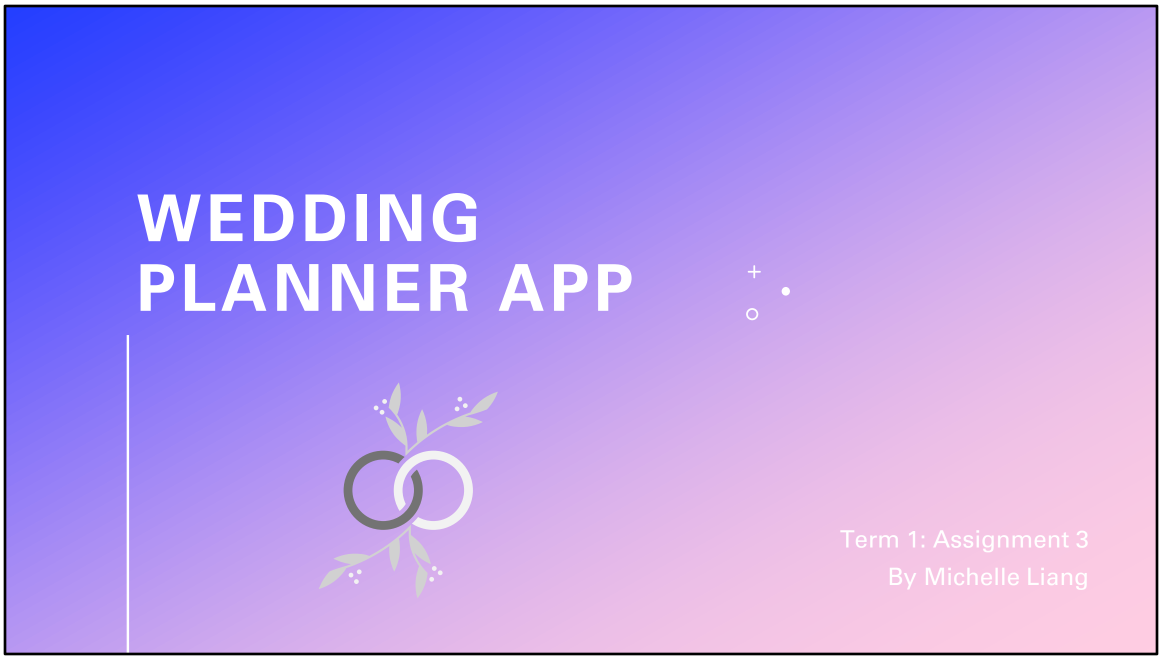
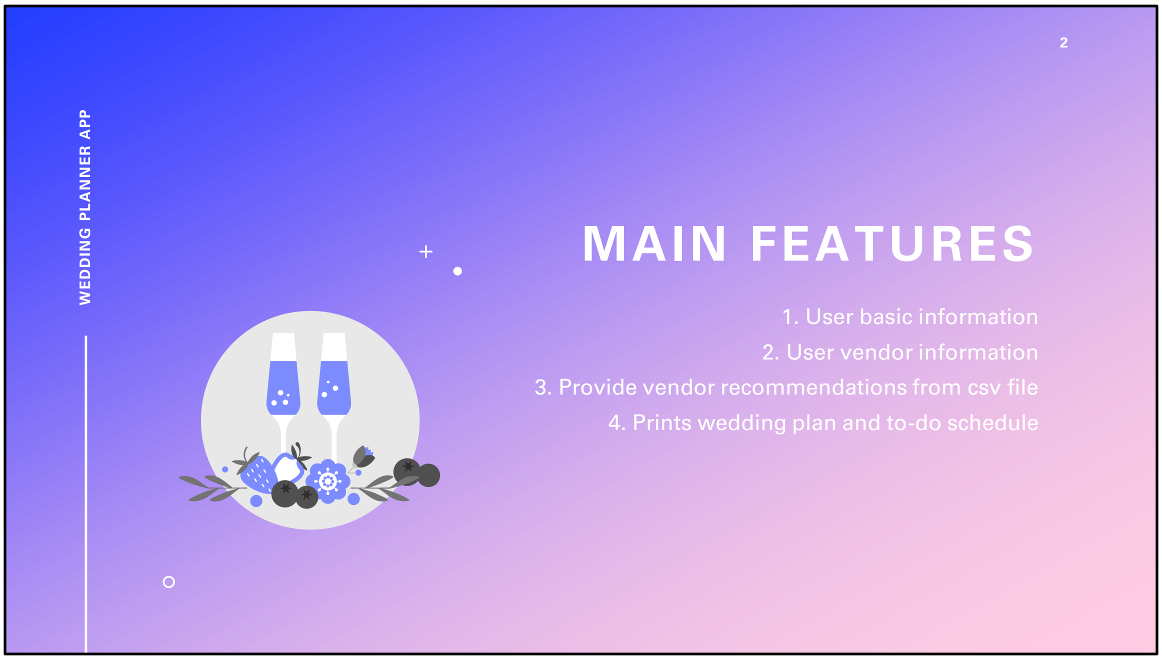


WEDDING PLANNER APP

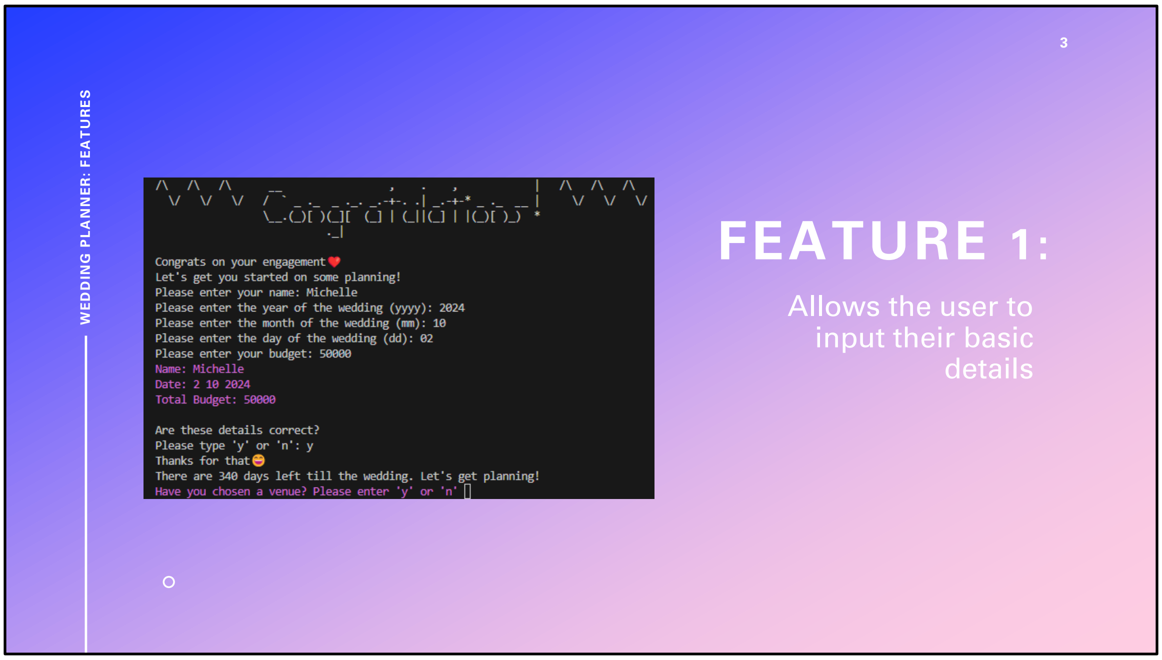
Term 1: Assignment 3
By Michelle Liang



WEDDING PLANNER APP

MAIN FEATURES

1. User basic information
2. User vendor information
3. Provide vendor recommendations from csv file
4. Prints wedding plan and to-do schedule



```
/\  /\  /\    __                ,  .  ,        |  /\  /\  /\
 \/  \/  \/  / ` _ ._  _ ._. _.-+-. .| _.-+-* _ ._ __|   \/  \/  \/
             \__.(_)[ )(_][  (_] | (_||(_] | |(_)[ )_)  *
                   ._|

Congrats on your engagement❤️
Let's get you started on some planning!
Please enter your name: Michelle
Please enter the year of the wedding (yyyy): 2024
Please enter the month of the wedding (mm): 10
Please enter the day of the wedding (dd): 02
Please enter your budget: 50000
Name: Michelle
Date: 2 10 2024
Total Budget: 50000

Are these details correct?
Please type 'y' or 'n': y
Thanks for that😍
There are 340 days left till the wedding. Let's get planning!
Have you chosen a venue? Please enter 'y' or 'n'
```

FEATURE 1:

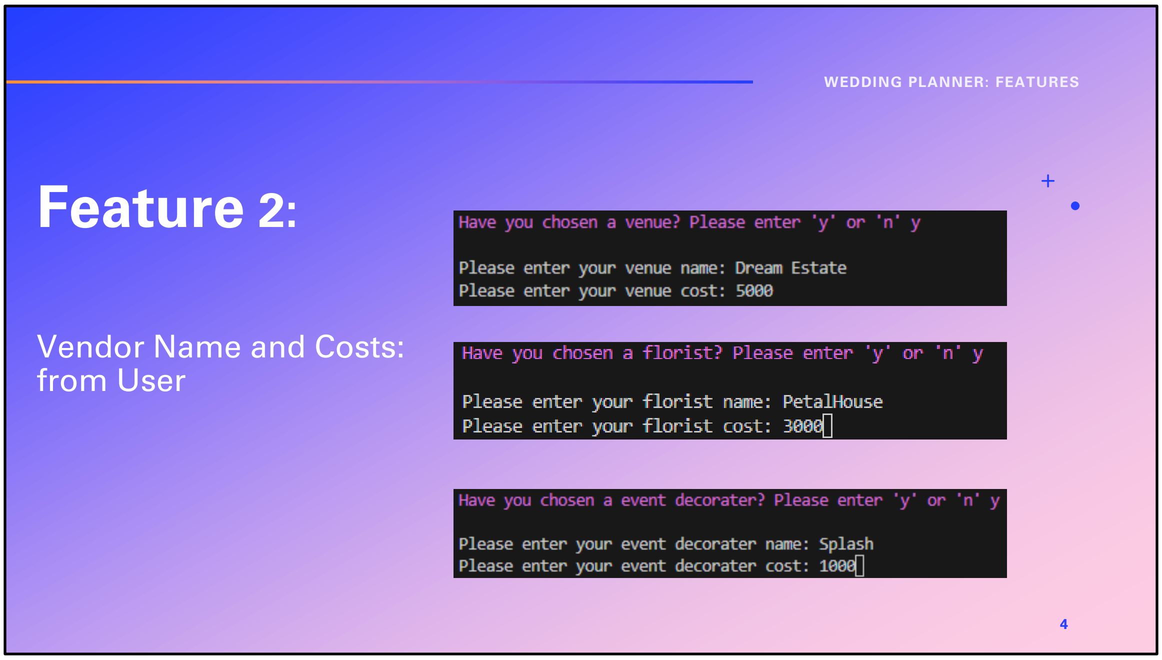
Allows the user to input their basic details

The user will be prompted to enter their:

1. Name
2. Wedding Date
3. Budget

The app will print out the inputs and ask the user to confirm their details

Once confirmed, the app will print out how many days left till the wedding and prompt the user to the next planning feature



WEDDING PLANNER: FEATURES

Feature 2:

Vendor Name and Costs: from User

```
Have you chosen a venue? Please enter 'y' or 'n' y

Please enter your venue name: Dream Estate
Please enter your venue cost: 5000
```

```
Have you chosen a florist? Please enter 'y' or 'n' y

Please enter your florist name: PetalHouse
Please enter your florist cost: 3000
```

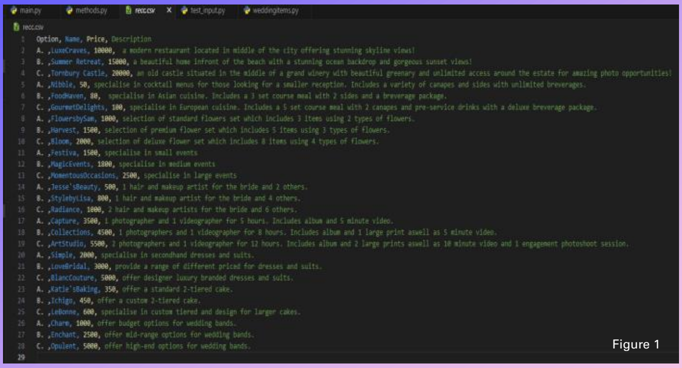
```
Have you chosen a event decorater? Please enter 'y' or 'n' y

Please enter your event decorater name: Splash
Please enter your event decorater cost: 1000
```

4

The second feature goes through several vendors for the user and asks them if they have a supplier in mind already.
If the user answers yes, they will be prompted to enter the name and cost of the supplier.
These vendors include: Venue, Catering, Florist, Décor, Beauty, Photography, Dress, Cake and Rings.

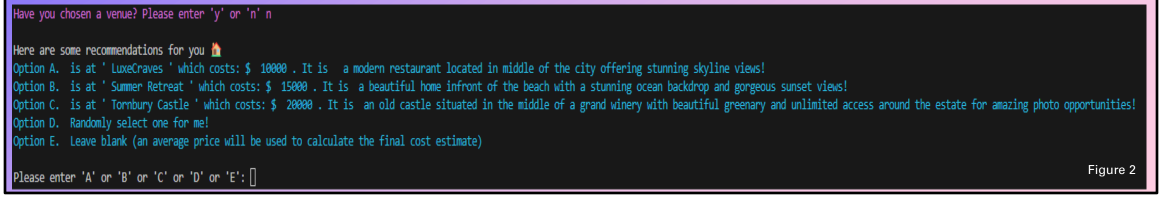
Feature 3:

Vendor Name and Costs: recommendations



```
Option, Name, Price, Description
A. ,LuxeCraves, 10000,  a modern restaurant located in middle of the city offering stunning skyline views!
B. ,Summer Retreat, 15000, a beautiful home infront of the beach with a stunning ocean backdrop and gorgeous sunset views!
C. ,Tornbury Castle, 20000, an old castle situated in the middle of a grand winery with beautiful greenary and unlimited access around the estate for amazing photo opportunities!
A. ,Nibble, 50, specialise in cocktail menus for those looking for a smaller reception. Includes a variety of canapes and sides with unlimited breverages.
B. ,FoodHaven, 80,  specialise in Asian cuisine. Includes a 3 set course meal with 2 sides and a breverage package.
C. ,GourmetDelights, 100, specialise in European cuisine. Includes a 5 set course meal with 2 canapes and pre-service drinks with a deluxe breverage package.
A. ,FlowersbySam, 1000, selection of standard flowers set which includes 3 items using 2 types of flowers.
B. ,Harvest, 1500, selection of premium flower set which includes 5 items using 3 types of flowers.
C. ,Bloom, 2000, selection of deluxe flower set which includes 8 items using 4 types of flowers.
A. ,Festiva, 1500, specialise in small events
B. ,MagicEvents, 1800, specialise in medium events
C. ,MomentousOccasions, 2500, specialise in large events
A. ,Jesse'sBeauty, 500, 1 hair and makeup artist for the bride and 2 others.
B. ,StylebyLisa, 800, 1 hair and makeup artist for the bride and 4 others.
C. ,Radiance, 1000, 2 hair and makeup artists for the bride and 6 others.
A. ,Capture, 3500, 1 photographer and 1 videographer for 5 hours. Includes album and 5 minute video.
B. ,Collections, 4500, 1 photographers and 1 videographer for 8 hours. Includes album and 1 large print aswell as 5 minute video.
C. ,ArtStudio, 5500, 2 photographers and 1 videographer for 12 hours. Includes album and 2 large prints aswell as 10 minute video and 1 engagement photoshoot session.
A. ,Simple, 2000, specialise in secondhand dresses and suits.
B. ,LoveBridal, 3000, provide a range of different priced for dresses and suits.
C. ,BlancCouture, 5000, offer designer luxury branded dresses and suits.
A. ,Katie'sBaking, 350, offer a standard 2-tiered cake.
B. ,Ichigo, 450, offer a custom 2-tiered cake.
C. ,LeBonne, 600, specialise in custom tiered and design for larger cakes.
A. ,Charm, 1000, offer budget options for wedding bands.
B. ,Enchant, 2500, offer mid-range options for wedding bands.
C. ,Opulent, 5000, offer high-end options for wedding bands.
```

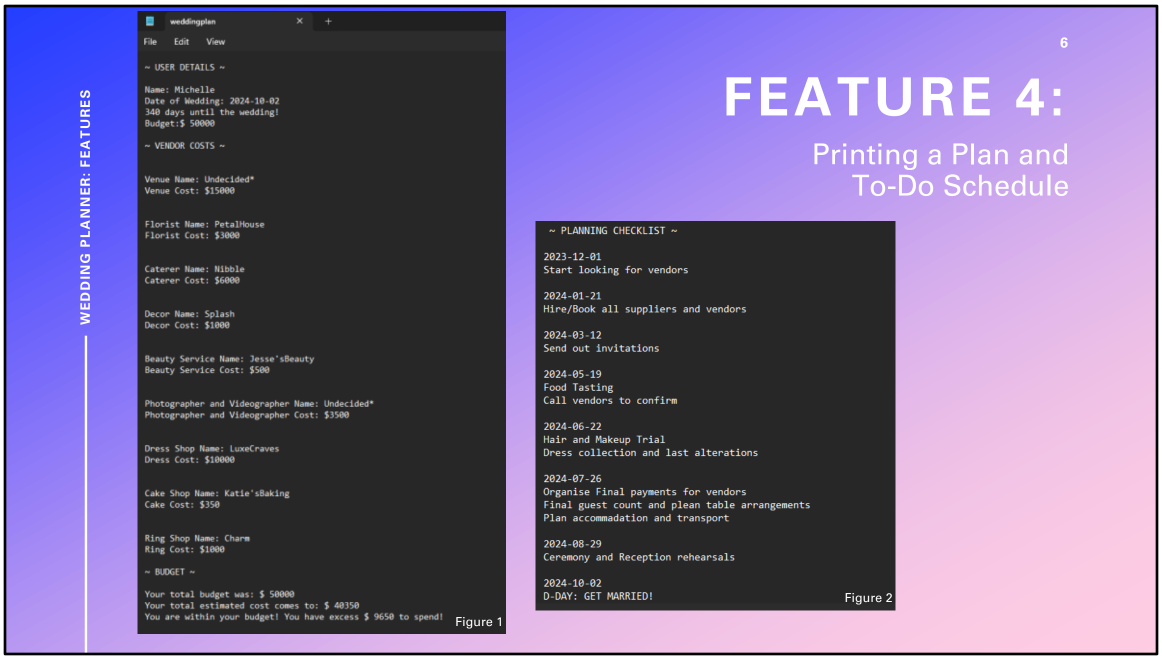
Figure 1



```
Have you chosen a venue? Please enter 'y' or 'n' n

Here are some recommendations for you 🏠
Option A.  is at ' LuxeCraves ' which costs: $  10000 . It is   a modern restaurant located in middle of the city offering stunning skyline views!
Option B.  is at ' Summer Retreat ' which costs: $  15000 . It is  a beautiful home infront of the beach with a stunning ocean backdrop and gorgeous sunset views!
Option C.  is at ' Tornbury Castle ' which costs: $  20000 . It is  an old castle situated in the middle of a grand winery with beautiful greenary and unlimited access around the estate for amazing photo opportunities!
Option D.  Randomly select one for me!
Option E.  Leave blank (an average price will be used to calculate the final cost estimate)

Please enter 'A' or 'B' or 'C' or 'D' or 'E':
```

Figure 2

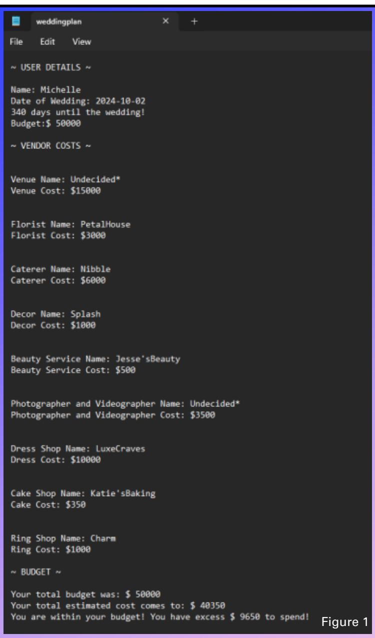
The third feature goes extracts recommendations from a csv file (figure 1).
If the user doesn't have an input for the venues, they will be offered 5 options. These include 3 recommendations (Options A-C), an option to have the app pick a random supplier (Option D) or leave the supplier blank (Option E) which results in an average costs inputted for final calculations (figure 2).



WEDDING PLANNER: FEATURES

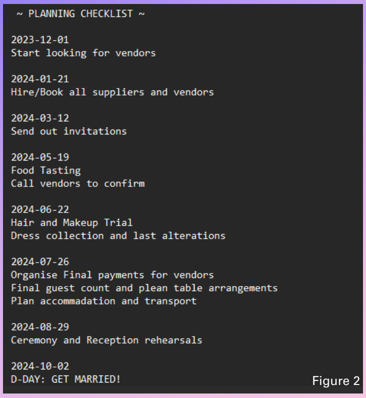
FEATURE 4:

Printing a Plan and To-Do Schedule



```
~ USER DETAILS ~

Name: Michelle
Date of Wedding: 2024-10-02
340 days until the wedding!
Budget:$ 50000

~ VENDOR COSTS ~


Venue Name: Undecided*
Venue Cost: $15000


Florist Name: PetalHouse
Florist Cost: $3000


Caterer Name: Nibble
Caterer Cost: $6000


Decor Name: Splash
Decor Cost: $1000


Beauty Service Name: Jesse'sBeauty
Beauty Service Cost: $500


Photographer and Videographer Name: Undecided*
Photographer and Videographer Cost: $3500


Dress Shop Name: LuxeCraves
Dress Cost: $10000


Cake Shop Name: Katie'sBaking
Cake Cost: $350


Ring Shop Name: Charm
Ring Cost: $1000

~ BUDGET ~

Your total budget was: $ 50000
Your total estimated cost comes to: $ 40350
You are within your budget! You have excess $ 9650 to spend!
```

Figure 1

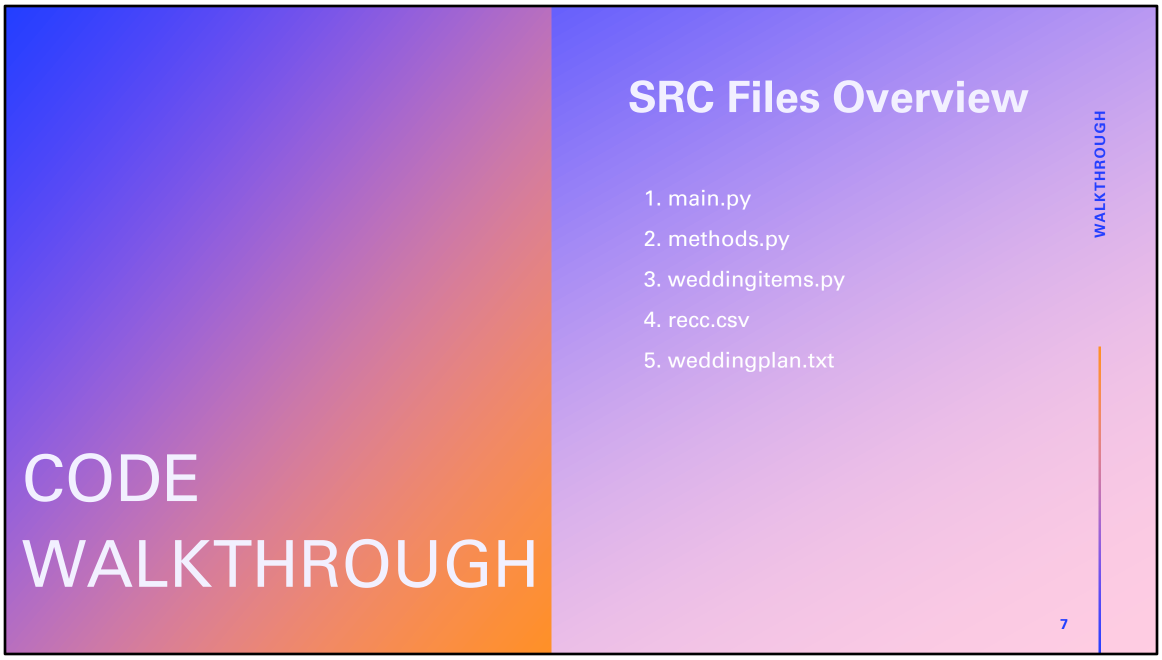


```
 ~ PLANNING CHECKLIST ~

2023-12-01
Start looking for vendors

2024-01-21
Hire/Book all suppliers and vendors

2024-03-12
Send out invitations

2024-05-19
Food Tasting
Call vendors to confirm

2024-06-22
Hair and Makeup Trial
Dress collection and last alterations

2024-07-26
Organise Final payments for vendors
Final guest count and plean table arrangements
Plan accommadation and transport

2024-08-29
Ceremony and Reception rehearsals

2024-10-02
D-DAY: GET MARRIED!
```

Figure 2

The last feature is providing a text file of their wedding plan for the user to save. At the end, the app will open a text file with the user's basic information, vendor selections, a final cost estimate and a message to let them know if they are over or under budget and by how much (figure 1).

It will also print out a scheduled to-do list with dates to help the user tick off important tasks before their big day (figure 2).



CODE WALKTHROUGH

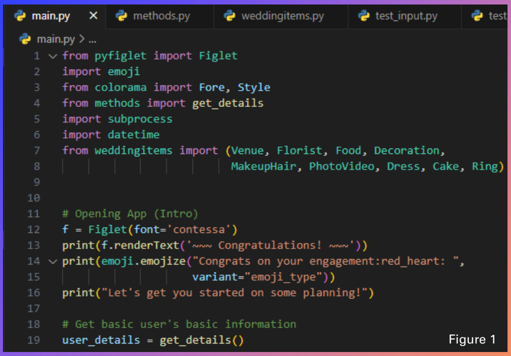
SRC Files Overview

1. main.py
2. methods.py
3. weddingitems.py
4. recc.csv
5. weddingplan.txt

WALKTHROUGH

7

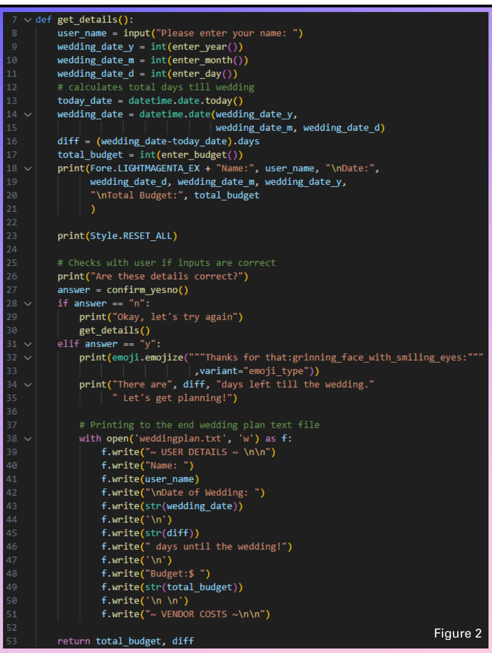
Code Walkthrough:
First an introduction to the different files:
1. main.py: is the app’s main file
2. methods.py: contains functions used in the main file
3. weddingitems.py: contains all the vendor classes used in the app
4. recc.csv: contains the data used to provide recommendations to the user
5. weddingplan.txt: is the text file outputted at the end with the users wedding plan ready to save



```python
1  from pyfiglet import Figlet
2  import emoji
3  from colorama import Fore, Style
4  from methods import get_details
5  import subprocess
6  import datetime
7  from weddingitems import (Venue, Florist, Food, Decoration,
8                            MakeupHair, PhotoVideo, Dress, Cake, Ring)
9
10
11 # Opening App (Intro)
12 f = Figlet(font='contessa')
13 print(f.renderText('~~~ Congratulations! ~~~'))
14 print(emoji.emojize("Congrats on your engagement:red_heart: ",
15                     variant="emoji_type"))
16 print("Let's get you started on some planning!")
17
18 # Get basic user's basic information
19 user_details = get_details()
```

Figure 1

Intro and User Info



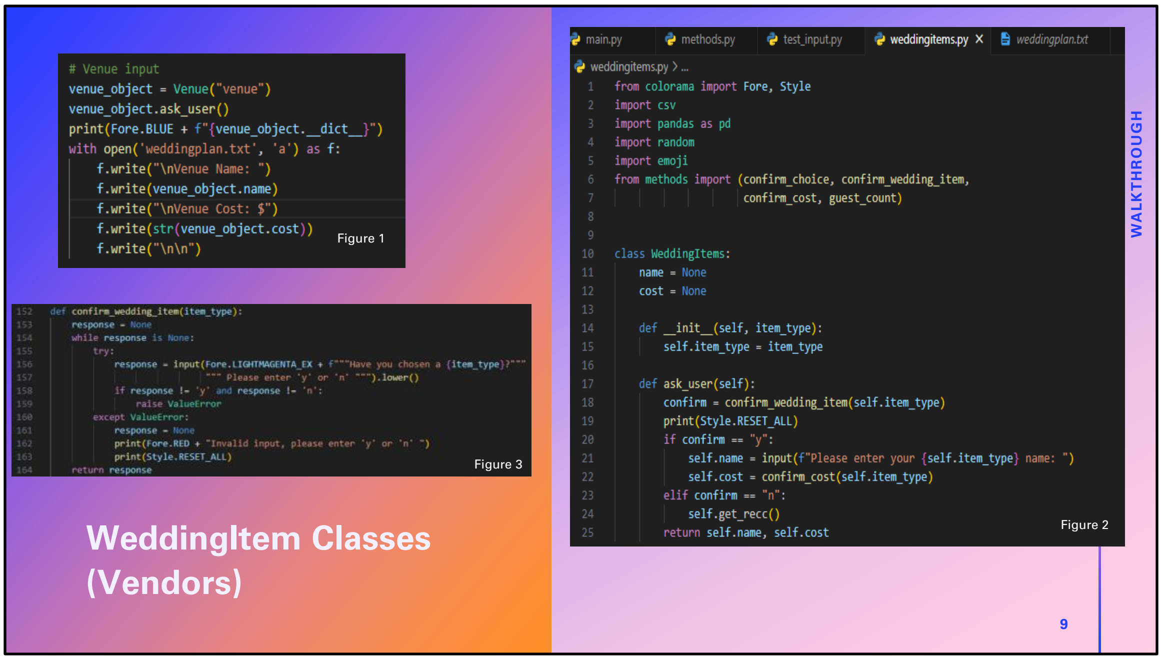
```python
7  def get_details():
8      user_name = input("Please enter your name: ")
9      wedding_date_y = int(enter_year())
10     wedding_date_m = int(enter_month())
11     wedding_date_d = int(enter_day())
12     # calculates total days till wedding
13     today_date = datetime.date.today()
14     wedding_date = datetime.date(wedding_date_y,
15                                  wedding_date_m, wedding_date_d)
16     diff = (wedding_date-today_date).days
17     total_budget = int(enter_budget())
18     print(Fore.LIGHTMAGENTA_EX + "Name:", user_name, "\nDate:",
19           wedding_date_d, wedding_date_m, wedding_date_y,
20           "\nTotal Budget:", total_budget
21           )
22
23     print(Style.RESET_ALL)
24
25     # Checks with user if inputs are correct
26     print("Are these details correct?")
27     answer = confirm_yesno()
28     if answer == "n":
29         print("Okay, let's try again")
30         get_details()
31     elif answer == "y":
32         print(emoji.emojize("""Thanks for that:grinning_face_with_smiling_eyes:"""
33                             ,variant="emoji_type"))
34         print("There are", diff, "days left till the wedding."
35               " Let's get planning!")
36
37         # Printing to the end wedding plan text file
38         with open('weddingplan.txt', 'w') as f:
39             f.write("~ USER DETAILS ~ \n\n")
40             f.write("Name: ")
41             f.write(user_name)
42             f.write("\nDate of Wedding: ")
43             f.write(str(wedding_date))
44             f.write('\n')
45             f.write(str(diff))
46             f.write(" days until the wedding!")
47             f.write('\n')
48             f.write("Budget:$ ")
49             f.write(str(total_budget))
50             f.write('\n \n')
51             f.write("~ VENDOR COSTS ~\n\n")
52
53     return total_budget, diff
```

Figure 2

WALKTHROUGH

The application begins with an opening to welcome the users (lines 12 to 16, figure 1)
Next it asks for some basic information from the user through the get_details() function
The get_details() function is shown in figure 2. It involves:

1. Asking for user's name and date of wedding (line 7 -11) and uses the latter and the time.date module to calculate how many days are left till the wedding (line 12-16)., storing this in the variable "diff".
2. It then asks the user for their budget (line 17) and prints their inputted details before asking the user to confirm their inputs. If they are incorrect, the user can type 'n' and restart the process or 'y' to proceed (line 18 – 34).
3. The function then begins printing the user's basic information onto a text file "weddingplan.txt" (line 38 – 51).
4. Finally in line 53, the function will return two values : 'total_budget'(user inputted budget) and 'diff'(days till the wedding) which are stored in a tuple in user_details (line 19, figure 1) for later use.



```
# Venue input
venue_object = Venue("venue")
venue_object.ask_user()
print(Fore.BLUE + f"{venue_object.__dict__}")
with open('weddingplan.txt', 'a') as f:
    f.write("\nVenue Name: ")
    f.write(venue_object.name)
    f.write("\nVenue Cost: $")
    f.write(str(venue_object.cost))
    f.write("\n\n")
```

Figure 1

```
def confirm_wedding_item(item_type):
    response = None
    while response is None:
        try:
            response = input(Fore.LIGHTMAGENTA_EX + f"""Have you chosen a {item_type}?"""
                             """ Please enter 'y' or 'n' """).lower()
            if response != 'y' and response != 'n':
                raise ValueError
        except ValueError:
            response = None
            print(Fore.RED + "Invalid input, please enter 'y' or 'n' ")
            print(Style.RESET_ALL)
    return response
```

Figure 3

WeddingItem Classes (Vendors)

```
1   from colorama import Fore, Style
2   import csv
3   import pandas as pd
4   import random
5   import emoji
6   from methods import (confirm_choice, confirm_wedding_item,
7                        confirm_cost, guest_count)
8
9
10  class WeddingItems:
11      name = None
12      cost = None
13
14      def __init__(self, item_type):
15          self.item_type = item_type
16
17      def ask_user(self):
18          confirm = confirm_wedding_item(self.item_type)
19          print(Style.RESET_ALL)
20          if confirm == "y":
21              self.name = input(f"Please enter your {self.item_type} name: ")
22              self.cost = confirm_cost(self.item_type)
23          elif confirm == "n":
24              self.get_recc()
25          return self.name, self.cost
```

Figure 2

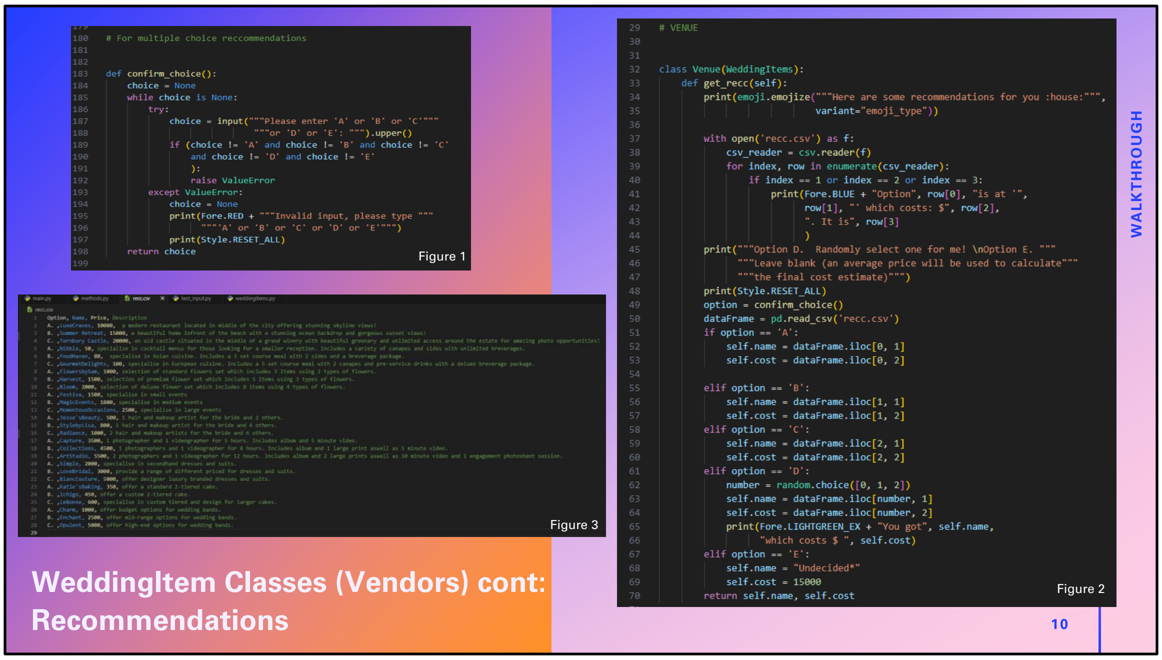
WALKTHROUGH

9

After the user details are recorded, the app will then begin going through different types of vendors and asking the user if they have or have not chosen vendors already. The app will call the different vendors such as 'venue' in figure 1.

Each type of vendor is a subclass of the WeddingItem Class (line 10, figure 2). The class has 3 attributes: a name, a cost and an item type (lines 11-15, figure 2). They also have an ask_user() method to ask the user whether they have a chosen vendor in mind already or not (line 18, figure 2). The method can be seen in more detail in figure 3. If they have a vendor and answer y, they will be prompted to input the name and cost which are stored as the name and cost of the instance (lines 21 and 22, figure 2).

If the user answers no to the ask_user() method, the app runs to the get_recc() function (line 24, figure 2) which leads to the third feature of the app on the next slide:



```
180  # For multiple choice reccommendations
181
182
183  def confirm_choice():
184      choice = None
185      while choice is None:
186          try:
187              choice = input("""Please enter 'A' or 'B' or 'C'"""
188                              """or 'D' or 'E': """).upper()
189              if (choice != 'A' and choice != 'B' and choice != 'C'
190                  and choice != 'D' and choice != 'E'
191                  ):
192                  raise ValueError
193          except ValueError:
194              choice = None
195              print(Fore.RED + """Invalid input, please type """
196                      """'A' or 'B' or 'C' or 'D' or 'E'""")
197              print(Style.RESET_ALL)
198      return choice
199
```

Figure 1

```
main.py   methods.py   recc.csv ×   test_input.py   weddingitems.py
recc.csv
Option, Name, Price, Description
A. ,LuxeCraves, 10000,  a modern restaurant located in middle of the city offering stunning skyline views!
B. ,Summer Retreat, 15000, a beautiful home infront of the beach with a stunning ocean backdrop and gorgeous sunset views!
C. ,Tornbury Castle, 20000, an old castle situated in the middle of a grand winery with beautiful greenary and unlimited access around the estate for amazing photo opportunities!
A. ,Nibble, 50, specialise in cocktail menus for those looking for a smaller reception. Includes a variety of canapes and sides with unlimited breverages.
B. ,FoodHaven, 80,  specialise in Asian cuisine. Includes a 3 set course meal with 2 sides and a breverage package.
C. ,GourmetDelights, 100, specialise in European cuisine. Includes a 5 set course meal with 2 canapes and pre-service drinks with a deluxe breverage package.
A. ,FlowersbySam, 1000, selection of standard flowers set which includes 3 items using 2 types of flowers.
B. ,Harvest, 1500, selection of premium flower set which includes 5 items using 3 types of flowers.
C. ,Bloom, 2000, selection of deluxe flower set which includes 8 items using 4 types of flowers.
A. ,Festiva, 1500, specialise in small events
B. ,MagicEvents, 1800, specialise in medium events
C. ,MomentousOccasions, 2500, specialise in large events
A. ,Jesse'sBeauty, 500, 1 hair and makeup artist for the bride and 2 others.
B. ,StylebyLisa, 800, 1 hair and makeup artist for the bride and 4 others.
C. ,Radiance, 1000, 2 hair and makeup artists for the bride and 6 others.
A. ,Capture, 3500, 1 photographer and 1 videographer for 5 hours. Includes album and 5 minute video.
B. ,Collections, 4500, 1 photographers and 1 videographer for 8 hours. Includes album and 1 large print aswell as 5 minute video.
C. ,ArtStudio, 5500, 2 photographers and 1 videographer for 12 hours. Includes album and 2 large prints aswell as 10 minute video and 1 engagement photoshoot session.
A. ,Simple, 2000, specialise in secondhand dresses and suits.
B. ,LoveBridal, 3000, provide a range of different priced for dresses and suits.
C. ,BlancCouture, 5000, offer designer luxury branded dresses and suits.
A. ,Katie'sBaking, 350, offer a standard 2-tiered cake.
B. ,Ichigo, 450, offer a custom 2-tiered cake.
C. ,LeBonne, 600, specialise in custom tiered and design for larger cakes.
A. ,Charm, 1000, offer budget options for wedding bands.
B. ,Enchant, 2500, offer mid-range options for wedding bands.
C. ,Opulent, 5000, offer high-end options for wedding bands.
```

Figure 3

WeddingItem Classes (Vendors) cont: Recommendations

```
29  # VENUE
30
31
32  class Venue(WeddingItems):
33      def get_recc(self):
34          print(emoji.emojize("""Here are some recommendations for you :house:""",
35                              variant="emoji_type"))
36
37          with open('recc.csv') as f:
38              csv_reader = csv.reader(f)
39              for index, row in enumerate(csv_reader):
40                  if index == 1 or index == 2 or index == 3:
41                      print(Fore.BLUE + "Option", row[0], "is at '",
42                            row[1], "' which costs: $", row[2],
43                            ". It is", row[3]
44                            )
45          print("""Option D.  Randomly select one for me! \nOption E. """
46                """Leave blank (an average price will be used to calculate"""
47                """the final cost estimate)""")
48          print(Style.RESET_ALL)
49          option = confirm_choice()
50          dataFrame = pd.read_csv('recc.csv')
51          if option == 'A':
52              self.name = dataFrame.iloc[0, 1]
53              self.cost = dataFrame.iloc[0, 2]
54
55          elif option == 'B':
56              self.name = dataFrame.iloc[1, 1]
57              self.cost = dataFrame.iloc[1, 2]
58          elif option == 'C':
59              self.name = dataFrame.iloc[2, 1]
60              self.cost = dataFrame.iloc[2, 2]
61          elif option == 'D':
62              number = random.choice([0, 1, 2])
63              self.name = dataFrame.iloc[number, 1]
64              self.cost = dataFrame.iloc[number, 2]
65              print(Fore.LIGHTGREEN_EX + "You got", self.name,
66                    "which costs $ ", self.cost)
67          elif option == 'E':
68              self.name = "Undecided*"
69              self.cost = 15000
70          return self.name, self.cost
```

Figure 2

WALKTHROUGH

10

As mentioned earlier, if the user answers no, the get_recc() method is called. This method is defined the different subclasses of WeddingItems. Using the “Venue” example again, you can see in figure 2 that once the get_recc method is called, the app will open a csv file (‘recc.cvs’ in figure 3) where all the recommended vendors are stored. Once opened, the method will iterate through the first 3 recommendations (lines 39-44, figure 2) and print out the 3 options as well as a fourth option to randomly select an option and a fifth option to leave the vendor empty (line 45-47, figure 2).

The user is then asked to pick an option using the confirm_choice() function (figure 1). This function handles some error handling which will be further discussed in later slides, but we can see the input is returned as ‘choice’ (line 198, figure 1) and using the returned ‘choice’, the app will open the recc.csv file again and go through some conditional statements that depend on the user’s ‘choice’. Using dataframes, it will locate the corresponding venue name and costs, and these will be stored in the name and cost attributes (lines 51-60, figure 2).

Option D uses the random module to randomly select a row in the csv file out of the 3 venues (lines 61-66) and option E will set name to “undecided” and cost to a set average cost for venue suppliers (in this case 15000) (line 67-69, figure 2).

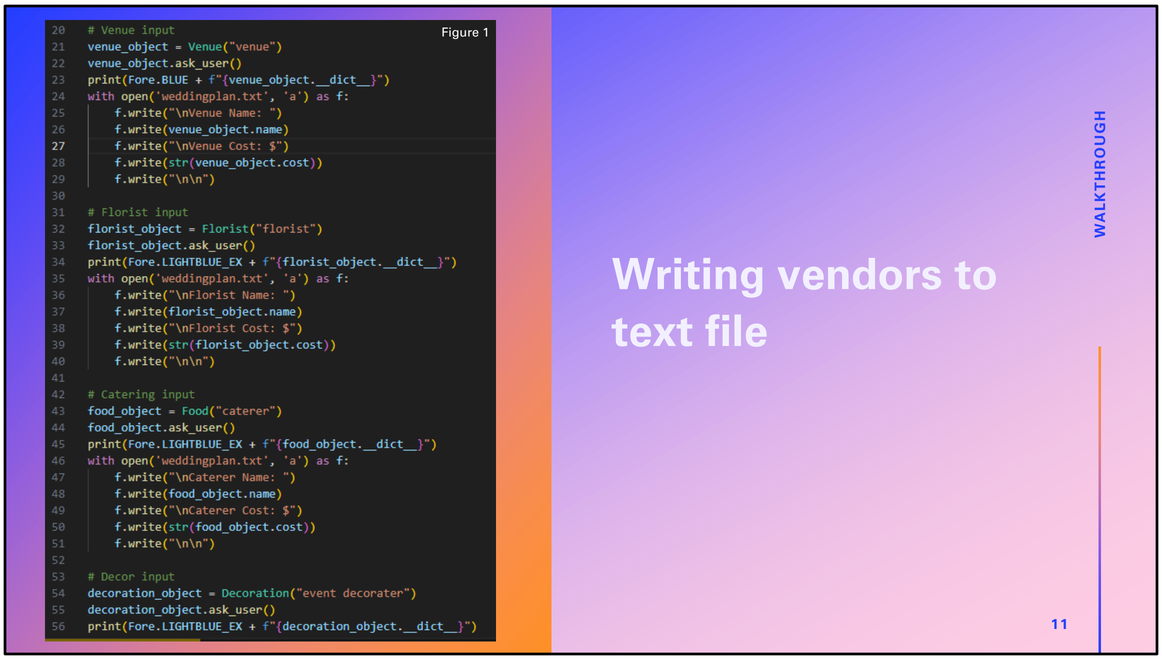


Figure 1

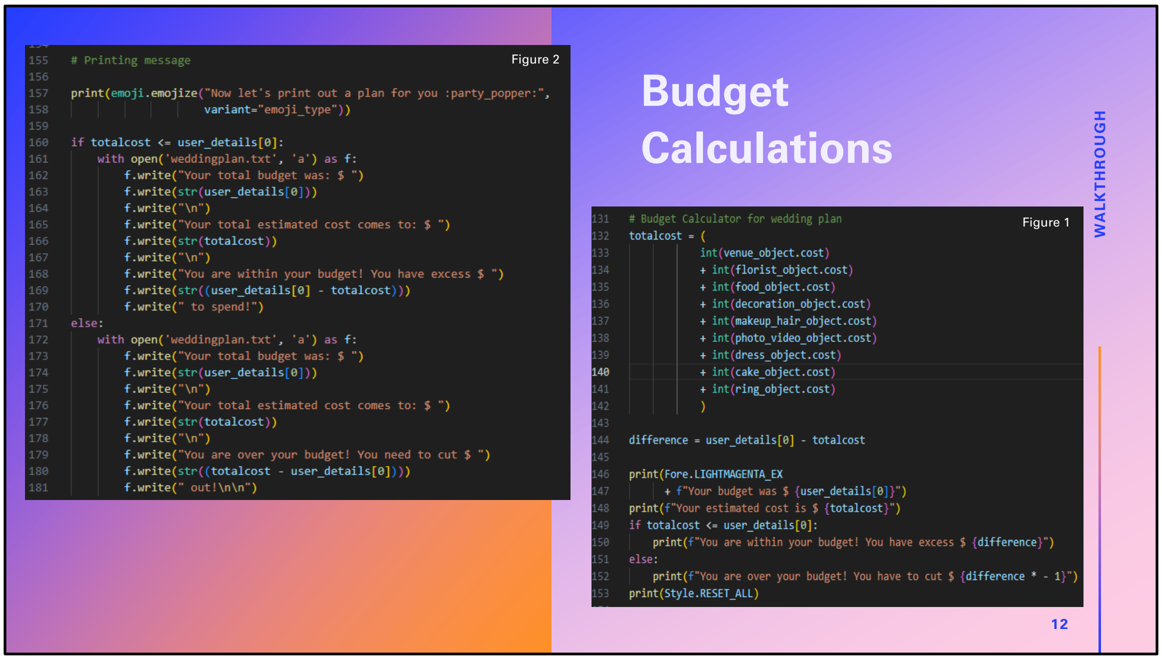
```
20  # Venue input
21  venue_object = Venue("venue")
22  venue_object.ask_user()
23  print(Fore.BLUE + f"{venue_object.__dict__}")
24  with open('weddingplan.txt', 'a') as f:
25      f.write("\nVenue Name: ")
26      f.write(venue_object.name)
27      f.write("\nVenue Cost: $")
28      f.write(str(venue_object.cost))
29      f.write("\n\n")
30
31  # Florist input
32  florist_object = Florist("florist")
33  florist_object.ask_user()
34  print(Fore.LIGHTBLUE_EX + f"{florist_object.__dict__}")
35  with open('weddingplan.txt', 'a') as f:
36      f.write("\nFlorist Name: ")
37      f.write(florist_object.name)
38      f.write("\nFlorist Cost: $")
39      f.write(str(florist_object.cost))
40      f.write("\n\n")
41
42  # Catering input
43  food_object = Food("caterer")
44  food_object.ask_user()
45  print(Fore.LIGHTBLUE_EX + f"{food_object.__dict__}")
46  with open('weddingplan.txt', 'a') as f:
47      f.write("\nCaterer Name: ")
48      f.write(food_object.name)
49      f.write("\nCaterer Cost: $")
50      f.write(str(food_object.cost))
51      f.write("\n\n")
52
53  # Decor input
54  decoration_object = Decoration("event decorater")
55  decoration_object.ask_user()
56  print(Fore.LIGHTBLUE_EX + f"{decoration_object.__dict__}")
```

WALKTHROUGH

Writing vendors to text file

11

Once the user has inputted a vendor name and cost or picked a recommendation, ask_user() is completed. The app will then print out the dictionary of the class (lines 23, 34, 45) so that the user can see the inputs/choices and finally, the app will write down the name and cost to the weddingplan.txt file. This process is repeated for all the different subclasses of WeddingItems such as the Florist, Catering, Décor, etc. (e.g., lines 24-29).



Budget Calculations

WALKTHROUGH

Figure 1

```
131 # Budget Calculator for wedding plan
132 totalcost = (
133             int(venue_object.cost)
134             + int(florist_object.cost)
135             + int(food_object.cost)
136             + int(decoration_object.cost)
137             + int(makeup_hair_object.cost)
138             + int(photo_video_object.cost)
139             + int(dress_object.cost)
140             + int(cake_object.cost)
141             + int(ring_object.cost)
142             )
143
144 difference = user_details[0] - totalcost
145
146 print(Fore.LIGHTMAGENTA_EX
147       + f"Your budget was $ {user_details[0]}")
148 print(f"Your estimated cost is $ {totalcost}")
149 if totalcost <= user_details[0]:
150     print(f"You are within your budget! You have excess $ {difference}")
151 else:
152     print(f"You are over your budget! You have to cut $ {difference * - 1}")
153 print(Style.RESET_ALL)
```

Figure 2

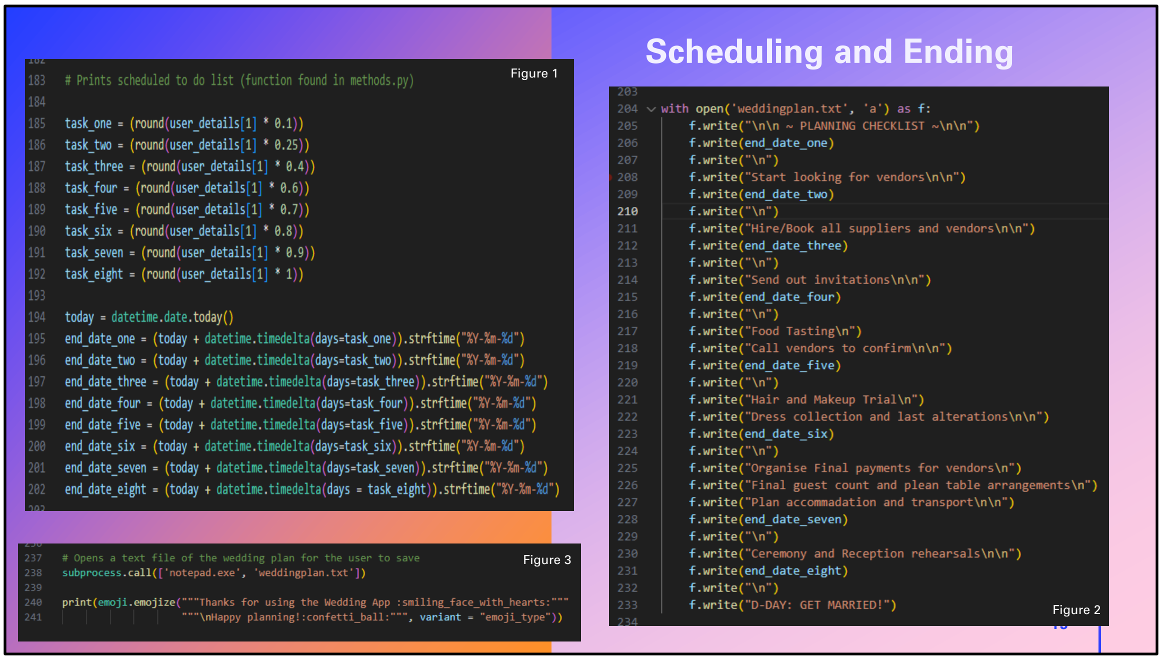
```
155 # Printing message
156
157 print(emoji.emojize("Now let's print out a plan for you :party_popper:",
158                     variant="emoji_type"))
159
160 if totalcost <= user_details[0]:
161     with open('weddingplan.txt', 'a') as f:
162         f.write("Your total budget was: $ ")
163         f.write(str(user_details[0]))
164         f.write("\n")
165         f.write("Your total estimated cost comes to: $ ")
166         f.write(str(totalcost))
167         f.write("\n")
168         f.write("You are within your budget! You have excess $ ")
169         f.write(str((user_details[0] - totalcost)))
170         f.write(" to spend!")
171 else:
172     with open('weddingplan.txt', 'a') as f:
173         f.write("Your total budget was: $ ")
174         f.write(str(user_details[0]))
175         f.write("\n")
176         f.write("Your total estimated cost comes to: $ ")
177         f.write(str(totalcost))
178         f.write("\n")
179         f.write("You are over your budget! You need to cut $ ")
180         f.write(str((totalcost - user_details[0])))
181         f.write(" out!\n\n")
```

12

Next up, we have our budget calculations in figure 1 (lines 131-141) which takes the integer form of the cost attributes of all the vendors. The difference between the costs and the user inputted budget is calculated in line 144 where we extract the first tuple element from our earlier get_details() function which returned user_details().

The total cost, inputted user budget and the difference is printed in the app for the user to see immediately (line 146-153, figure 1).

Figure 2 is printing the same details but this time to the weddingplan text file.



Scheduling and Ending

Figure 1

```
183  # Prints scheduled to do list (function found in methods.py)
184
185  task_one = (round(user_details[1] * 0.1))
186  task_two = (round(user_details[1] * 0.25))
187  task_three = (round(user_details[1] * 0.4))
188  task_four = (round(user_details[1] * 0.6))
189  task_five = (round(user_details[1] * 0.7))
190  task_six = (round(user_details[1] * 0.8))
191  task_seven = (round(user_details[1] * 0.9))
192  task_eight = (round(user_details[1] * 1))
193
194  today = datetime.date.today()
195  end_date_one = (today + datetime.timedelta(days=task_one)).strftime("%Y-%m-%d")
196  end_date_two = (today + datetime.timedelta(days=task_two)).strftime("%Y-%m-%d")
197  end_date_three = (today + datetime.timedelta(days=task_three)).strftime("%Y-%m-%d")
198  end_date_four = (today + datetime.timedelta(days=task_four)).strftime("%Y-%m-%d")
199  end_date_five = (today + datetime.timedelta(days=task_five)).strftime("%Y-%m-%d")
200  end_date_six = (today + datetime.timedelta(days=task_six)).strftime("%Y-%m-%d")
201  end_date_seven = (today + datetime.timedelta(days=task_seven)).strftime("%Y-%m-%d")
202  end_date_eight = (today + datetime.timedelta(days = task_eight)).strftime("%Y-%m-%d")
```

Figure 2

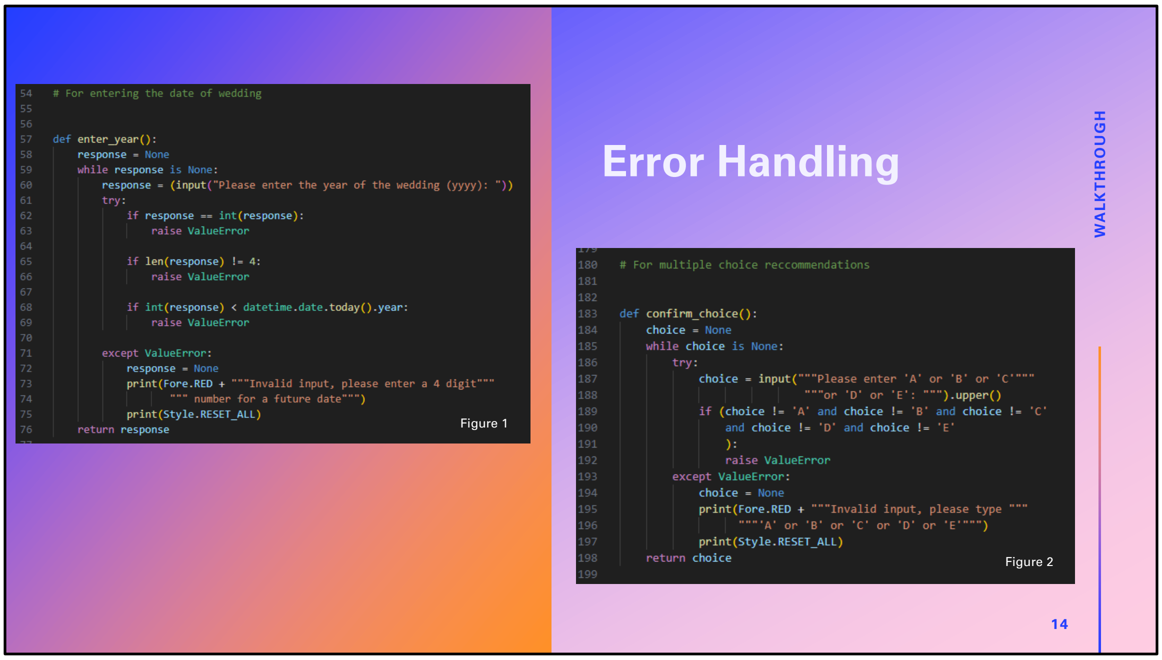
```
203
204  with open('weddingplan.txt', 'a') as f:
205      f.write("\n\n ~ PLANNING CHECKLIST ~\n\n")
206      f.write(end_date_one)
207      f.write("\n")
208      f.write("Start looking for vendors\n\n")
209      f.write(end_date_two)
210      f.write("\n")
211      f.write("Hire/Book all suppliers and vendors\n\n")
212      f.write(end_date_three)
213      f.write("\n")
214      f.write("Send out invitations\n\n")
215      f.write(end_date_four)
216      f.write("\n")
217      f.write("Food Tasting\n")
218      f.write("Call vendors to confirm\n\n")
219      f.write(end_date_five)
220      f.write("\n")
221      f.write("Hair and Makeup Trial\n")
222      f.write("Dress collection and last alterations\n\n")
223      f.write(end_date_six)
224      f.write("\n")
225      f.write("Organise Final payments for vendors\n")
226      f.write("Final guest count and plean table arrangements\n")
227      f.write("Plan accommadation and transport\n\n")
228      f.write(end_date_seven)
229      f.write("\n")
230      f.write("Ceremony and Reception rehearsals\n\n")
231      f.write(end_date_eight)
232      f.write("\n")
233      f.write("D-DAY: GET MARRIED!")
234
```

Figure 3

```
237  # Opens a text file of the wedding plan for the user to save
238  subprocess.call(['notepad.exe', 'weddingplan.txt'])
239
240  print(emoji.emojize("""Thanks for using the Wedding App :smiling_face_with_hearts:"""
241                      """\nHappy planning!:confetti_ball:""", variant = "emoji_type"))
```

Finally, we have the scheduling feature. In figure 1, the time stamps are set in lines 185-192. This is done by multiplying the number of days till the wedding (which was returned in the get_details() function and stored in the second element of 'user_details' earlier) with a value between 0 and 1. So line 185 multiplies days left with 0.1 giving us a timestamp at 90% of they days remaining till the wedding. This figure is then added to today's date in lines 194-202 which gives us an exact date in dd/mm/yyyy format. Finally in figure 2, we open the weddingplan text again and append it with important tasks for the couple to do before the wedding date.

At the very end, we open the file for the user in notepad (line 238, figure 3) to let the user save the file to their local PC. Once the user is done they can close the file and a final thank you message is printed (line 240-241)!



```
54  # For entering the date of wedding
55
56
57  def enter_year():
58      response = None
59      while response is None:
60          response = (input("Please enter the year of the wedding (yyyy): "))
61          try:
62              if response == int(response):
63                  raise ValueError
64
65              if len(response) != 4:
66                  raise ValueError
67
68              if int(response) < datetime.date.today().year:
69                  raise ValueError
70
71          except ValueError:
72              response = None
73              print(Fore.RED + """Invalid input, please enter a 4 digit"""
74                    """ number for a future date""")
75              print(Style.RESET_ALL)
76      return response
```

Figure 1

Error Handling

WALKTHROUGH

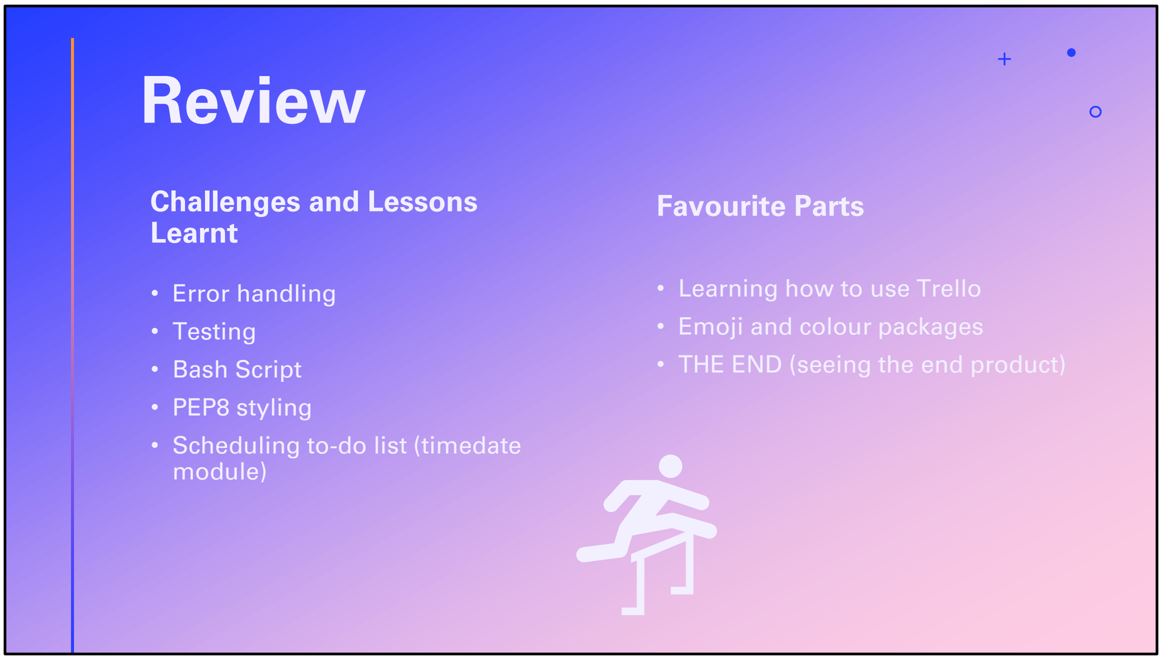
```
# For multiple choice reccommendations


def confirm_choice():
    choice = None
    while choice is None:
        try:
            choice = input("""Please enter 'A' or 'B' or 'C'"""
                                """or 'D' or 'E': """).upper()
            if (choice != 'A' and choice != 'B' and choice != 'C'
                and choice != 'D' and choice != 'E'
                ):
                raise ValueError
        except ValueError:
            choice = None
            print(Fore.RED + """Invalid input, please type """
                      """'A' or 'B' or 'C' or 'D' or 'E'""")
            print(Style.RESET_ALL)
    return choice

```

Figure 2

14

For error handling, I focused on mostly on potential input errors. Most of there error handling uses the same structure as the two examples above. In figure 1, for the function enter_year(),a while loop is set so that the user will be repeatedly be asked for an input until they enter a valid input (that is until no exceptions are raised). For this example, exceptions are raised when the input is not an integer (line 62, figure 1), when the input is not 4 characters long as the year requires 4 digits line 65, figure 1) and finally if the date entered is less than today's date to ensure the user enters a future date (line 68, figure 1).

In our second example in figure 2, we have our answer prompt (confirm_choice()) for the venue recommendation choices. Similarly, a while loop is used to ensure the user will continue to be prompted until they enter a valid option. In this case, any answer that is NOT A or B or C or D or E will raise and exception. The .upper() has been added to allow the user to enter lower case letters for more leniency.



Review

Challenges and Lessons Learnt

- Error handling
- Testing
- Bash Script
- PEP8 styling
- Scheduling to-do list (timedate module)

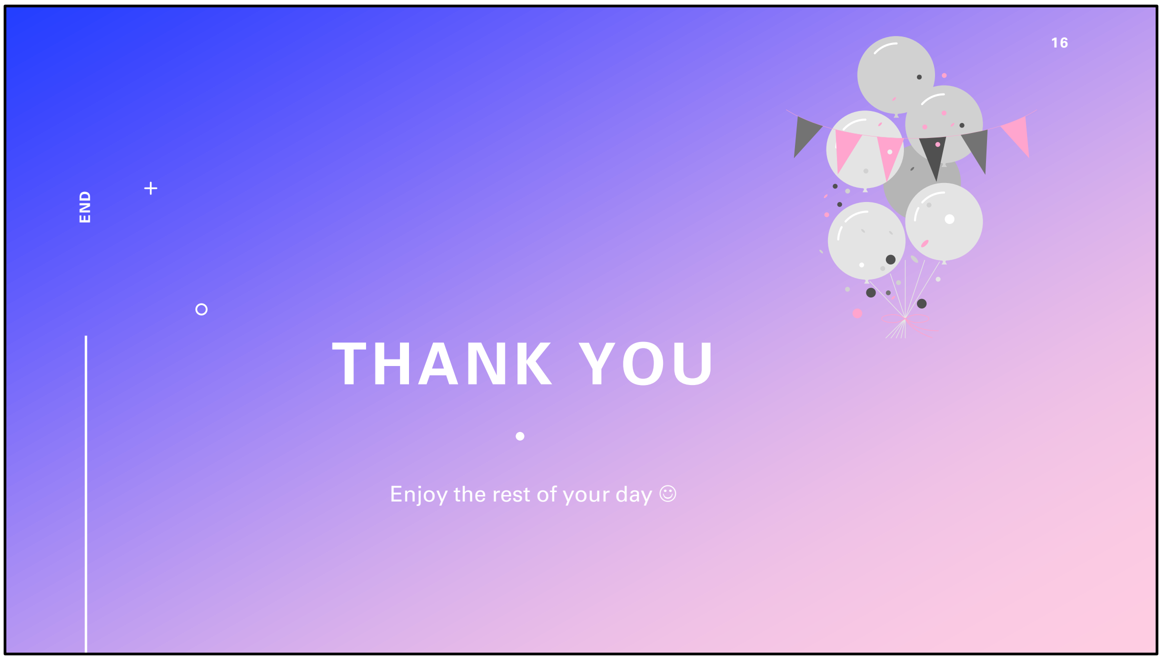
Favourite Parts

- Learning how to use Trello
- Emoji and colour packages
- THE END (seeing the end product)

Some of the challenges I experienced was firstly the error handling and testing. I had not had a good grasp on these two topics during class lessons so when it came do doing these for my app, I was very lost. I especially had trouble figuring out how to test functions in my code that had no parameters and had user inputs within the functions. After lots of research, I learnt how to use monkeypatch to solve this issue. As the lesson on bash scripting was independent learning, I had some trouble learning how to write bash scripts on my own.
I was also short on time and was having trouble trying to do the to-do list because it was taking a while to understand how to use the timedate module. In the end, I pushed through the documentation and had some spare time left to add it in.

I also had some difficulties with time management in regards with the PEP8 styling. I wrote all my code without adhering to any standard first and had to correct all the styling towards the end of the project which was very time consuming. In hindsight, learning and reading about the styling code before writing would've saved me a lot of stress and time.

My favourite parts of the project included learning how to use Trello. The implementation planning helped immensely and once I had that going, I had a clear direction for the project and a clear daily plan for the week. Although the colour and emoji additions were small features of the app, I had lots of fun adding them in as it was nice to see something bright and eye-catching pop up in the terminal. The best part of the project by far was the very end as this was when I saw the finished product. Seeing everything working after so many days of things NOT working was very satisfying and I’m really happy with the final result considering I was struggling quite a bit with some of the python concepts during class ☺



END

THANK YOU

Enjoy the rest of your day ☺