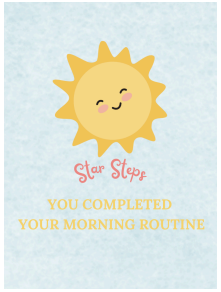
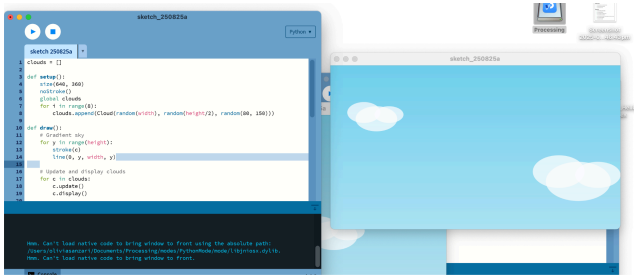
STAR STEPS CODING JOURNAL

WEEK	GOAL	ACHIEVED	CHALLENGES
WEEK 6	→ create visuals and finalise them    	→ Prototype interfaces for the Morning and Night Routines were developed using Canva to design the visual layout. A dedicated project folder, titled **"Star Steps"**, was created on the desktop and integrated with Visual Studio Code to facilitate streamlined development. The designs were implemented into the application as follows: **"star_steps_bg.png"** represents the initial page of the routine, **"StarSteps2.png"** corresponds to the conclusion of the morning routine, and **"StarSteps3.png"** serves as the interface for the night routine. Pre-existing code →(GeeksforGeeks, 2019) → (Python Software Foundation, 2019.) → Previous Python scripts were leveraged to incorporate the **Canvas setup**, establishing the design's dimensional framework, while the **load_background** function was employed to render the visual backgrounds, enabling the integration of the custom interfaces created in Canva.  → Using preexisting code I was able to identify the geometry and size of the window of the application (Python, 2025) Window Geometry	→ Experimenting with processing and visual studio code,  → Using processing I attempted to communicate a visual sky however I didn't aesthetically appeal to it so i decided to consider using canvas set up → To locate instructional resources that facilitate understanding and coding the integration of images within Visual Studio Code. → To systematically investigate and educate myself on the step-by-step process of implementing images in Visual Studio Code through

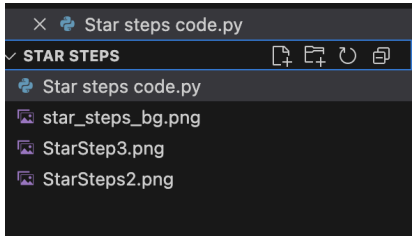
```
class StarStepsApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Star Steps")
        self.root.geometry("450x900")
        self.root.configure(bg='#C8E6ED')
```

Image Geometry

```
# Canvas setup
self.canvas = Canvas(root, width=450, height=900, bg='#C8E6ED', highlightthickness=0)
self.canvas.pack(fill=tk.BOTH, expand=True)
```

Load as the background

```
# Images
self.bg_image = None
self.task_bg_image = None
self.final_image = None
self.final_step_image = None
self.load_background()
```

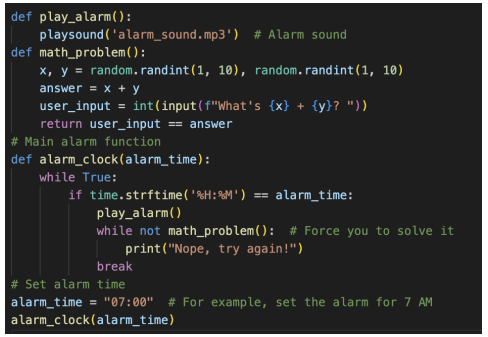


a designated project folder, including utilising the “Select File” functionality.

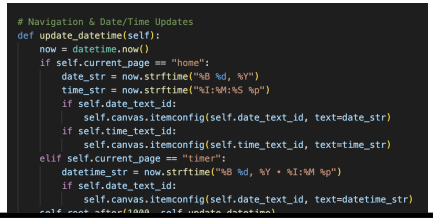
```
from PIL import Image, ImageTk
```

→ Coding the measurements of the images was difficult to understand, however through experimenting with different measurements finally achieved success.

```
def load_background(self):
    try:
        if os.path.exists('star_steps_bg.png'):
            bg_img = Image.open('star_steps_bg.png').resize((450, 900), Image.Resampling.LANCZOS)
            self.bg_image = ImageTk.PhotoImage(bg_img)
        if os.path.exists('task_page_bg.png'):
            task_img = Image.open('task_page_bg.png').resize((450, 900), Image.Resampling.LANCZOS)
            self.task_bg_image = ImageTk.PhotoImage(task_img)
        if os.path.exists('StarSteps2.png'):
            final_img = Image.open('StarSteps2.png').resize((450, 900), Image.Resampling.LANCZOS)
            self.final_image = ImageTk.PhotoImage(final_img)
        if os.path.exists('StarStep3.png'):
            final_step_img = Image.open('StarStep3.png').resize((450, 900), Image.Resampling.LANCZOS)
            self.final_step_image = ImageTk.PhotoImage(final_step_img)
```

WEEK 7	→ test morning routine → complete interactive alarms → timed notification → refine user interface design	→ Designed the morning routine interface and conducted research to determine the most effective step-by-step routine for children with ADHD, which informed the development of a structured sequence of actions. → Refined the user interface by iteratively testing the step backgrounds—for example, **"Get Up & Make Your Bed."** While the initial design incorporated a timer for each step, the final implementation prioritised displaying the current date and time, featuring a prominent central timer alongside Start and Reset buttons with a 5-minute countdown, aiming to optimise user engagement and productivity. Wake Up & Make Your Bed	→Interactive alarm functionality proved challenging, so its development was deferred to a subsequent week, as the tutorials consulted consistently presented issues within Visual Studio, and the timer interface dimensions did not align with my design vision, as when I included the alarm, the visuals wouldn't play. Code from Soares (2021)  → I decided not to implement time-based notifications, as the visual countdown provided within the interface sufficiently conveys temporal progress. However, I remain undecided regarding the inclusion of auditory alarms, given repeated difficulties in integrating sound with the code, which proved both time-consuming and technically frustrating.
WEEK 8	→ develop evening and night routines → generate user interface such as buttons, schedule, and the interactivity of the app	→While developing the night routine, I decided to remove the evening routine feature, as scheduling this period proved unpredictable and varied significantly between children. Instead, I focused on creating a more structured morning and night routine that could accommodate a wider range of users.	→ Finding pre existing code that was suitable and aesthetically pleasing for the interface, having all features in the correct spots that I intended it to.

```
def play_alarm():
    playsound('alarm_sound.mp3')  # Alarm sound
def math_problem():
    x, y = random.randint(1, 10), random.randint(1, 10)
    answer = x + y
    user_input = int(input(f"What's {x} + {y}? "))
    return user_input == answer
# Main alarm function
def alarm_clock(alarm_time):
    while True:
        if time.strftime('%H:%M') == alarm_time:
            play_alarm()
            while not math_problem():  # Force you to solve it
                print("Nope, try again!")
            break
# Set alarm time
alarm_time = "07:00"  # For example, set the alarm for 7 AM
alarm_clock(alarm_time)
```

	→ research how to customise with the step by step routine → generate customised alarm on specific wake up	→ During this process, I gained a stronger understanding of how to implement colour within the interface. I selected the background colour **#C8E6ED** (a hexadecimal colour code) because it complemented the overall aesthetic and evoked a sense of calm and morning freshness. By examining pre-existing Python code, I learned how to effectively apply colour within the program's design, enhancing both the functionality and visual coherence of the application. def init (self, root) : `fill='#FF8A80', justify='center')` `fill='#FF8A80', justify='center')` self. root = root self. root . title ("Star Steps") self. root. geometry ("450x900) self. root. Configure (bg=#C8E6ED) `self.canvas.create_rectangle(0, 0, 450, 900, fill='#C8E6ED', outline='')` → developing customised alarms however most children have different times of wake up so I was indecisive to implement an alarm.	
WEEK 9	→ finalise the morning routine → finalise the alarms → finalise the buttons → finalise and implement real time clock → introduce a timer → develop on the evening and night routine	→ still working on the morning routine → implementing datetime of the date and the time throughout for the user to identify throughout their steps → I decided to get rid of the alarms and that can be a separate application for children as 7:00 does not cater to all children. → introduced a timer and a count down all minutes are different depending on each step → starting on the night routine	→ I initially considered implementing an alarm feature to make the application more inclusive, as children have different wake-up times. However, I found it difficult to accommodate varying schedules with a single, fixed alarm. As a result, I decided to remove the alarm function and instead allow users to rely on external alarm technologies that better suit individual needs. 

```
# Navigation & Date/Time Updates
def update_datetime(self):
    now = datetime.now()
    if self.current_page == "home":
        date_str = now.strftime("%B %d, %Y")
        time_str = now.strftime("%I:%M:%S %p")
        if self.date_text_id:
            self.canvas.itemconfig(self.date_text_id, text=date_str)
        if self.time_text_id:
            self.canvas.itemconfig(self.time_text_id, text=time_str)
    elif self.current_page == "timer":
        datetime_str = now.strftime("%B %d, %Y • %I:%M %p")
        if self.date_text_id:
            self.canvas.itemconfig(self.date_text_id, text=datetime_str)
```

		`` task_data = { 1: ("Wake Up & Make Your Bed", 5), 2: ("Time to Brush Your Teeth\n& Wash Your Face", 5), 3: ("Get Dressed", 5), 4: ("Have Breakfast", 20), 5: ("Pack All School Supplies,\nLunch Box & Water Bottle\nIn Your School Bag", 5), 6: ("Time for School", 360), 7: ("Dinner is Finished,\nTime To Have A Shower", 5), 8: ("Brush Your Teeth", 2), 9: ("Lay Out Your Clothes &\nBackPack For The Next Day", 5), 10: ("Read Together", 15), 11: ("Go To Bed", 0) } ``` Button geometry ``` def draw_rounded_button(self, x, y, width, height, radius=50, tag='button_bg'): points = [x + radius, y, x + width - radius, y, x + width, y, x + width, y + radius, x + width, y + height - radius, x + width, y + height, x + width - radius, y + height, x + radius, y + height, x, y + height, x, y + height - radius, x, y + radius, x, y ``` ``` #Timer Functions def toggle_timer(self, event=None): self.timer_running = not self.timer_running if self.timer_running: self.canvas.itemconfig(self.start_stop_btn_text, text="Stop", fill='#FF9D6D') self.run_timer() else: self.canvas.itemconfig(self.start_stop_btn_text, text="Start", fill='#4CAF50') def reset_timer(self, event=None): self.timer_running = False # Reset to the appropriate time for current step task_data = { 1: 2, 2: 2, 3: 2, 4: 2, 5: 2, 6: 2, 7: 5, 8: 2, 9: 5, 10: 15, 11: 2 } timer_minutes = task_data.get(self.current_step, 2) self.total_seconds = timer_minutes * 60 self.canvas.itemconfig(self.timer_display_id, text=self.format_time(self.total_seconds)) self.canvas.itemconfig(self.start_stop_btn_text, text="Start", fill='#4CAF50') ``` The Buttons geometry I found difficult understanding it and making the buttons measurements the way I wanted them to be (stackoverflow, 2017) used to help
WEEK 10	→ finalised the evening and night routine → implement an loop for the morning → ensure all interface is running correctly	→ finalised the morning routine with 6 steps → in development of the night routine → all interface from morning and middle of the night routine is running correctly  	→ The main challenge this week was that the night routine was not functioning as smoothly as the morning routine. The night interface behaved unpredictably, and each time I added new code, it resulted in recurring issues that disrupted its performance.
WEEK 11	→ finalise everything	→ created a loop each morning → in development of the night routine ``` def main(): root = tk.Tk() app = StarStepsApp(root) root.mainloop() if __name__ == "__main__": main() ```	→ the loop wasn't running correctly and it was challenging as the code terminal always seem to have a problem ``` if self.current_step == 7: # After step 6, show StarSteps2 self.draw_final_page() elif self.current_step == 12: # After step 11 (Go To Bed), loop back to home self.draw_home_page() else: self.timer_running = False self.draw_timer_page() ```

WEEK 12	→ development of the reflection of the assignment	→I finalised all aspects of the night routine this week, focusing on refining the interface design. I decided to enlarge the background to achieve a more balanced layout, which required adjusting the pixel measurements in the code to ensure the dimensions aligned precisely with the visual design.	→ Attempting to code the measurements differently thus creating visual problems with the interface.
WEEK 13	→ finishing touches make sure everything is correct and works → submit	→ All interface is done → All functionality of the application is working → Everything is finalised	→ The only problem was time constraints due to finalising everything with this application.

Original Timeline

Development for Week 2 to Week 11

WEEK 2	WEEK 11