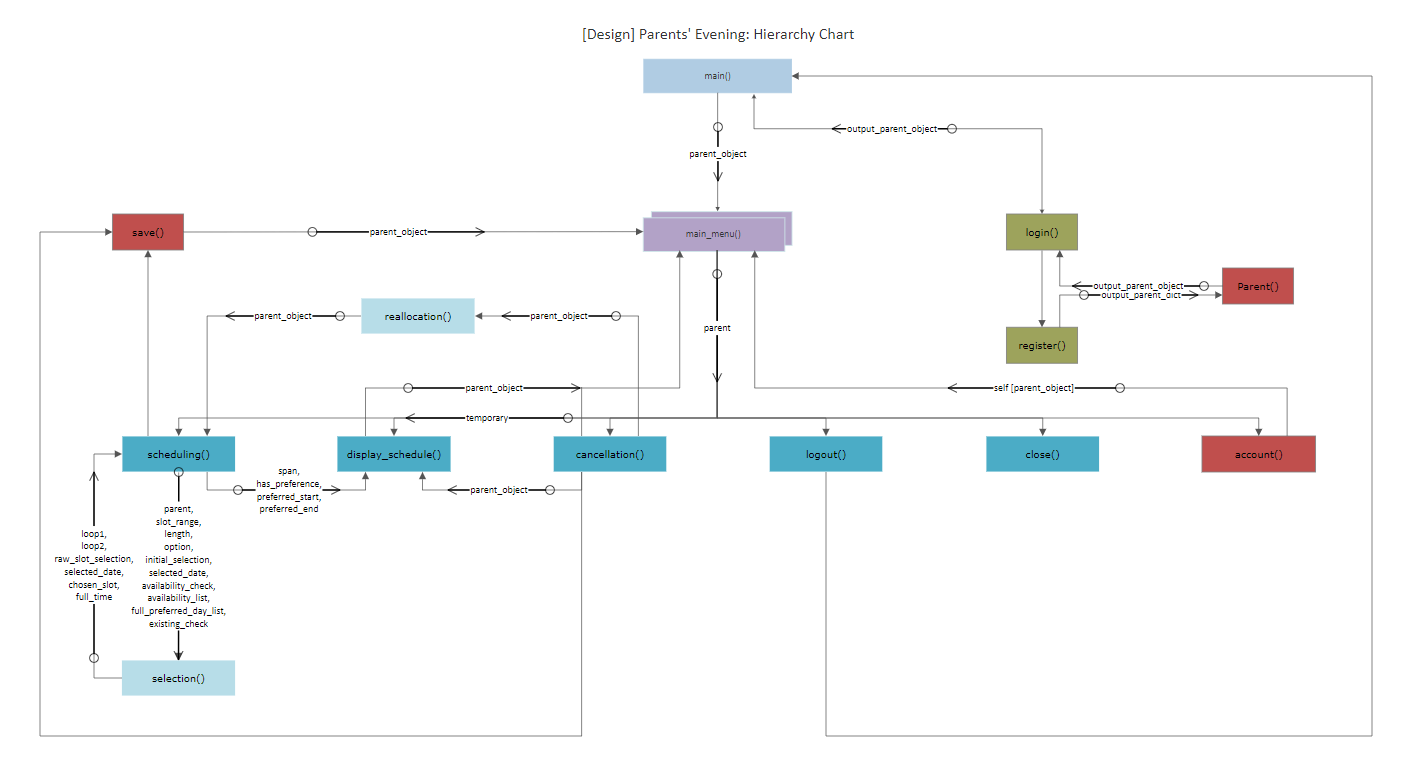
# *Report Sections*

## Analysis

* **Introduction**
  + Parents’ Evening: A short scheduled meeting or conferences between the parents and teachers of students to discuss both a child’s progress at school and find solutions to academic or behavioural problems. Since teachers will have dozens of students to interact with, a scheduling system must be implemented to allocate specific time slots to students, parents and teachers to reduce clutter, confusion and delays.
* **Requirements**
  + *Assign an ID as well as a parent’s information (incl. student information).*
  + *Receive one’s preferred day and time frame during the 3-day Parents’ Evening period.*
  + *Display the available time slots and fill them in on a first-come-first-serve basis.*
    - *Create logic to accommodate for conflicting time slots.*
  + *Allow the parent(s) to both delete and/or reallocate their scheduled meeting.*
  + *Provide a menu displaying multiple actions that the user can perform from a home page.*
  + *Output the final scheduled timetable to the parent/teacher.*
  + *Save any edited data to both the ‘parent’ and ‘schedule’ JSON files.*
* **Sub-problems** [**Decomposition:** The process of breaking down a large problem down into smaller, more manageable parts which are easier to solve.]
  1. Parent information
     + To uniquely identify each parent, this sub-problem should both verify and allow parents to register new information.
  2. Time allocation (inputs)
     + The main function of the program: receive a preferred day and time frame from the user; display available slots within the preference (or bordering available time slots if the former is unavailable); confirm the selection; convert to an object; store the data.
  3. Displaying the schedule
     + By incorporating a table-formatting library, display the schedule for both the teacher (one which may span the 3-day period) and the parent (one which may only span 1 day with their time slot included) depending on who is accessing the program.
  4. Cancellations and reallocations
     + Re-display the user’s selection and ask for confirmation to delete the time slot. If they choose to also reallocate their meeting, continue the function and link it back to the time allocation function.
  5. Exiting and saving the program (data)
     + Force quit the program when the user has completed their desired actions and write both their user information and scheduling data to a (JSON) file.
  6. Menu
     + Neatly formatted menu to allow the user to select an index to be redirected to the other functions performing different actions.
  7. Main program – information
     + Initial start-up of the program: introductory information to initiate the program.

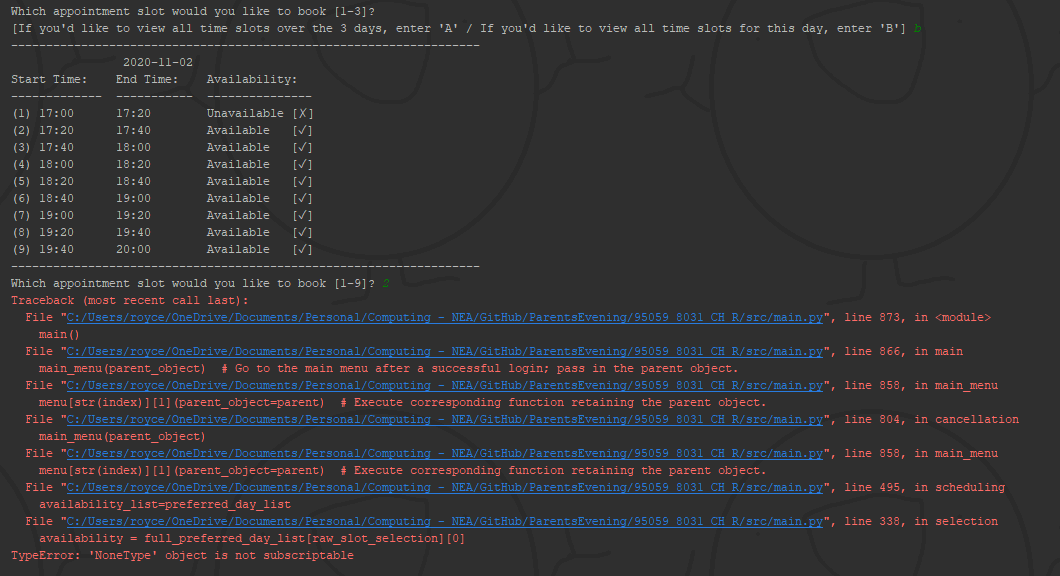
## Design

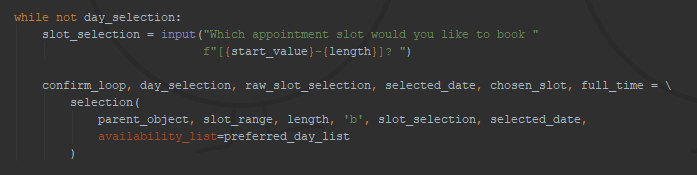
* **Solution Design Algorithms**

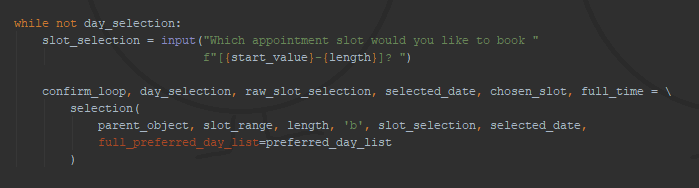
****

* **Individual Subroutine Plans – In order of appearance…**
  + **Parent.account()**
    - Prints the user’s information based on a parent object, which contains their username, unique user ID, children’s information and meeting information, if applicable.
  + **Parent.save()**
    - Either saves (writes) the user’s updated information (i.e., meeting information) or resets/clears the user’s meeting information following cancellation.
  + **register()**
    - If the user does not have a pre-existing account, register() creates a new parent’s profile, appends it to the parents.json file and returns a dictionary to login() to create a new parent object; it ensures the username is unique before its submission.
  + **login()**
    - Prompts the user to register into the system and creates a parent object to be passed around.
  + **preferences()**
    - If the user has a preferred time in which the meeting lies, preferences() prompts the user and receives their desired day, starting time of the preferred time frame/slot and ending time of the preferred time frame/slot.
  + **selection()**
    - Checks whether a selected meeting index is both within a given range and available; their provisional meeting is confirmed and passed back to scheduling().
  + **scheduling()**
    - Arranges a meeting with the user after receiving a selected slot number either via a preference or via the full 3-day schedule.
  + **display\_schedule()**
    - If the user has a scheduled meeting, display\_schedule() displays a single-day table indicating their chosen slot and returns to the menu. Furthermore, display\_schedule() can be used in relation to scheduling() when prompting the user for a selection (e.g., one-day display, full 3-day display, etc.).
  + **cancellation()**
    - Cancels the user’s meeting, which thus clears their ‘meeting’ key in parents.json and refreshes the parent object. The user is also prompted about whether they would like to reschedule their meeting after cancellation.
  + **reallocation()**
    - After the user has cancelled a meeting via cancellation(), reallocation() allows the user to reschedule a new meeting; this mirrors a standard scheduling() sequence.
  + **logout()**
    - Refreshes the parent object and ‘reruns’ the program back to the initial login page.
  + **close()**
    - Shuts down the program.
  + **main\_menu()**
    - Displays a range of actions a user can perform concerning the whole program: displaying account information, scheduling a meeting, displaying their schedule, cancelling/reallocating their meeting and accessing the ‘basic’ functions of entering and exiting a program.
  + **main()**
    - The ‘backbone’ of the program; it initiates and introduces the program.
* **Initial Test Plan**
  + *Included as part of the final and completed test plan.*

## Implementation

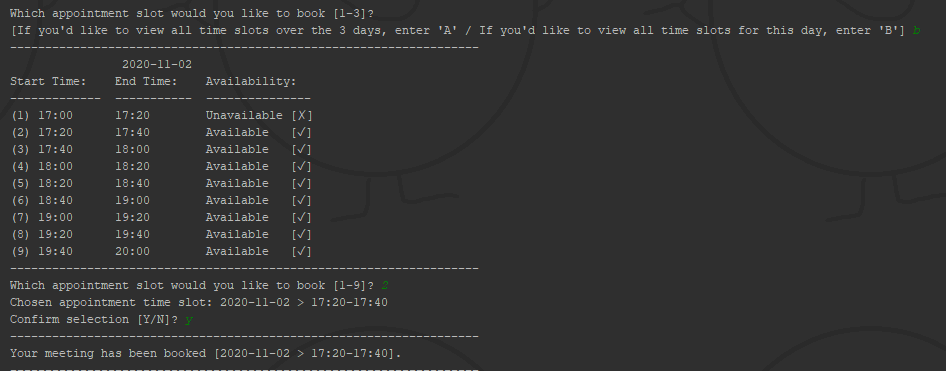
* **Program Code**
  + *Raw source code and executable files are included within this folder.*
  + *GitHub link:* [*https://github.com/RoyceLWC/ParentsEvening*](https://github.com/RoyceLWC/ParentsEvening)
* **Screenshots of Debugging Skills to Correct Errors + Algorithm Refinements and Justifications**
  + Bug #1 – Fixing the keyword arguments given to selection()
    - **Runtime Error –** When entering an appointment slot within the ‘b’ (full 1-day schedule) path, the program raised a *TypeError.*
    - **Broken code / error messages:**



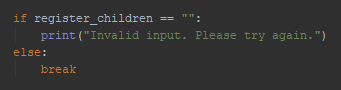
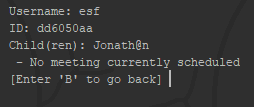
* + - **Fixed code / correct output:**

****

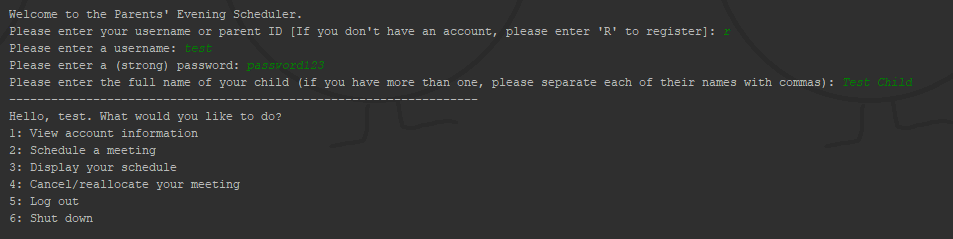
Reference to parameters:

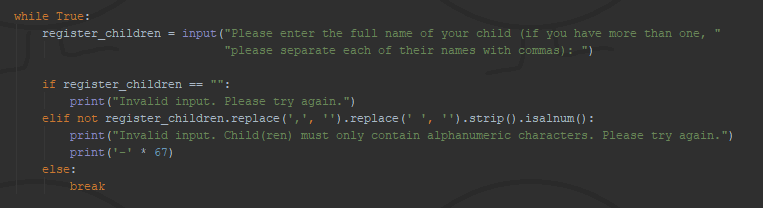
****

* + - **Changes:**
      * Rewrote the *keyword argument* given:
        + availability\_list ⟶ full\_preferred\_day\_list
  + Bug #2 – Ensuring child name input is properly validated
    - **Logic Error –** When entering a child’s name containing a symbol (e.g., ‘?’), the program continued the sequence and allowed the user to incorporate different (‘unwanted’) symbols.
    - **Broken code / ‘error’ messages:**



* + - **Fixed code / correct output:**





* + - **Changes:**
      * Included an extra ‘elif’ statement that took advantage of the .isalnum() method; ensured that the string was stripped of any commas/spaces beforehand, which would also raise an ‘error’ if the parent had multiple children.

## Testing

* **Completed Test Plan**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Stage 2 –** Initial Test Plan | | | | | | | | | | | | **Stage 3/4 –** Test Table | | | |
| **Test No.** | | | **Type of Data** | | | **Purpose of Test** | | **Test Data** | | **Expected Result** | | **Actual Result** | | **Action Needed/comments** | |
| 1 | | | Normal | | | To check whether main\_menu() index is valid and in range. | | 2 | | Executes scheduling() | | ✓ | |  | |
| 2 | | | Boundary | | | 0,1 | | “Out of range! Please try again.”, executes parent.account() | | ✓ | |  | |
| 3 | | | Erroneous | | | ‘hello’ | | “Invalid index. Please try again.” | | ✓ | |  | |
| 4 | | | Normal | | | To check whether the user can ‘escape’ out of parent.account() | | ‘B’, ‘b’ | | Returns to main\_menu(). | | ✓ | |  | |
| 5 | | | Boundary | | |  | |  | |  | |  | |
| 6 | | | Erroneous | | | 10, ‘abc’ | | No output; re-prints the escape prompt. | | ✓ | |  | |
| 7 | | | Normal | | | To check whether the user can choose to book an appointment based on a preferred day in scheduling() | | ‘y’, ‘n’ | | Executes preferences(), displays the full 3-day schedule. | | ✓ | |  | |
| 8 | | | Boundary | | |  | |  | |  | |  | |
| 9 | | | Erroneous | | | ‘ ’, ‘a1’ | | “Invalid input. Please try again.” | | ✓ | |  | |
| ‘y’ | | 10 | Normal | | | To check whether the user can select a day based on preferences. | | ‘Sun’, ‘TuEsdaY’ | | Stores their preferred day (object) and asks for their preferred time frame. | | ✓ | |  | |
| 11 | Boundary | | |  | |  | |  | |  | |
| 12 | Erroneous | | | 123, ‘Friday’ | | Prints ‘Invalid day entered. Please try again.’ | | ✓ | |  | |
| 13 | Normal | | | To check whether the user has inputted a valid time within a range. | | ‘17:30’ | | Stores their preferred starting time and asks for their ending time. | | ✓ | |  | |
| 14 | Boundary | | | ‘16:30’ | | Still continues to ask for their preferred ending time frame (but returns an error after). | | ✓ | |  | |
| 15 | Erroneous | | | ‘0:71’, 2 | | Still continues to ask for their preferred ending time frame (but later returns ‘Incorrect data format(s); it should be in the following format: hh:mm --- Please try again.’). | | ✓ | |  | |
| 16 | Normal | | | To check whether the user has inputted a valid time within a range. | | ‘18:00’ | | Returns the 3 values and goes back to scheduling(). | | ✓ | |  | |
| 17 | Boundary | | | ‘21:00’ | | Prints ‘Invalid time input(s); the times inputted do not fit in the same time frame and/or are in the wrong order. Please try again.’ | | ✓ | |  | |
| 18 | Erroneous | | | ‘17:00’ **(Before start time)** | | Prints ‘Invalid time input(s); the times inputted do not fit in the same time frame and/or are in the wrong order. Please try again.’ | | ✓ | |  | |
| 19 | Normal | | | To check whether the user can either select an appointment or view a different formatted schedule. | | ‘a’, ‘b’, ‘6’ | | Displays full 3-day schedule, displays full 1-day schedule, executes selection(). | | ✓ | |  | |
| 20 | Boundary | | | 0, 10 | | Executes selection() but prints “Invalid input; your selection is out of range. Please try again.” | | ✓ | |  | |
| 21 | Erroneous | | | ‘c’ | | Executes selection() but prints “Invalid input. Please try again.” | | ✓ | |  | |
| ‘a’ | 22 | Normal | | To check whether the user can select an appointment slot within a full 3-day range. | | 1,8,27 | | Enters selection() and confirms their selection. | | ✓ | |  |
| 23 | Boundary | | 0,29 | | Enters selection() but prints ‘Invalid input; your selection is out of range. Please try again’. | | ✓ | |  |
| 24 | Erroneous | | ‘no’ | | ‘Enters selection() but prints ‘Invalid input. Please try again’. | | ✓ | |  |
| ‘b’ | 25 | Normal | | To check whether the user can select an appointment slot within a full-day range. | | 2,3,7 | | Enters selection() and confirms their selection. | | ✓ | |  |
| 26 | Boundary | | 0,10 | | Enters selection() but prints ‘Invalid input; your selection is out of range. Please try again.’ | | ✓ | |  |
| 27 | Erroneous | | ‘yes’ | | ‘Enters selection() but prints ‘Invalid input. Please try again.’ | | ✓ | |  |
| int | 28 | Normal | |  | |  | |  | |  | |  |
| 29 | Boundary | |  | |  | |  | |  |
| 30 | Erroneous | |  | |  | |  | |  |
| ‘n’ | | 31 | Normal | | | To check whether the user can select an appointment slot within a full day range. | | 1,5,20 | | Enters selection() and confirms their selection. | | ✓ | |  | |
| 32 | Boundary | | | 0,50 | | Enters selection() but prints ‘Invalid input; your selection is out of range (or already occupied). Please try again.’ | | ✓ | |  | |
| 33 | Erroneous | | | ‘one’ | | ‘Enters selection() but prints ‘Invalid input. Please try again.’ | | ✓ | |  | |
| 34 | | | Normal | | | To check whether a user can confirm their selection. | | ‘yes’, ‘nO’ | | Returns to scheduling() and establishes their selection (both writing and updating objects), prints ‘Restarting selection…’ and asks them for another appointment slot. | | ✓ | |  | |
| 35 | | | Boundary | | |  | |  | |  | |  | |
| 36 | | | Erroneous | | | 7 | | Prints ‘Invalid input. Please try again.’ | | ✓ | |  | |
| 37 | | | Normal | | | To check whether a user can confirm the cancellation of their meeting. | | ‘yEs’, ‘n’ | | Reinstates the parent object and prompts them about a potential reallocation, prints ‘Returning back to menu…’ | | ✓ | |  | |
| 38 | | | Boundary | | |  | |  | |  | |  | |
| 39 | | | Erroneous | | | ‘?’ | | Prints ‘Invalid input. Please try again.’ | | ✓ | |  | |
| 40 | | | Normal | | | To check whether a user can have the option to reallocate their meeting. | | ‘Y’, ‘No’ | | Prints ‘Your meeting has been cancelled. Proceeding to scheduler’, prints ‘Your meeting has been cancelled. Returning back to menu…’ [cut short] | | ✓ | |  | |
| 41 | | | Boundary | | |  | |  | |  | |  | |
| 42 | | | Erroneous | | | ‘ ’, ‘!’ | | Prints ‘invalid input. Please try again.’ | | ✓ | |  | |
| 43 | | | Normal | | | To check whether the user can input their username. | | ‘user1’, ‘r’ | | Continues to password input, executes register() | | ✓ | |  | |
| 44 | | | Boundary | | |  | |  | |  | |  | |
| 45 | | | Erroneous | | | *Unmatching username* | | Continues to password input but prints ‘The username/parent ID and/or password you’ve entered doesn’t match a registered account. Please try again.’ | | ✓ | |  | |
| 46 | | | Normal | | | To check whether the user can input their password. | | ‘password123’ | | If matching, returns output\_parent\_object and goes back to main\_menu(). | | ✓ | |  | |
| 47 | | | Boundary | | |  | |  | |  | |  | |
| 48 | | | Erroneous | | | *Unmatching password, ‘ ’* | | Prints ‘The username/parent ID and/or password you’ve entered doesn’t match a registered account. Please try again.’, prints ‘Field left empty, please try again.’ | | ✓ | |  | |
| 49 | | | Normal | | | To check whether the user can register a (unique) username. | | ‘user2’ | | If unique, continues to password input. | | ✓ | |  | |
| 50 | | | Boundary | | |  | |  | |  | |  | |
| 51 | | | Erroneous | | | *An already registered username, ‘ ’* | | Prints ‘Username ‘’ has already been taken. Please try again.’, prints ‘Invalid input. Please try again.’ | | ✓ | |  | |
| 52 | | | Normal | | | To check whether the user can register a new password. | | ‘password789’ | | Continues to child(ren) input. | | ✓ | |  | |
| 53 | | | Boundary | | | ‘pass7’ | | Prints ‘Your password is too short. Please try again.’ | | ✓ | |  | |
| 54 | | | Erroneous | | | ‘ ’, ‘password’ | | Prints ‘Invalid input. Please try again.’, prints ‘Your password must contain at least one digit. Please try again.’ | | ✓ | |  | |
| 55 | | | Normal | | | To check whether the user can register their child(ren). | | ‘Mary, John’, ‘Tom’ | | Returns to login() with a list of the child(ren) assigned to the parent object. | | ✓ | |  | |
| 56 | | | Boundary | | |  | |  | |  | |  | |
| 57 | | | Erroneous | | | ‘ ’, ‘?’ | | Prints ‘Invalid input. Please try again.’ | | Allowed the user to enter a child name containing symbols. | | Make a check to ensure all characters are alphanumeric. | |
| (57) | | (Erroneous) | | | To re-check the user’s ability to enter symbols in a child’s name. | | ‘?’ | | Prints ‘Invalid input. Child(ren) must only contain alphanumeric characters. Please try again.’ | | ✓ | |  | |

## Evaluation

* **Evaluation and Reflection of Key Requirements of the Program**
  + *Assign an ID as well as a parent’s information (incl. student information).*
    - In the logging-in and registering phase, the user can successfully provide their desired account details including a simple username/password and child names.
    - Here is the list of things that can be displayed and their impacts:
      * **Unique user ID** – ID generated from the first 8 characters of a UUID4 random generation successfully produces a unique string of alphanumeric characters to distinguish each individual parent and mitigate the chances of overlapping meetings.
      * **Username** – A desired string of characters that is first checked to see if it has already been taken/is duplicated to, again, ensure there are no potential overlaps.
      * **Children –** The parent’s child(ren): I made sure to include this for future refinements to allow for multiple children, born in the same academic year, to be booked into the system under one parent. However, I did not include any dates of birth or nationalities as I did not feel like they were essential/necessary when it came to booking meetings. In the future, the children could be given their own ID and have ‘teacher notes’ assigned to them to incorporate another element of functionality from a teacher’s perspective.
      * **Meeting –**  The (provisional) scheduled meeting: user’s can be shown a concise line of the date and time of their meeting. If the do not have a meeting schedule, I made sure to include a note so that they could be prompted to eventually go and schedule one.
  + *Receive one’s preferred day and time frame during the 3-day Parents’ Evening period.*
    - Overall, I feel like I excelled within this requirement as I went beyond the prompt asking for parents to send in their ‘two preferred appointment times’. Instead of limiting parents to 2 preferences, I created a system where parents could choose their preferred *day* and then continue to ask about a preferred *time frame*, which could potentially span all slots in that day, three or just one. Furthermore, I ensured that, if the time frame contained no available slots, the user would still be prompted with *bordering* available appointments so that the user would not get ‘stuck’ mid-way through selection.
    - In terms of validation, I added in extra flexibility when it came to selecting a preferred time frame: users could input either the standard 3-letter day abbreviations or the full day’s name and provide a simplistic time input with a declared format of ‘hh:mm’ which would then create a full datettime object.
  + *Display the available time slots and fill them in on a first-come-first-serve basis.*
    - Although I used an external library, I would say that using ‘Tabulate’ was vital to neatly presenting the start and end time of each slot and where exactly the parent’s selected appointment lied as it provides a structured and distinctly divided table; it was also useful in aiding the user during the appointment selection as I could clearly indicate which meetings were available and their corresponding slot number.
    - Above each table, I clearly stated the date each table represented above to reduce the likelihood of confusion when it came to selecting an appointment with a selection spanning the full 3 days.
  + *Allow the parent(s) to both delete and/or reallocate their scheduled meeting.*
    - Included within the ‘refinements’ section, I managed to add in the ability to both delete and reallocate one’s meeting and have it overwrite any JSON files/object data. The reallocation linked nicely back to the scheduling() function so that there would be a sense of consistency throughout the code, which would, in turn, simplify the code and take advantage of the fact my program consisted of reusable modules.
  + *Provide a menu displaying multiple actions that the user can perform from a home page.*
    - After having logged in, users are met with a concise and self-explanatory menu that provides a gateway to all the other functions (options). In addition, I used a dictionary to pair each index value (i.e., the option) with its corresponding function; this allowed the program code to be even more concise since the selected index could immediately be executed without further ‘selection’.
  + *Output the final scheduled timetable to the parent/teacher.*
    - Although I was unsuccessful with creating an ‘admin’ user (i.e., for teachers) that could view the full timetable, I was able to neatly display the parent’s scheduled meeting in relation to the displaying of time slots. Again, display\_schedule() was critically adapted in order to be reused throughout the program either during selection or during a final overview of one’s meeting(s).
  + *Save any edited data to both the ‘parent’ and ‘schedule’ JSON files.*
    - Since this programming project required some form of recording user/meeting details, I deliberately chose to work with JSON files to extend my knowledge of writing data (beyond ordinary text files) as well as to exploit the dictionary-like format to incorporate user data and meeting data neatly with the ‘Parent’ class.
* In conclusion, I would say that everything covered in my NEA has successfully met the requirements of the brief/prompt I chose. Although I was unable to add in the twin functionality extension, I ensured that my program was adaptable and provided the ‘back-bone structure’ for myself to complete outside this project’s deadline in the future. Through vigorous validation checks and user-friendly interfaces throughout, this program is highly functional and allows for users to schedule appointments, cancel/reallocate appointments and view their final scheduled meeting within a full-day table; all of this is saved to two JSON files and is always re-accessible when re-run.

## References

* **Any sources of information you used to complete the project. – Name of source/Date of access**
  + **Documentations:**
    - Using the datetime library *[15/02/2021]:*
      * <https://docs.python.org/3/library/datetime.html>
  + **Stack Overflow:**
    - Double-except statements (using parenthesised tuples) *[17/02/2021]:*
      * <https://stackoverflow.com/questions/6470428/catch-multiple-exceptions-in-one-line-except-block>
  + **Programiz**
    - Revising object-oriented programming *[09/02/2021]:* <https://www.programiz.com/python-programming/object-oriented-programming>
  + **Real Python**
    - Revising how to read from and write to JSON files *[11/02/2021]:*
      * <https://realpython.com/python-json/>