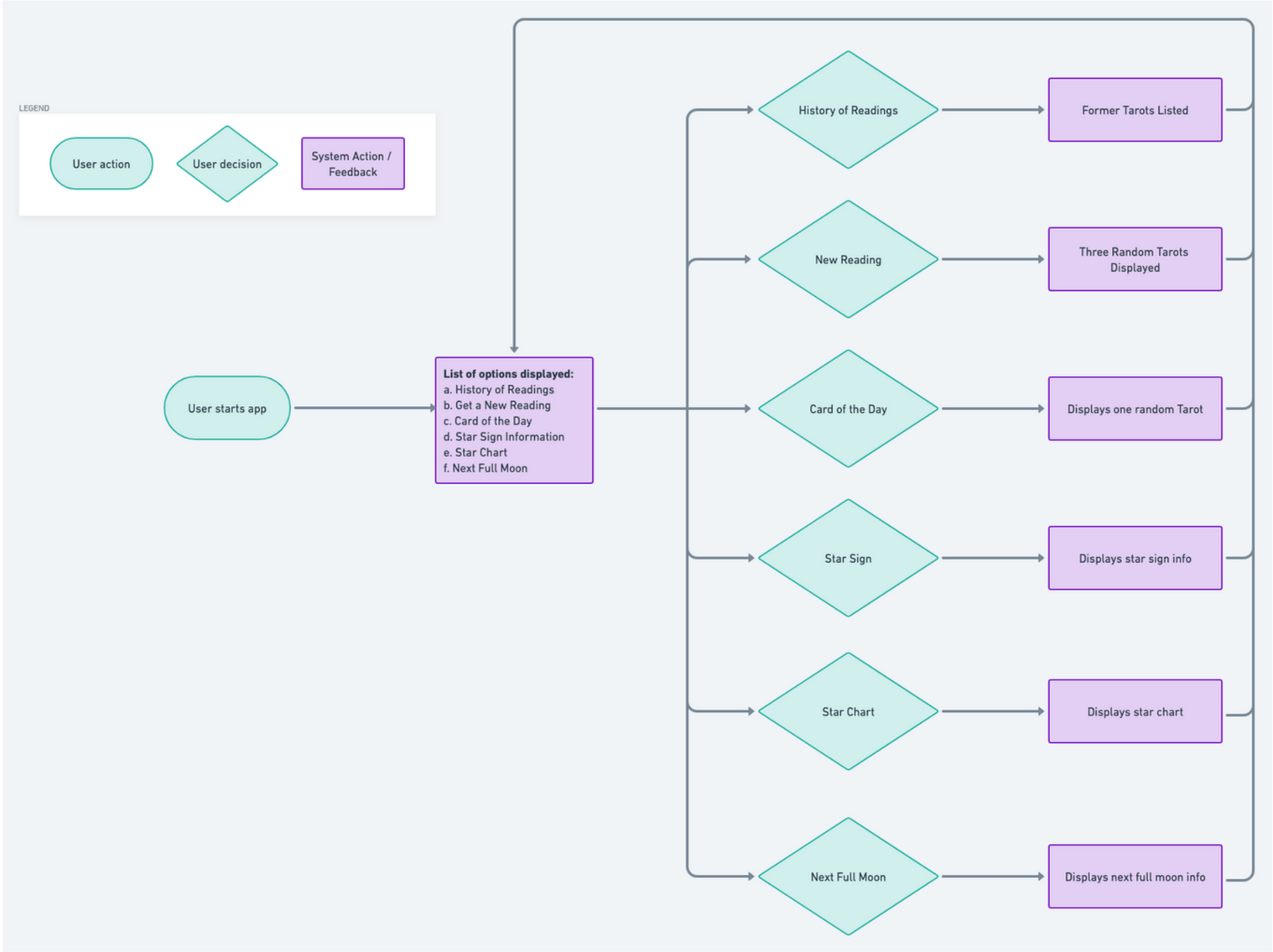
TABLE OF CONTENTS



WHY?

Terminal apps are an essential tool for any developer as they're excellent way to automate tasks, run tests, install packages, and sometimes systems won't have a GUI and the only way to get things done is by typing commands.

DESIGN & DEVELOPMENT PROCESS

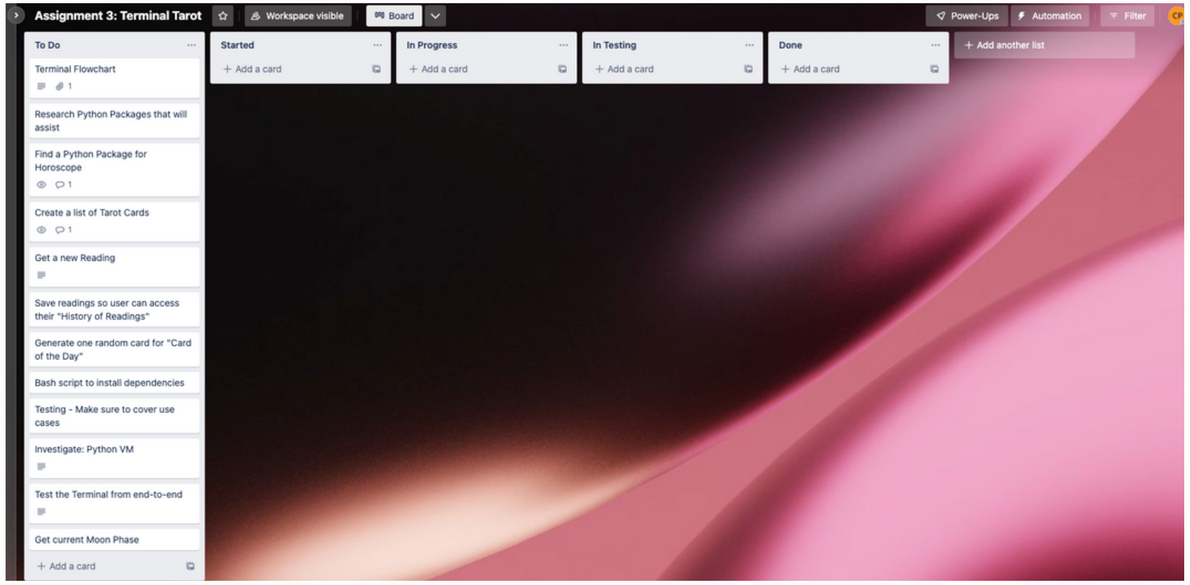


Idea: Tarot is expensive and difficult to find. So this app will have a few options for a user to obtain a reading, get card of the day, the next full moon, some horoscope information, and their star chart.

Firstly, I created a flow chart of how I thought the app would operate.

This would help me plan out what kinds of tools I'd need plus how the app would flow from start up, input, result, and display.

DESIGN & DEVELOPMENT PROCESS

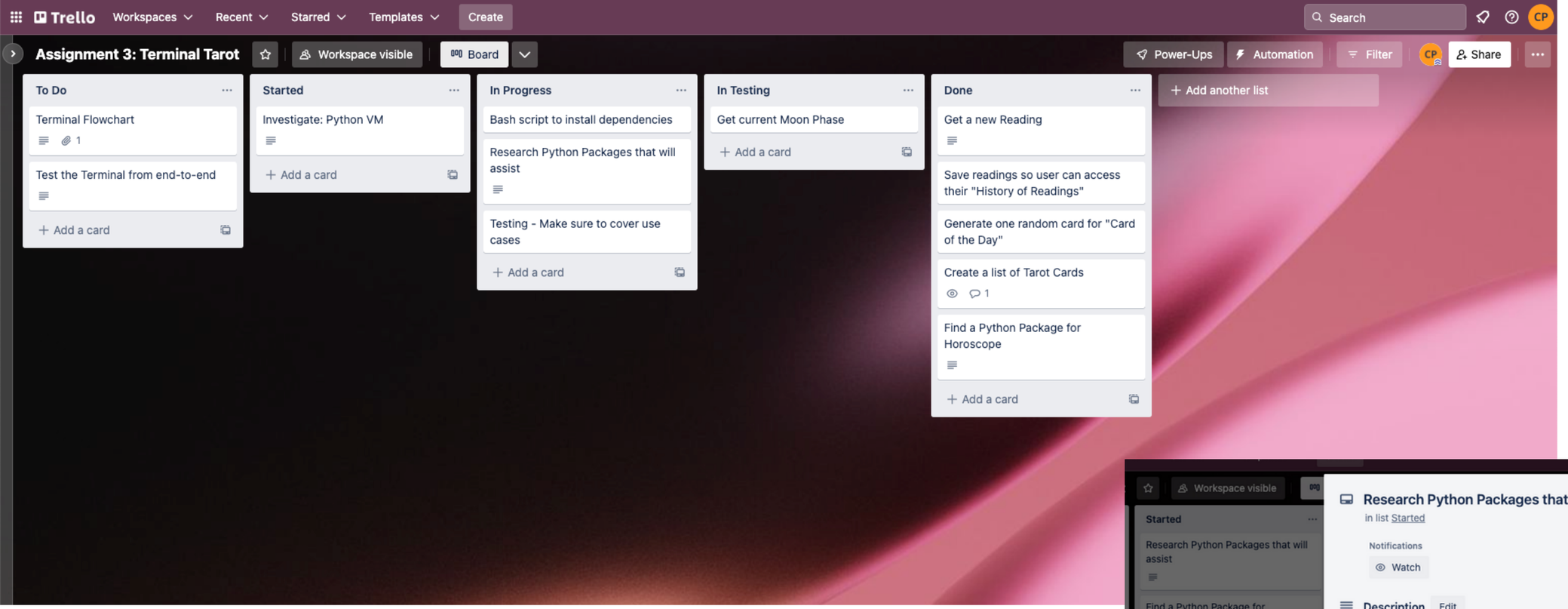


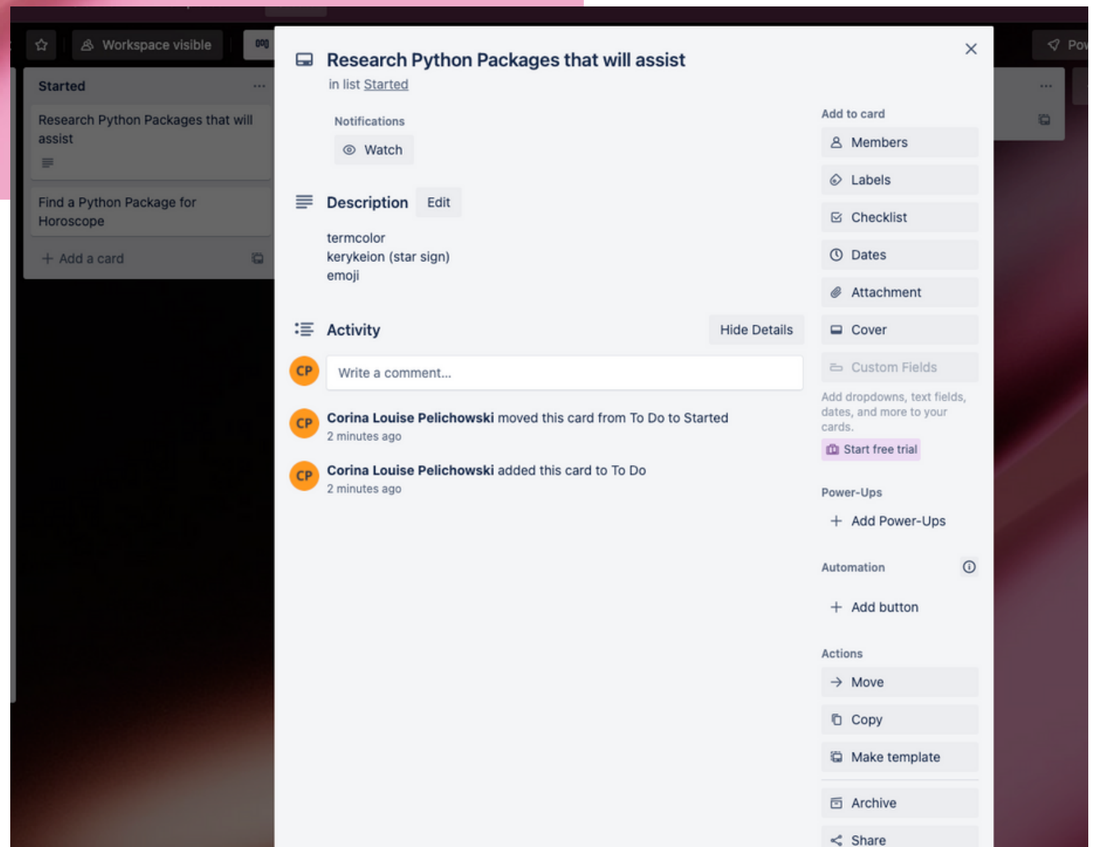
Next I used Trello to break down each feature and task, keep track of progress, and add any relevant notes/resources to each card.

Some tasks were easier than others while some were related. For example: Get a reading and get the history of a reading were related because once a reading was done, it needed to be appended to the history option so a user can recall it for later.

I used a similar workflow process that we do at work by having a "To-Do", "Started", "In Progress", "In Testing" and "Done" tabs. As it was just me working on this project, there was no option for "Peer Review".

The main difference between the "Started" column and "In Progress" column was that for "Started" I had a brief idea or a useful link to get that ticket started. For "In Progress" meant that I had begun the actual code.


Trello
Workspaces
Recent
Starred
Templates
Create
Search
CP
Assignment 3: Terminal Tarot
Workspace visible
Board
Power-Ups
Automation
Filter
Share
To Do
Terminal Flowchart
1
Test the Terminal from end-to-end
Add a card
Started
Investigate: Python VM
Add a card
In Progress
Bash script to install dependencies
Research Python Packages that will assist
Testing - Make sure to cover use cases
Add a card
In Testing
Get current Moon Phase
Add a card
Done
Get a new Reading
Save readings so user can access their "History of Readings"
Generate one random card for "Card of the Day"
Create a list of Tarot Cards
1
Find a Python Package for Horoscope
Add a card
Add another list


Workspace visible
Started
Research Python Packages that will assist
Find a Python Package for Horoscope
Add a card
Research Python Packages that will assist
in list Started
Notifications
Watch
Description
Edit
termcolor
kerykeion (star sign)
emoji
Activity
Hide Details
Write a comment...
Corina Louise Pelichowski moved this card from To Do to Started
2 minutes ago
Corina Louise Pelichowski added this card to To Do
2 minutes ago
Add to card
Members
Labels
Checklist
Dates
Attachment
Cover
Custom Fields
Add dropdowns, text fields, dates, and more to your cards.
Start free trial
Power-Ups
Add Power-Ups
Automation
Add button
Actions
Move
Copy
Make template
Archive
Share

OPTION 1: NEW READING

For the user to get a new reading, there had to be a some kind of Tarot card reference/storage.

I chose list because I wanted three random cards to be selected and returned and this was the easiest and simplest solution.

I created a class option so that when the user inputs the integer 1, it will kick off two functions. One function selected three random cards from the list and the second function returned those cards to the user.

I split it up in this way because it made for easier reading and testing. I was able to write tests for getting the cards and for returning the cards.

Furthermore, with the function of getting three random cards; I was able to slip in another command self.history.append which saved the reading to the user's history (Option 2).

```
class Option:
    NEW_READING = 1
    HISTORY = 2
    CARD_OF_THE_DAY = 3
    STAR_SIGN = 4
    STAR_CHART = 5
    FULL_MOON = 6
    QUIT = 7
```

```
def print_options(self):
    self.print_divider(
        TerminalApp.DIVIDER_LEN, "OPTIONS", '*', 'magenta')
    print(colored("1. New Reading", 'yellow'))
    print(colored("2. History", 'green'))
    print(colored("3. Card of the Day", 'white'))
    print(colored("4. Star Sign Information", 'blue'))
    print(colored("5. Print Star Chart", 'magenta'))
    print(colored("6. Next Full Moon", 'white'))
    print(colored("7. Quit", 'red'))
    self.print_divider(
        TerminalApp.DIVIDER_LEN, divider='*', color='magenta')
```

```
class TerminalApp:
    NUMBER_OF_CARDS_IN_READING = 3
    DAY_RANGE = (1, 32)
    MONTH_RANGE = (1, 13)
    YEAR_RANGE = (1920, 2030)
    OPTION_RANGE = (1, 8)
    DIVIDER_LEN = 20
```

```
# Get reading and append to History
def get_reading(self):
    reading_cards = sample(
        (self.deck), TerminalApp.NUMBER_OF_CARDS_IN_READING)
    self.history.append(reading_cards)
    return reading_cards

# Show the reading
def print_reading(self):
    reading_cards = self.get_reading()
    for card in reading_cards:
        print(card)
```

```
****************************************
Select an option: 1
The Tower: Total destruction. Prepare for things to be levelled and dismantled.
Temperance: Hold back from quick judgment. Resist the temptation to react immediately.
The Moon: Trust your intuition and push forward, you will receive deliverance from what binds you.
*****************OPTIONS*****************
1. New Reading
2. History
3. Card of the Day
4. Star Sign Information
5. Print Star Chart
6. Next Full Moon
7. Quit
****************************************
Select an option:
```

OPTION 2: READING HISTORY

For the user to get their reading history, I added self.history.append which saved the reading to the session's history.

At the moment, since there is no login option; it is just history in general and not pertaining to an individual. But this can be an option for future development.

I also wanted to have a clear display of the reading history so I used the code on the right image to modify the output.

Each reading history has a title, the three cards and a divider.

```
class Option:
    NEW_READING = 1
    HISTORY = 2
    CARD_OF_THE_DAY = 3
    STAR_SIGN = 4
    STAR_CHART = 5
    FULL_MOON = 6
    QUIT = 7
```

```
def print_options(self):
    self.print_divider(
        TerminalApp.DIVIDER_LEN, "OPTIONS", '*', 'magenta')
    print(colored("1. New Reading", 'yellow'))
    print(colored("2. History", 'green'))
    print(colored("3. Card of the Day", 'white'))
    print(colored("4. Star Sign Information", 'blue'))
    print(colored("5. Print Star Chart", 'magenta'))
    print(colored("6. Next Full Moon", 'white'))
    print(colored("7. Quit", 'red'))
    self.print_divider(
        TerminalApp.DIVIDER_LEN, divider='*', color='magenta')
```

```
class TerminalApp:
    NUMBER_OF_CARDS_IN_READING = 3
    DAY_RANGE = (1, 32)
    MONTH_RANGE = (1, 13)
    YEAR_RANGE = (1920, 2030)
    OPTION_RANGE = (1, 8)
    DIVIDER_LEN = 20
```

```
# Get a history of readings
def get_history(self):
    return self.history

# Show a history of readings
def print_history(self):
    history = self.get_history()
    for i, val in enumerate(history):
        print("Reading {}".format(i + 1))
        for j, card in enumerate(val):
            print("Card {}: {}".format(j + 1, card))
        self.print_divider(TerminalApp.DIVIDER_LEN)
```

```
****************************************
Select an option: 2
Reading 1
Card 1: The Empress: Connect with your feminine side through creativity, elegance, sensuality, fertility, and nurturing.
Card 2: The Tower: Total destruction. Prepare for things to be levelled and dismantled.
Card 3: The Emperor: You have to pursue your goals similar to the way that the Emperor does, structured, strategically, and with a lot of pers
everance.
----------------------------------------
Reading 2
Card 1: Strength: You can obtain true control of a situation, and not simply the power to forcefully impose your will.
Card 2: Devil: Redirect your attention away from the satisfaction of desires and toward the things that really matter.
Card 3: Suit of Wands: Fire - Bring attention to your goals and dreams.
----------------------------------------
Reading 3
Card 1: Wheel of Fortune: A chance to discover more about yourself and to listen to your gut. There is nothing to fear.
Card 2: The Empress: Connect with your feminine side through creativity, elegance, sensuality, fertility, and nurturing.
Card 3: The Hierophant: Time to learn the necessary values from a trusted source such as a mentor or a spiritual counselor.
----------------------------------------
Reading 4
Card 1: Suit of Pentacles: Earth - Bring attention to your finances.
Card 2: Suit of Swords: Air - Bring attention to your communication.
Card 3: The Hierophant: Time to learn the necessary values from a trusted source such as a mentor or a spiritual counselor.
----------------------------------------
****************OPTIONS****************
```

OPTION 3: CARD OF THE DAY

I actually started off with this one first. It was the most simple, basic form of what I wanted the app to do: Take an input from a user and return something.

Once I understood what was happening and how to do it; I was able to build the rest of the app.

However, I made "Get a Reading" the first option as that's better user experience. People use Tarot to get a reading; it's rarely a random card of the day.

I used the choice() method as it returns a randomly selected element from the specified sequence. Meaning that I was able to return a random card from the list inside deck.py

```
class Option:
    NEW_READING = 1
    HISTORY = 2
    CARD_OF_THE_DAY = 3
    STAR_SIGN = 4
    STAR_CHART = 5
    FULL_MOON = 6
    QUIT = 7
```

```
def print_options(self):
    self.print_divider(
        TerminalApp.DIVIDER_LEN, "OPTIONS", '*', 'magenta')
    print(colored("1. New Reading", 'yellow'))
    print(colored("2. History", 'green'))
    print(colored("3. Card of the Day", 'white'))
    print(colored("4. Star Sign Information", 'blue'))
    print(colored("5. Print Star Chart", 'magenta'))
    print(colored("6. Next Full Moon", 'white'))
    print(colored("7. Quit", 'red'))
    self.print_divider(
        TerminalApp.DIVIDER_LEN, divider='*', color='magenta')
```

```
# Get a random card from the list
def get_card_of_the_day(self):
    return choice(self.deck)

# Show the random card from the list
def print_card_of_the_day(self):
    day_card = self.get_card_of_the_day()
    print(day_card)
```

```
**********OPTIONS**********
1. New Reading
2. History
3. Card of the Day
4. Star Sign Information
5. Print Star Chart
6. Next Full Moon
7. Quit
**************************************
Select an option: 3
The Empress: Connect with your feminine side through creativity, elegance, sensuality, fertility, and nurturing.
**********OPTIONS**********
```

OPTION 4: STAR SIGN

I used the python package Kerykeion for this part. It is a python library for Astrology and that has many features.

I was able to utilise Day and Month inputs to return the user's astrological sign, the emoji, and the associated element.

One of the many interesting things I've learned about code is that with control flow, not everything reads from top to bottom, left to right.

So while the get_star_sign function is placed above the print_star_sign function, it s the second one that's gathering all the data for the first function.

Let's take a closer look.

```
class Option:
    NEW_READING = 1
    HISTORY = 2
    CARD_OF_THE_DAY = 3
    STAR_SIGN = 4
    STAR_CHART = 5
    FULL_MOON = 6
    QUIT = 7
```

```
def print_options(self):
    self.print_divider(
        TerminalApp.DIVIDER_LEN, "OPTIONS", '*', 'magenta')
    print(colored("1. New Reading", 'yellow'))
    print(colored("2. History", 'green'))
    print(colored("3. Card of the Day", 'white'))
    print(colored("4. Star Sign Information", 'blue'))
    print(colored("5. Print Star Chart", 'magenta'))
    print(colored("6. Next Full Moon", 'white'))
    print(colored("7. Quit", 'red'))
    self.print_divider(
        TerminalApp.DIVIDER_LEN, divider='*', color='magenta')
```

```
# Get star sign
def get_star_sign(self, day, month):
    info = KrInstance("User", month=month, day=day).sun
    return info

# Show star sign
def print_star_sign(self):
    # This will store the star sign info after we get user input
    star_sign = None
    try:
        # User inputs a number to select a day
        day_input = self.get_int_input_from_terminal(
            "Day: ", TerminalApp.DAY_RANGE)
        # User inputs a number to select a month
        month_input = self.get_int_input_from_terminal(
            "Month: ", TerminalApp.MONTH_RANGE)
        # Get star sign based on user's input for day and month
        star_sign = self.get_star_sign(
            day_input,
            month_input
        )
    except ValueError as e:
        print(e)
    # Show star sign info to user
    print("Information")
    self.print_divider(TerminalApp.DIVIDER_LEN)
    print("Sign: {} {} {}".format(
        star_sign["sign"], star_sign["emoji"], star_sign["element"]))
```

```
    # Get star sign
    def get_star_sign(self, day, month):
        info = KrInstance("User", month=month, day=day).sun
        return info

    # Show star sign
    def print_star_sign(self):
        # This will store the star sign info after we get user input
        star_sign = None
        try:
            # User inputs a number to select a day
            day_input = self.get_int_input_from_terminal(
                "Day: ", TerminalApp.DAY_RANGE)
            # User inputs a number to select a month
            month_input = self.get_int_input_from_terminal(
                "Month: ", TerminalApp.MONTH_RANGE)
            # Get star sign based on user's input for day and month
            star_sign = self.get_star_sign(
                day_input,
                month_input
            )
        except ValueError as e:
            print(e)
        # Show star sign info to user
        print("Information")
        self.print_divider(TerminalApp.DIVIDER_LEN)
        print("Sign: {} {} {}".format(
            star_sign["sign"], star_sign["emoji"], star_sign["element"]))

```

```
class TerminalApp:
    NUMBER_OF_CARDS_IN_READING = 3
    DAY_RANGE = (1, 32)
    MONTH_RANGE = (1, 13)
    YEAR_RANGE = (1920, 2030)
    OPTION_RANGE = (1, 8)
    DIVIDER_LEN = 20
```

OPTION 5: STAR CHART

Another functionality of the Kerykeion was to print out a star chart based on the input of Day, Month, and Year. There are other variables that a user can have (such as time and location) but for simplicity sake, I went with the first three options.

There were three ways to print out the star chart: a detailed and colourful SVG file, a report table and a text output. As a front-end developer; I'm all about that user experience. I didn't want my users to have to download or have to purchase another piece of software use my app.

So the SVG file was out. In keeping with the overall theme, I went for the report table. It's simple and easy to view.

```
class Option:
    NEW_READING = 1
    HISTORY = 2
    CARD_OF_THE_DAY = 3
    STAR_SIGN = 4
    STAR_CHART = 5
    FULL_MOON = 6
    QUIT = 7
```

```
class TerminalApp:
    NUMBER_OF_CARDS_IN_READING = 3
    DAY_RANGE = (1, 32)
    MONTH_RANGE = (1, 13)
    YEAR_RANGE = (1920, 2030)
    OPTION_RANGE = (1, 8)
    DIVIDER_LEN = 20
```

```
def print_options(self):
    self.print_divider(
        TerminalApp.DIVIDER_LEN, "OPTIONS", '*', 'magenta')
    print(colored("1. New Reading", 'yellow'))
    print(colored("2. History", 'green'))
    print(colored("3. Card of the Day", 'white'))
    print(colored("4. Star Sign Information", 'blue'))
    print(colored("5. Print Star Chart", 'magenta'))
    print(colored("6. Next Full Moon", 'white'))
    print(colored("7. Quit", 'red'))
    self.print_divider(
        TerminalApp.DIVIDER_LEN, divider='*', color='magenta')
```

```
120     # Get star chart
121     def get_star_chart(self, year, month, day):
122         star_report = KrInstance("User", day=day, month=month, year=year)
123         return star_report
124
125
126     # Show star chart
127     def print_star_chart(self):
128         # This will store the star chart info after we get user input
129         star_chart = None
130         try:
131             # User inputs a number to select a day
132             day_input = self.get_int_input_from_terminal(
133                 "Day: ", TerminalApp.DAY_RANGE)
134             # User inputs a number to select a month
135             month_input = self.get_int_input_from_terminal(
136                 "Month: ", TerminalApp.MONTH_RANGE)
137                 # User inputs a number to select a year
138             year_input = self.get_int_input_from_terminal(
139                 "Year: ", TerminalApp.YEAR_RANGE)
140             # Get star chart based on user's input for day, month, and year
141             star_chart = self.get_star_chart(
142                 year_input,
143                 month_input,
144                 day_input
145             )
146         except ValueError as e:
147             print(e)
148         report = Report(star_chart)
149         report.print_report()
150
```

OPTION 6: FULL MOON

I used another python package called "fullmoon".

The description of the package is as follows:

This python module is a translation and implementation of the Astro::MoonPhase Perl module via the Ruby implementation.

This package did a lot of the heavy lifting and all I needed to do was to write a simple function that accessed the module inside that package and return when the next full moon would be.

Though I wanted to add a little customisation by including the message "The next full moon will be: ". Since the package returns numbers and the message is a string; I had to concat both into a string so the message and moon date would be returned.

```python
class Option:
    NEW_READING = 1
    HISTORY = 2
    CARD_OF_THE_DAY = 3
    STAR_SIGN = 4
    STAR_CHART = 5
    FULL_MOON = 6
    QUIT = 7
```

```python
def print_options(self):
    self.print_divider(
        TerminalApp.DIVIDER_LEN, "OPTIONS", '*', 'magenta')
    print(colored("1. New Reading", 'yellow'))
    print(colored("2. History", 'green'))
    print(colored("3. Card of the Day", 'white'))
    print(colored("4. Star Sign Information", 'blue'))
    print(colored("5. Print Star Chart", 'magenta'))
    print(colored("6. Next Full Moon", 'white'))
    print(colored("7. Quit", 'red'))
    self.print_divider(
        TerminalApp.DIVIDER_LEN, divider='*', color='magenta')
```

```python
152     # Get the next Full Moon
153     def get_next_full_moon(self):
154         n = NextFullMoon()
155         return n.next_full_moon()
156
157     # Show the Next Full Moon
158     def print_next_full_moon(self):
159         moon = self.get_next_full_moon()
160         print("The next full moon will be: " + str(moon))
```

OPTION 7: QUIT

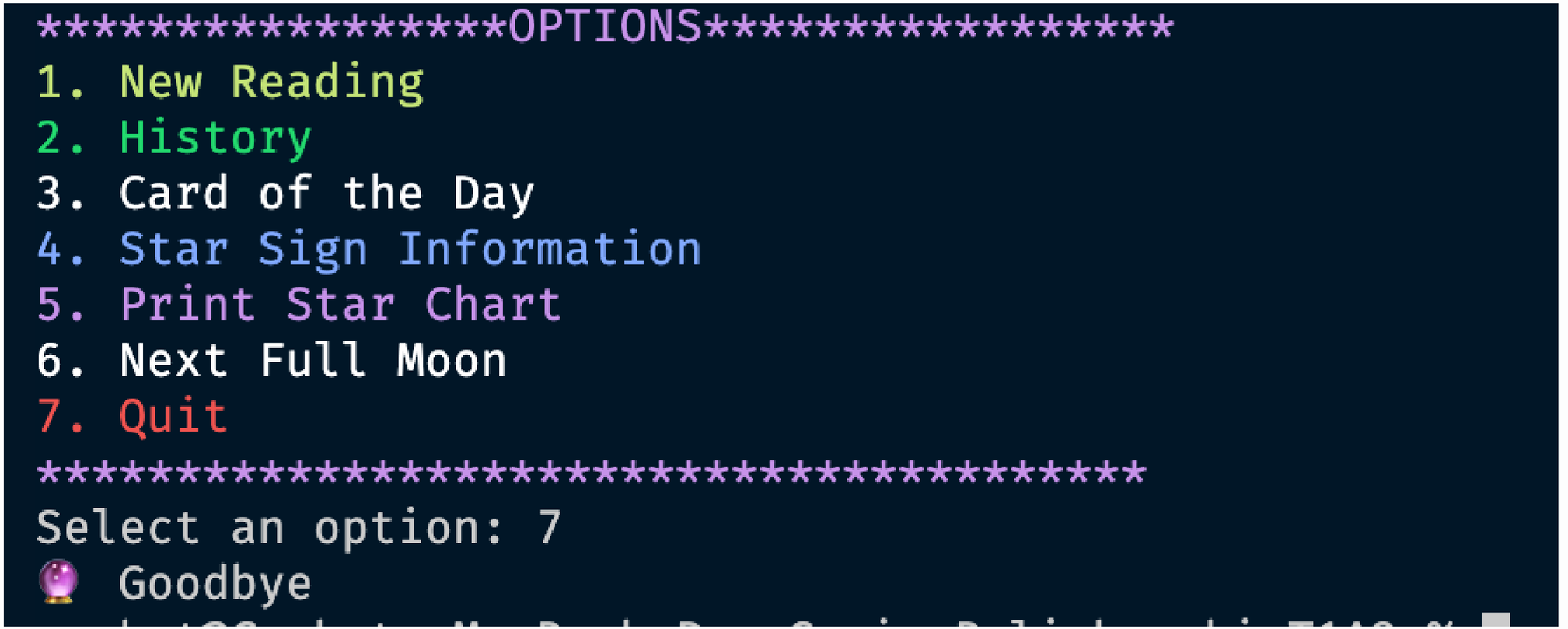
I'm a visual person so I wanted to add some kind of image to the terminal app.

I found the emoji python package and implemented the crystal ball emoji with a message "Goodbye" as it's a nice touch.

```
class Option:
    NEW_READING = 1
    HISTORY = 2
    CARD_OF_THE_DAY = 3
    STAR_SIGN = 4
    STAR_CHART = 5
    FULL_MOON = 6
    QUIT = 7
```

```
def print_options(self):
    self.print_divider(
        TerminalApp.DIVIDER_LEN, "OPTIONS", '*', 'magenta')
    print(colored("1. New Reading", 'yellow'))
    print(colored("2. History", 'green'))
    print(colored("3. Card of the Day", 'white'))
    print(colored("4. Star Sign Information", 'blue'))
    print(colored("5. Print Star Chart", 'magenta'))
    print(colored("6. Next Full Moon", 'white'))
    print(colored("7. Quit", 'red'))
    self.print_divider(
        TerminalApp.DIVIDER_LEN, divider='*', color='magenta')
```

```
def quit(self):
    farewell = emoji.emojize(':crystal_ball: Goodbye')
    print(farewell)
    exit(0)
```


**********OPTIONS**********
1. New Reading
2. History
3. Card of the Day
4. Star Sign Information
5. Print Star Chart
6. Next Full Moon
7. Quit
****
Select an option: 7
Goodbye

CHALLENGES& LEARNINGS

- Bash scripting was a lot easier than I thought and a fun way to automatically install packages and dependencies.
- Each programming language has it's limitations
- DRY principles vs. KISS
- Packages can do a whole range of things - be sure to research as there's no point in reinventing the wheel.
- Planning out how the app will function gave a lot of clarity
- I was surprised on how much over-thinking that goes into wording git commit messages. Need to be short, to the point, yet clear for the next person.

A link to this presentation video can be found here: https://vimeo.com/782220290/ff6f2a5d3d