

WEDDING PLANNER APP

+
o •



Term 1: Assignment 3
By Michelle Liang





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

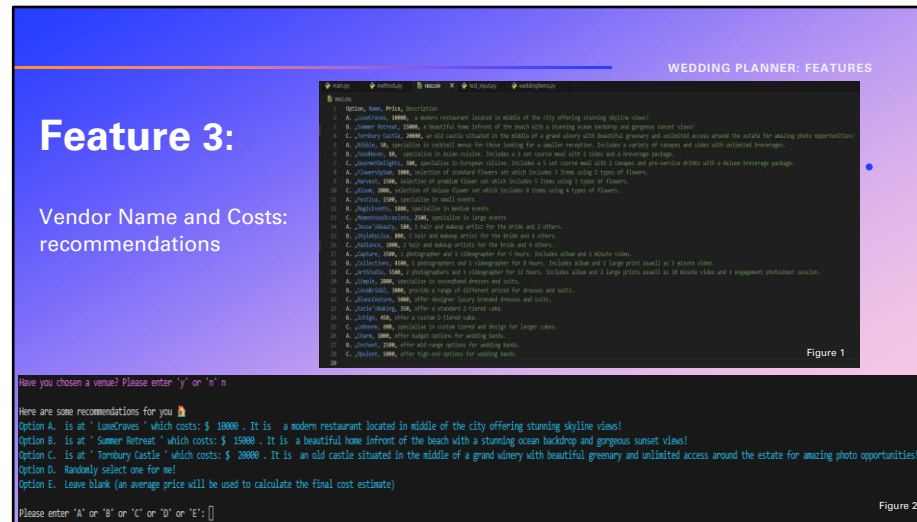
Have you chosen a event decorater? Please enter 'y' or 'n' y

Please enter your event decorater name: Splash

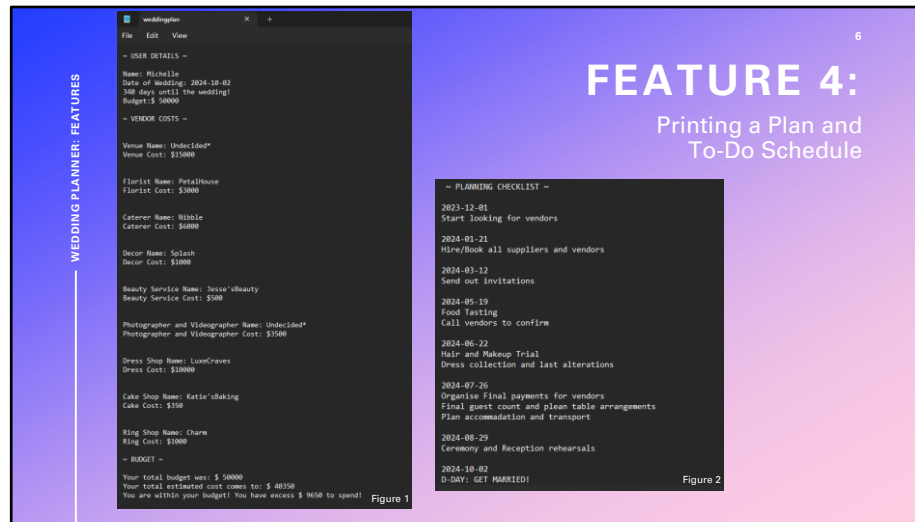
Please enter your event decorater cost: 1000

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.



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).



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).

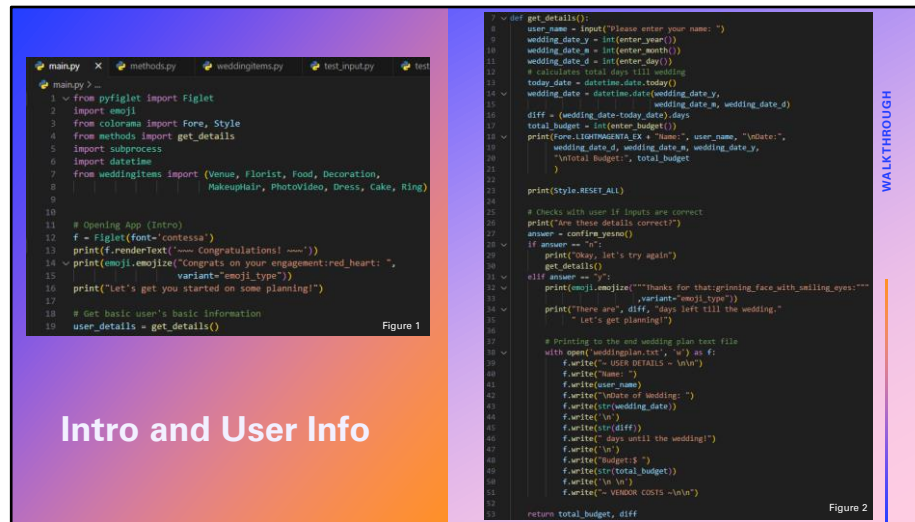
It will also print out a scheduled to-do list with dates to help the user tick off important tasks before their big day (figure 2).



Code Walkthrough:

First an introduction to the different files:

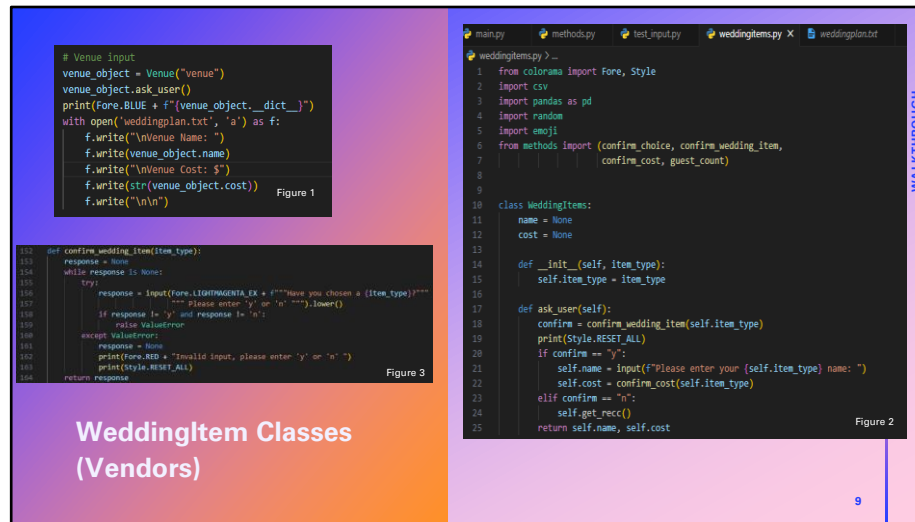
1. main.py: is the app's main file
2. methods.py: contains functions used in the main file
3. weddingitems.py: contains all the vendor classes used in the app
4. recc.csv: contains the data used to provide recommendations to the user
5. weddingplan.txt: is the text file outputted at the end with the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



Intro and User Info

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:

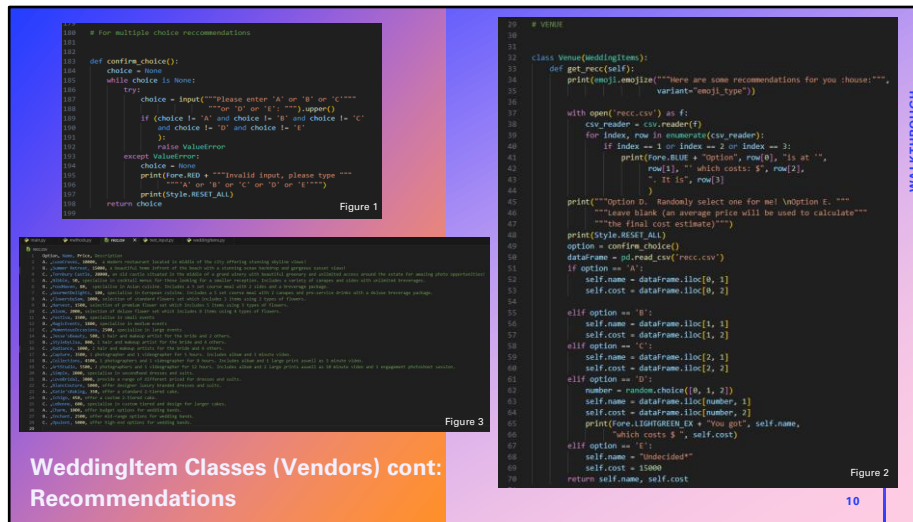
1. Asking for user's name and date of wedding (line 7 -11) and uses the latter and the `time.date` module to calculate how many days are left till the wedding (line 12-16)., storing this in the variable "diff".
2. It then asks the user for their budget (line 17) and prints their inputted details before 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.



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:



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.csv’ 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).

```
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 decorator")
55 decoration_object.ask_user()
56 print(Fore.LIGHTBLUE_EX + f"({decoration_object.__dict__})")
```

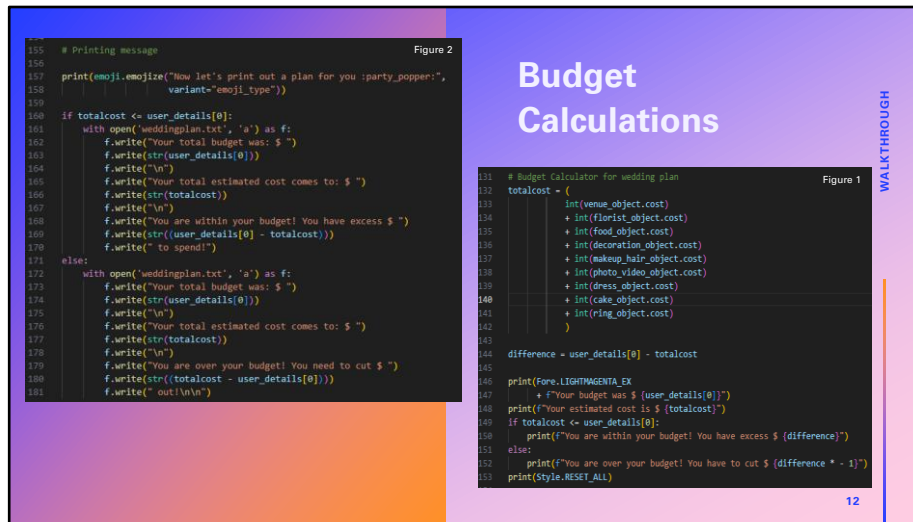
Figure 1

Writing vendors to text file

WALKTHROUGH

11

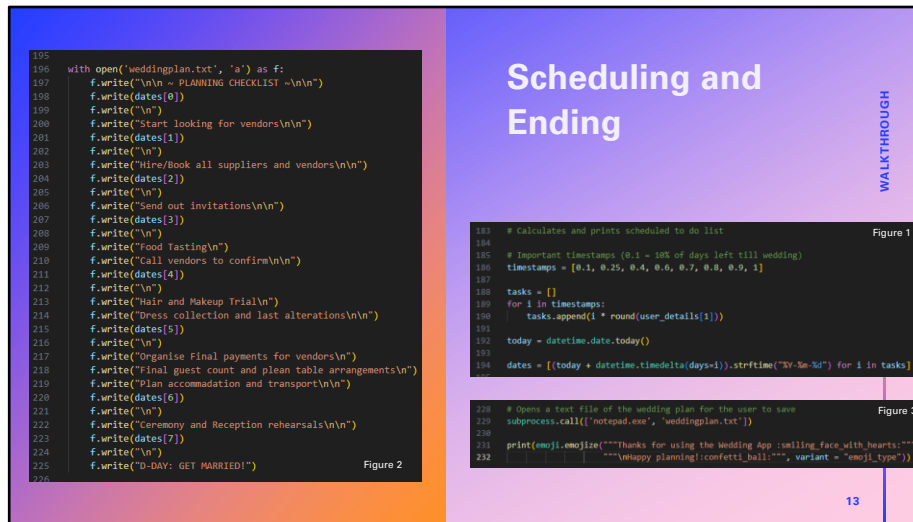
Once the user has inputted a vendor name and cost or picked a recommendation, `ask_user()` is completed. The app will then print out the dictionary of the class (lines 23, 34, 45) so that the user can see the inputs/choices and finally, the app will write down the name and cost to the `weddingplan.txt`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



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.



Finally, we have the scheduling feature. In figure 1, important timestamps are written in a list called timestamps (line 186). We then iterate through the list and multiply 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 the elements in the list. This gives us the number of days each timestamp equates to relative to the number of days till the wedding. A new list is created called dates in line 194 which iterates through 'tasks' and adds the days in task to today's date which gives us an exact date in dd/mm/yyyy format for each new element. 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 229, 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 231-232)!




```

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



```

180 # For multiple choice recommendations
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' or
188                           'D' or 'E': ").upper()
189             if (choice != 'A' and choice != 'B' and choice != 'C'
190                 and choice != 'D' and choice != 'E'):
191                 raise ValueError
192         except ValueError:
193             choice = None
194             print(fore.RED + "Invalid input, please type "
195                   + "'A' or 'B' or 'C' or 'D' or 'E'")
196             print(style.RESET_ALL)
197     return choice

```

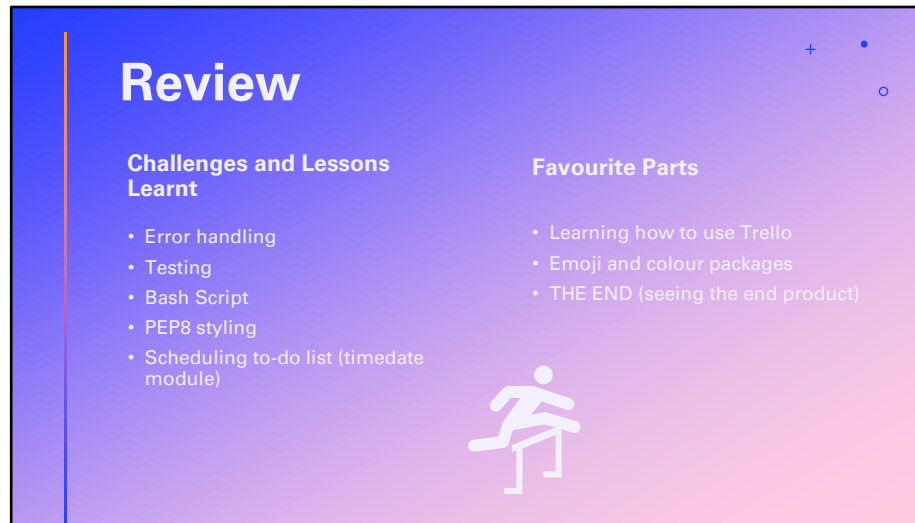
Figure 2

WALKTHROUGH

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.



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 to 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 😊

