Reflective Report: *Dear Acne*

Taya Bailey / 24916772
Subject: 52685 Working with Data & Code
Project: Dear Acne: Skin & Mood Tracker Web App

This reflection explores my experience developing *Dear Acne*, a prototype skin-and-mood tracker built in Python using Anvil. As a Communications student with no prior coding experience, this project became a crash course in code literacy, logic, and patience. The process was full of trial and error , mostly error , but it helped me understand how code, data, and design intersect in digital communication. Using Moon's (2006) reflective learning model and Norman's (2013) feedback-loop theory, I analyse key learning moments through the *What? So What? Now What?* framework. My insights also draw on peer feedback from Ruby and Saffanah, my project journal, and course readings.

What?

Debugging Form Hierarchies:

My biggest challenge was connecting multiple forms in Anvil's parent–child hierarchy. Every time I tried to switch screens, the same error appeared:

ModuleNotFoundError: No module named 'DashboardForm'

No matter what I changed , renaming components, checking imports, copying tutorial code , the error persisted. After hours of troubleshooting with classmates and AI tools, I discovered that my forms were accidentally **nested** within each other in Anvil's App Browser. *MainForm* contained *EntryForm*, which contained *DashboardForm* and *HistoryForm*, creating circular dependencies that prevented module imports. The fix wasn't syntactical but **structural** , dragging all forms to the same level so they became siblings.

Immediately after fixing hierarchy, a new error appeared:

TypeError: 'NoneType' object is not iterable

My Plot components expected data that didn't yet exist. I learned that Anvil components must be initialised *after* self.init_components() (Appendix A, Code Snippet 1). This moment taught me to think computationally , debugging wasn't about quick fixes, but understanding *why* the system behaved a certain way.

When *HistoryForm* continued to throw binding syntax errors, I deleted it entirely and replaced its **RepeatingPanel** with a simpler **DataGrid**. Ruby later suggested this was a good call , “sometimes rebuilding is faster than rescuing messy code.” That feedback made me more comfortable restarting rather than clinging to broken logic.

Connecting Data Tables:

Linking Anvil Data Tables so users could log daily mood and skin entries became another steep learning curve. My server code referenced app_tables.skin_entries, but my actual table was named entries, causing silent failures. Through UTS’s *Code Conundrums: Debugging Exercises* (2025) and discussions with Saffanah, I realised consistency between schema and code was vital. I corrected the mismatch (Appendix A, Snippet 2) and data finally persisted , a small but thrilling success.

This process revealed that what users see on-screen is only a fraction of the system; the “invisible” backend structures are equally crucial. Understanding this relationship between front-end interaction and backend logic transformed how I approached digital design.

Using Anvil’s Plot Components:

Initially, I planned to import Matplotlib for visualising mood patterns, but soon learned Anvil already includes built-in Plotly components. Once my forms and data tables were working, I created a function to convert text values into numeric scores (e.g. “Clear” = 4, “Severe” = 1) and plotted them using calming colours of sage green and lavender (Appendix A, Snippet 3). Seeing my first live graph render was a breakthrough moment , a visual confirmation that all my debugging finally aligned.

Saffanah’s peer feedback was to “keep it simple , prove the concept first before worrying about aesthetics.” Her advice helped me prioritise practicality and stability over design polish.

So What?

Each technical obstacle became a catalyst for deeper reflection. Moon (2006) argues that reflection transforms confusion into understanding , and every error forced me to pause, hypothesise, and iterate. Norman’s (2013) feedback-loop theory also came alive through this process: each error message acted as real-time feedback guiding my next decision.

For example:

- ModuleNotFoundError: taught me about Anvil’s hierarchy and sibling relationships.

- TypeError: revealed how data binding must follow component initialisation.

- SyntaxError: showed me when simplification (DataGrid > RepeatingPanel) was the smarter route.

These patterns turned frustration into logic. I began to see debugging not as failure but as communication between coder and system , a feedback dialogue.

Kuniavsky (2003) writes that prototypes are meant to evolve, not be perfect. My prototype embodied this principle, moving from non-functional chaos to a stable proof of concept. Each rebuild , nested forms, flattened forms, string-based navigation, taught new lessons in architecture.

Deci and Ryan's (1985) self-determination theory explains why I stayed motivated. Having autonomy to design my own idea, plus the intrinsic satisfaction of solving small problems, kept me engaged despite setbacks. Collaborating with ChatGPT and Claude AI introduced me to how modern developers work, asking precise questions, testing suggestions, and deciding what to implement. This human-AI partnership became an unexpected learning tool.

Ruby and Saffanah's peer feedback also grounded my reflection. Both advised me to **scale down my concept** and focus on getting the core data-tracking functionality working instead of perfecting every aesthetic detail. Their perspective helped me refine my scope and reminded me that usability outweighs visual perfection, a valuable mindset for any designer.

Ultimately, I realised debugging and reflective learning share the same cycle: observe then test then reflect then adjust. The iterative rhythm mirrors communication theory itself, where messages are sent, feedback is received, and meaning is refined.

Now What?

This project fundamentally changed how I perceive technology. Coding is not a series of right answers , it's a creative process requiring logic, experimentation, and reflection.

Key lessons I will carry forward:

1. **Start with structure, not features.** A simple system diagram of form relationships and data flow would have prevented days of circula import errors.

2. **Test incrementally.** Add one feature, test it, then proceed. This keeps debugging manageable.

3. **Understand the framework's logic.** Each environment (like Anvil) has rules; learning them early saves frustration.

4. **Embrace recreating over fixing.** Deleting broken code is not failure , it's clarity.

5. **Use simpler alternatives.** Simplicity equals efficiency; DataGrid was proof.

6. **Document everything.** I'll maintain a debugging log and use version control to track progress.

My final prototype includes:

- User authentication
- Four connected forms (*Main*, *Dashboard*, *Entry*, *History*)
- Working Data Table persistence
- Visual trend graphs
- A clean UI with wellbeing colours

Although unfinished in parts, the app demonstrates conceptual success. More importantly, the process developed my computational thinking and reflective skills. I now approach technology with curiosity rather than intimidation.

In future UX or digital-storytelling projects, I'll integrate this reflective method intentionally , mapping systems, anticipating dependencies, and viewing design as both aesthetic and logical. I hope to rebuild Dear Acne using Flask or React for greater flexibility, but Anvil provided the perfect entry point to understand how backend logic powers user experience.

Through Moon's (2006) reflection cycle, Norman's (2013) feedback loops, and Kuniavsky's (2003) prototyping ideas, I've come to see debugging as design in action , each error revealing new understanding. Learning to code has taught me that success isn't defined by a finished product, but by the growth achieved along the way.

References

Deci, E. L., & Ryan, R. M. (1985). *Intrinsic motivation and self-determination in human behavior.* Springer.

Kuniavsky, M. (2003). *Observing the user experience.* Morgan Kaufmann.

Moon, J. A. (2006). *Learning journals: A handbook for reflective practice and professional development.* Routledge.

Norman, D. A. (2013). *The design of everyday things* (Rev. ed.). Basic Books.

University of Technology Sydney. (2025). *Code Conundrums: Debugging Exercises.* Canvas.

Artificial Intelligence: Chat GPT & Claude AI- Prompts shown in appendix.

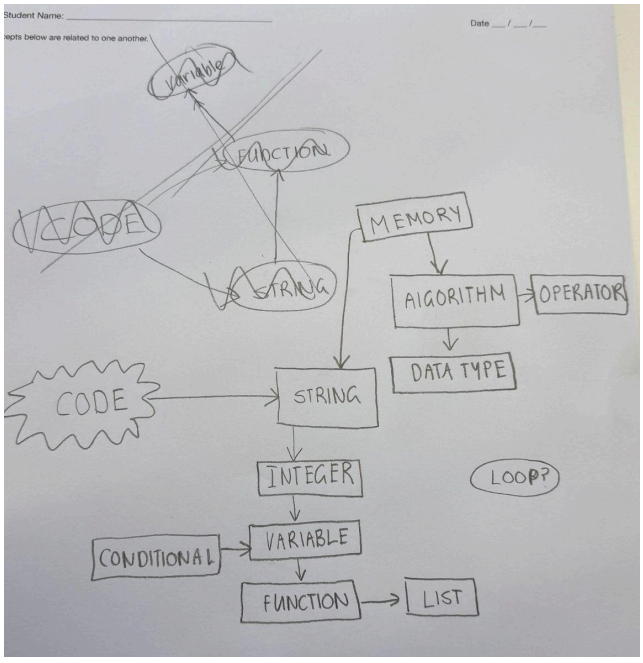
Appendix

[JOURNAL ENTRIES]

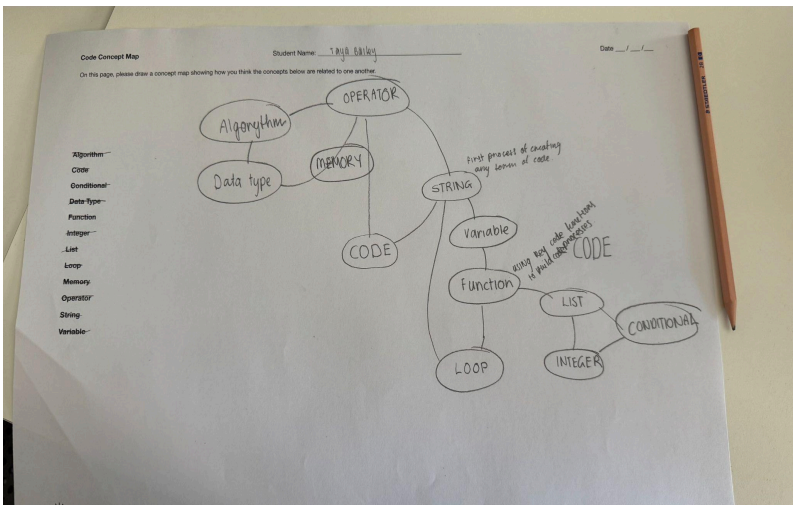
https://github.com/TayaBailey/DearAcneProtoypeFINAL/blob/65f4c1d24286abdb76a7b177aa59c58e2f72f404/Journal

[CODE CONCEPT MAP WEEK 1 VS WEEK 11]

WEEK 1



WEEK 11



Snippet 1: Form initialisation sequence (What?, p. 2):

```
def __init__(self, **properties):
    self.init_components(**properties)
    self.plot_skin.data = [ ]
    self.plot_mood.data = [ ]
```

Snippet 2 : Server data function (What?, p. 3):

```
@anvil.server.callable
def add_skin_entry(date, skin_condition, mood, triggers, notes):
    user = anvil.users.get_user()
    return app_tables.entries.add_row(
        user=user,
        date=date,
        skin_condition=skin_condition,
        mood=mood,
        triggers=triggers,
        notes=notes,
        created_at=datetime.now()
    )
```

Snippet 3 : Plot visualisation (What?, p. 4):

```
self.plot_skin.data = [{
    'x': data['dates'],
    'y': data['skin_scores'],
    'type': 'scatter',
    'mode': 'lines+markers',
    'line': {'color': '#8FBC8F', 'width': 3}
}]
```

Generative AI Declaration

I used ChatGPT (OpenAI, GPT-5) and Claude (v3 Opus) to support my learning of Anvil and Python while developing the Dear Acne prototype. Both tools acted as learning partners, helping me understand code structure, logic, and debugging principles rather than generating full code solutions.

Their use focused on:

- Explaining error messages and debugging logic
- Assisting with form connections and Anvil hierarchy setup
- Clarifying Python syntax, data-binding, and server/client interactions
- Providing comparative insights when troubleshooting persistent errors

All code was written, edited, and tested solely by me within the Anvil environment. ChatGPT and Claude were used to guide my understanding of concepts and problem-solving methods. Their responses were paraphrased, adapted, and implemented only after I verified the logic myself.

This aligns with UTS's Responsible Use of Generative AI Guidelines, ensuring AI assistance was applied for learning enhancement, not authorship. I take full responsibility for the final submitted work and confirm that all intellectual and creative decisions were made independently.

Below is a log of representative prompts and linked responses that informed my learning during the coding process.

Topic / Purpose	Prompt (Summary)	AI Tool Used	Response Link
Setting up code environment	"I'm a beginner coder using Anvil for the first time. Since Anvil is Python-based, should I set up server-side code or can everything run in the Main Form for now?"	ChatGPT	View Response

Learning resources	“Best learning resources for learning to code using Anvil web application.”	ChatGPT & Claude (for comparison)	View Response
UI logic & visibility	“Have I set up my true and false built-ins correctly?” + code for update_ui(self) function.	ChatGPT	View Response
Dropdown components	“Is this the correct way to connect dropdown options to a component?” (with list definition code).	ChatGPT & Claude	View Response
Import errors – form linking	“ModuleNotFoundError: No module named 'DashboardForm' appears is my form name wrong?”	ChatGPT	View Response
Plot components	“Can I use Anvil’s built-in plot components instead of importing Matplotlib?”	ChatGPT & Claude	View Response
Form-connection troubleshooting	“My forms still won’t connect where would you go from here as a programmer?”	ChatGPT & Claude (for strategy comparison)	View Response
Critical breakthrough - structural issue found	[Shared screenshot showing nested form hierarchy]Identified root cause: forms were nested inside each other (MainForm > EntryForm > DashboardForm > HistoryForm). Instructed to drag forms to same level as siblings.	Claude	“Your forms are nested inside each other in the hierarchy: This nested structure is causing the import issues. The forms need to be at the same level, not nested."