91906 & 91907 Complex Programming

Please add links to your work on this slide. Then remove this instruction. Note that most of the instructions have been supplied as speaker notes for later reference :) **Ensure your Github repositories, Trello board, and videos are set to public**.

Trello board / Project Management: <https://trello.com> **Make PUBLIC**

The project on Github: <https://github.com/> **Make PUBLIC**

Final Project Filename: <Currency_Converter_v2.py>

Final Testing Video: <https://drive.google.com/> **Make PUBLIC**

Project Management

Provide a brief description of the project management methodology you are using and explain why you chose to use it

I am using the management methodology named ‘agile’ this methodology consists of doing 1-4 week ‘sprints’ this is where you have 1-4 weeks to complete a series of tasks and log them. This keeps track of what you have accomplished and keeps you accountable.

Feedback for component trialling

https://docs.google.com/forms/d/e/1FAIpQLSfCID89dHTm9_C28On8zsxwisGqvG0EEy2CQ2A445sgyVLpmw/viewform?usp=dialog

For feedback, I have created and sent out a google form to get an insight into feedback that my peers could give towards my program. This allows me to select the most suitable components and improve the usability of my program.

component trialling

During the development of my interface, I trialled two different approaches. Option 1 being C_01_Converter_Gui_V1.py, and option 2 being C_01_Converter_Gui_V2.py, in option one i hard-coded every button individually. This made the code long and repetitive. For Option two I stored the button details in a list and used a loop to generate them dynamically. I chose version two as it made the code much cleaner and saved the buttons into a list, making it easier to disable the history button at the start using its index.

component trialling

Variables: Centralized vs. Hardcoded Constants

I experimented with storing fixed configuration values, including calculating bounds and temperature restrictions. Option 1: Hardcoding the boundary numbers right into my primary software files' validation logic. Option 2 (all_constants.py): Transferring all boundary limitations to a different constants file and importing it as needed. Option 2 is what I went with. Strong limits are introduced by storing constraints such as min_int_nzd and MAX_CALCS in an external all_constants.py file. Instead of looking through the main code, I simply need to update one file if any rules or boundary values need to be changed in the future.

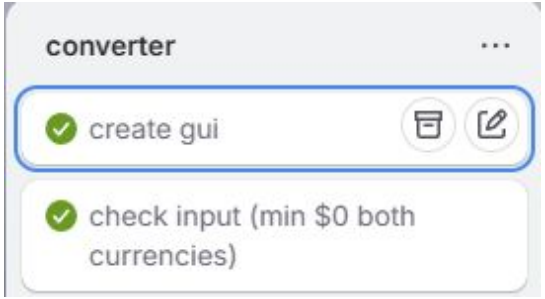
Project Management - Sprint Tracking

Add information about how your project is progressing each week based on your planning. Add a screenshot of the Trello board card entry

Task	Sprint Number	Start Date	End Date
Create base GUI	1	11/03/26	25/03/26

What are you working on?	What did you achieve?	What are the next steps?
Creating the gui and interactable buttons; To USD, To NZD, Help / Info, History / Export. Im working on making the GUI in a suitable layout and positioning.	Designed and created a suitable layout; Title, header, text box for an input, results, converter buttons.	Work on the other gui's and their functionality.

Evaluation - what worked well & what did not?
Switching up the positions of the gui worked well, it allowed me to understand where i wanted things.



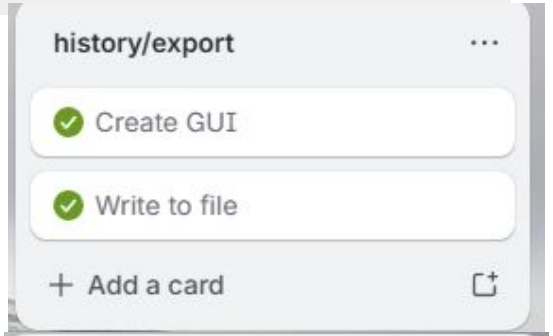
Project Management - Sprint Tracking

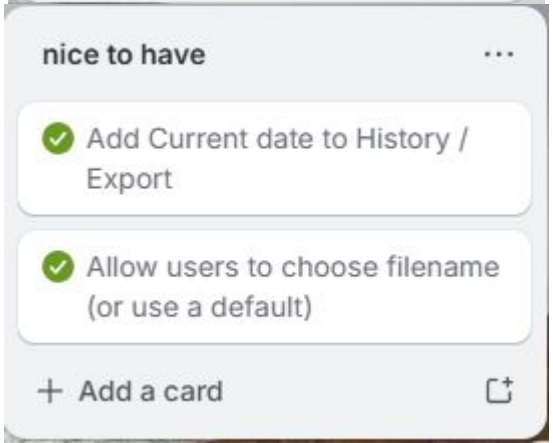
Add information about how your project is progressing each week based on your planning. Add a screenshot of the Trello board card entry

Task	Sprint Number	Start Date	End Date
Create history/export GUI	2	26/03/26	18/03/26

What are you working on?	What did you achieve?	What are the next steps?
I am working on creating a gui that logs the calculations done and is able to export them into a .txt file along with the date and the option to change the file name	I made the gui, using the other gui as a format and then worked around the template until i had what i wanted. I made it so you can export it as a .txt file while the default name being the 'currency_date.txt'.	Make the help gui and make it functional, a pop up that gives instructions.

Evaluation - what worked well & what did not?
Reusing my old template was quite helpful as i could replace old values.



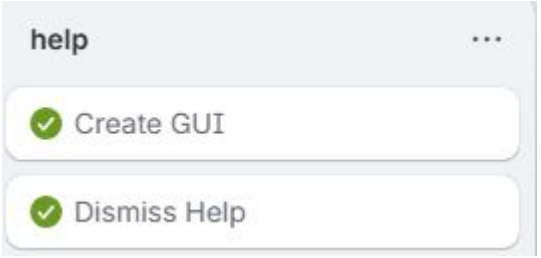


Project Management - Sprint Tracking

Task	Sprint Number	Start Date	End Date
Create help GUI and make it functional	3	19/03/26	25/03/26

What are you working on?	What did you achieve?	What are the next steps?
Creating a pop-up that informs the user on what to do, enter info then press a button to convert, and then gets dismissed after they press another button.	I made a popup button that informs the user on what they can do, and what they need to do to use the converter.	Make the converter, help button and export buttons functional. Create a working currency converter.

Evaluation - what worked well & what did not?
One thing that worked well was reusing the template for the layout, since this was just a popup it was pretty straightforward what it needed to do and creating it was simple.

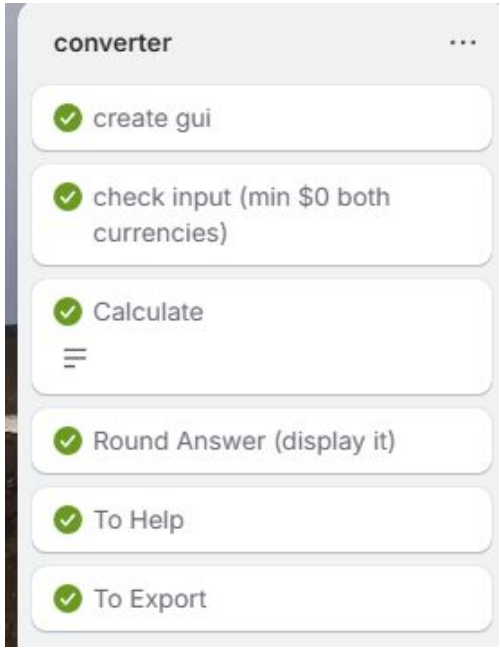


Project Management - Sprint Tracking

Task	Sprint Number	Start Date	End Date
Create conversion tool	4	26/03/26	1/04/26

What are you working on?	What did you achieve?	What are the next steps?
Creating the conversion tool, it can convert between usd and nzd, rounding up to the nearest dollar.	I created a conversion tool that goes between usd and nzd while rounding to the nearest dollar.	Incorporate these components into one main file.

Evaluation - what worked well & what did not?
I used my old code as a template and changed the values to make it accurate to the conversion rate, this worked well as the code i had previously done was well written.



Addressing Relevant Implications

Relevant implication: usability

Usability was improved in the code by using clear labels, simple instructions, and easy-to-read buttons. Error messages were added in the check_amount() function to tell users when they entered invalid input. The program also changes the entry box colour to red when there is an error, helping users notice mistakes. The help window (DisplayHelp) gives users instructions on how to use the program, while the history/export feature (Export History) lets users view and save previous calculations, making the program more convenient and user-friendly.

Addressing Relevant Implications

functionality

Functionality was achieved by creating functions that allow the program to correctly convert currencies and manage user interactions. The convert() function performs the NZD to USD and USD to NZD calculations using an exchange rate. The check_amount() function checks that the user input is valid before allowing the conversion. Additional functions such as to_help() and to_history() open extra windows for help information and calculation history. The export feature also adds functionality by saving the user’s calculations into a text file.

Addressing Relevant Implications

aesthetics

Aesthetics were achieved by using consistent colours, fonts, and layouts throughout the program. Different button colours were used to help users quickly identify each feature, such as green for converting to USD, purple for converting to NZD, and blue for history/export options. The program also uses bold Arial fonts and spacing between widgets to make the interface neat and easy to read. Background colours in the help and history windows were added to make the different sections visually clear and improve the overall appearance of the program.

Addressing Relevant Implications

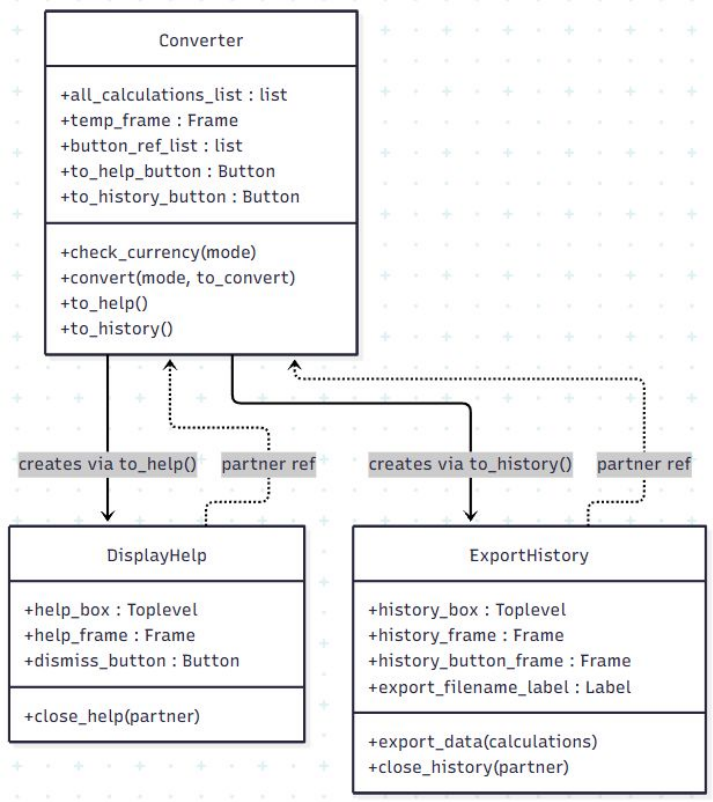
Accessibility

Accessibility was improved by making the interface simple and easy to understand for a wide range of users. Large fonts and clear labels help users read information more easily, while the wraplength setting ensures text stays organised instead of going off the screen. The program provides error feedback through both text messages and colour changes, helping users recognise mistakes quickly. Buttons are also clearly labelled with their purpose, such as “Help / Info” and “History / Export”, making navigation easier for users who may be unfamiliar with the program.

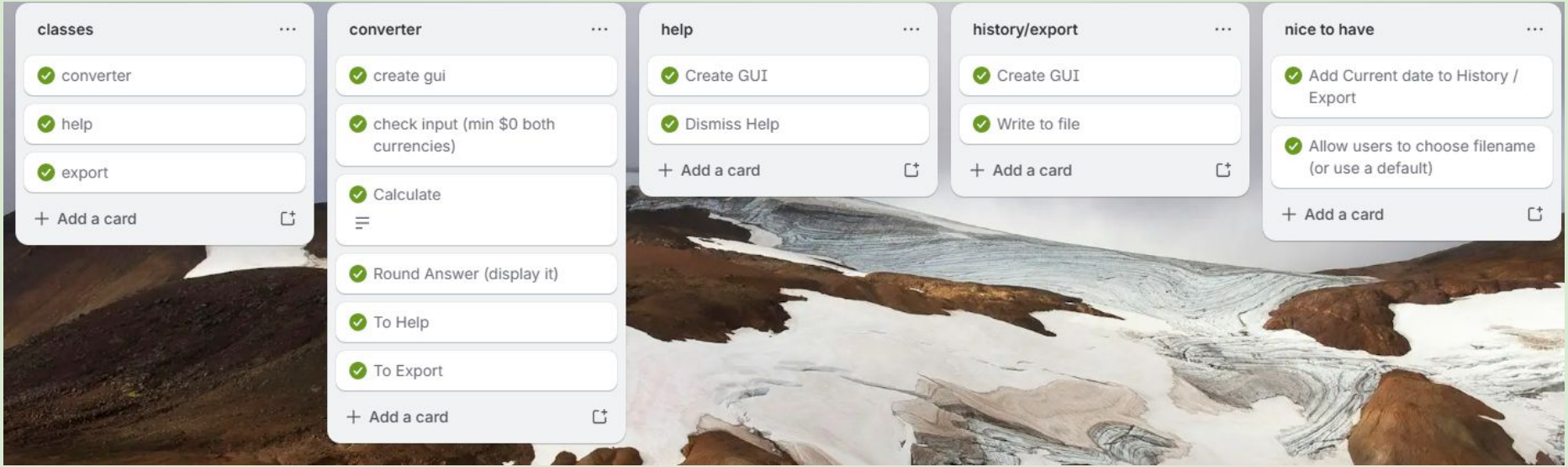
Graphical User Interface Design...

Create wireframes for your program's GUI. Please place them on this slide.

Program Structure



Project Decomposition



Develop your Components!!

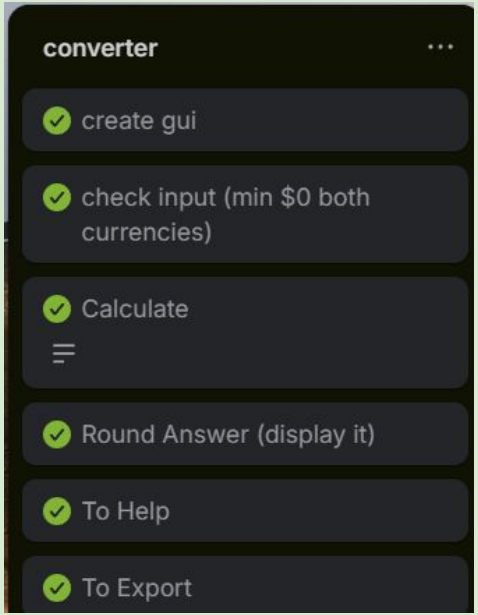
Make as many copies of the next three slides as you need. For each component you should have…

- A trello screenshot / evidence of ongoing use of your project management tools
- A test plan for the component that has been created BEFORE you start coding. Your plan should allow you to test all logical pathways for this component. It should also include test cases for relevant boundary and unexpected values.
- Screenshot evidence showing that you have worked through the test plan you developed.

You should also provide evidence of trialling multiple components & techniques. Please make slides as needed for that evidence.

Converter GUI

Duplicate this slide and use it to show your planning for the current component. This could be in the form of a trello screenshot.



Converter GUI

Create a test plan for this component BEFORE you start coding. Your plan should allow you to test all logical pathways for this component. It should also include test cases for relevant boundary and unexpected values.

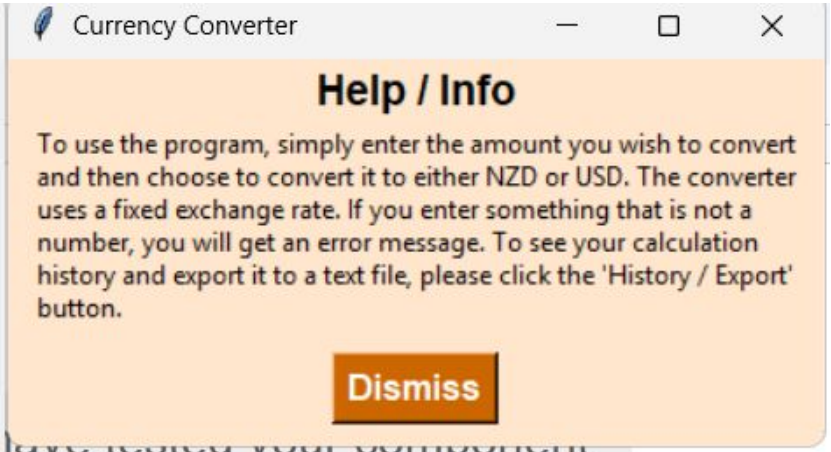
Data input	Expected output
Press “To NZD”	Enter a valid number
Press “To USD”	Enter a valid number
Press “Help / Info”	Pop-up appears, detailing the instructions

Converter GUI

Use this slide to provide evidence that you have tested your component in accordance with the test plan you developed earlier.

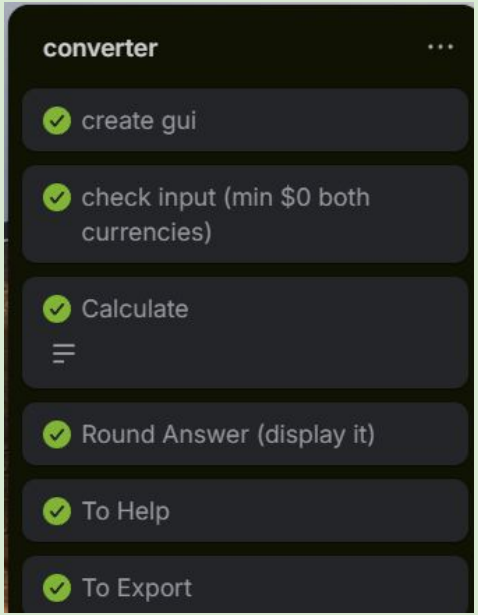
Converter GUI

Use this slide to provide evidence that you have tested your component in accordance with the test plan you developed earlier.



Converter calculations

Duplicate this slide and use it to show your planning for the current component. This could be in the form of a trello screenshot.



Converter calculations To NZD

Create a test plan for this component BEFORE you start coding. Your plan should allow you to test all logical pathways for this component. It should also include test cases for relevant boundary and unexpected values.

Data input	Expected output
xyz	Enter a valid number
0	$0.0 USD is $0.0 NZD
100	$100 USD is $166.67 NZD
5.5	$5.5 USD is $9.17 NZD

Converter Calculations To NZD

Use this slide to provide evidence that you have tested your component in accordance with the test plan you developed earlier.

```
['$0.0 USD is $0.0 NZD']
['$0.0 USD is $0.0 NZD', '$100.0 USD is $166.67 NZD']
['$0.0 USD is $0.0 NZD', '$100.0 USD is $166.67 NZD', '$5.5 USD is $9.17 NZD']
```

Converter calculations To USD

Create a test plan for this component BEFORE you start coding. Your plan should allow you to test all logical pathways for this component. It should also include test cases for relevant boundary and unexpected values.

Data input	Expected output
xyz	Enter a valid number
0	$0.0 NZD is $0.0 USD
100	$100 NZD is $60.0 USD
5.5	$5.5 NZD is $3.3 USD

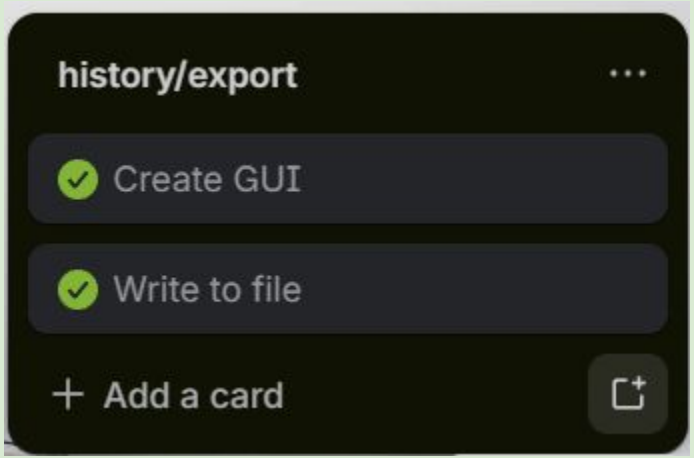
Converter Calculations To USD

Use this slide to provide evidence that you have tested your component in accordance with the test plan you developed earlier.

```
['$0.0 NZD is $0.0 USD']
['$0.0 NZD is $0.0 USD', '$100.0 NZD is $60.0 USD']
['$0.0 NZD is $0.0 USD', '$100.0 NZD is $60.0 USD', '$5.5 NZD is $3.3 USD']
```

History/Export

Duplicate this slide and use it to show your planning for the current component. This could be in the form of a trello screenshot.



History/Export

Create a test plan for this component BEFORE you start coding. Your plan should allow you to test all logical pathways for this component. It should also include test cases for relevant boundary and unexpected values.

Data input	Expected output
12 To NZD	$12.0 USD is $20.0 NZD
Press “History / Export”	History/Export pop-up appears
Press “Export”	Text says “Export Successful! The file is called currency_year_month_date”
Press “Close”	Closes the history/export popup

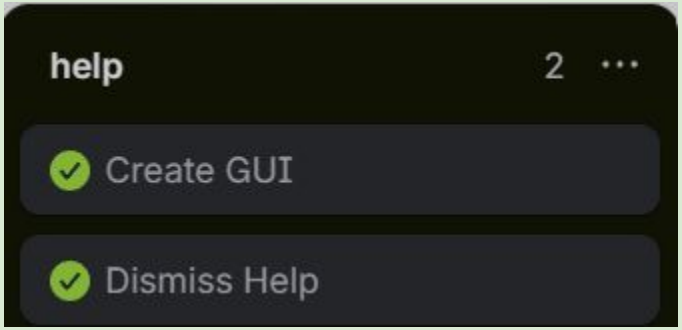
History/Export

Use this slide to provide evidence that you have tested your component in accordance with the test plan you developed earlier.

```
['$12.0 USD is $20.0 NZD']
```

Help/Info

Duplicate this slide and use it to show your planning for the current component. This could be in the form of a trello screenshot.



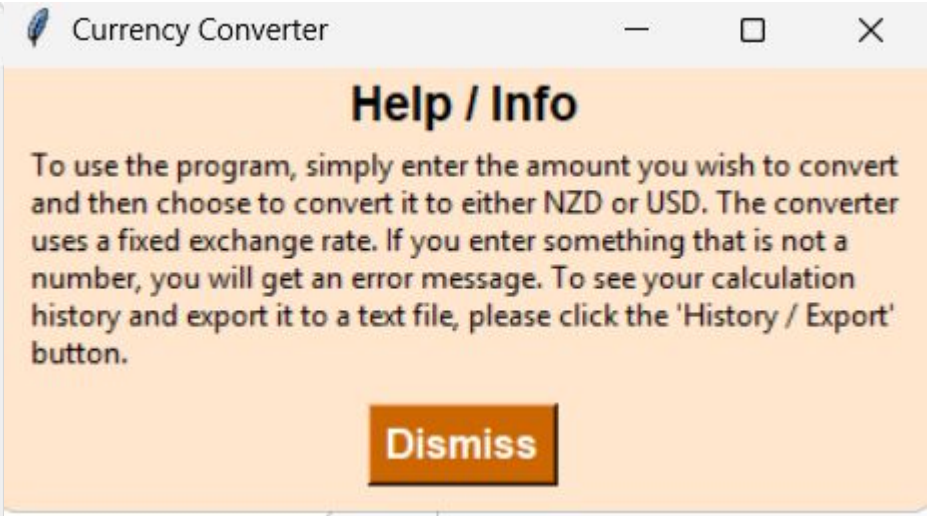
Help/Info

Create a test plan for this component BEFORE you start coding. Your plan should allow you to test all logical pathways for this component. It should also include test cases for relevant boundary and unexpected values.

Data input	Expected output
Press “help/info”	Help and information pop-up appears
Press “dismiss”	Pop-up disappears

Help/Info

Use this slide to provide evidence that you have tested your component in accordance with the test plan you developed earlier.



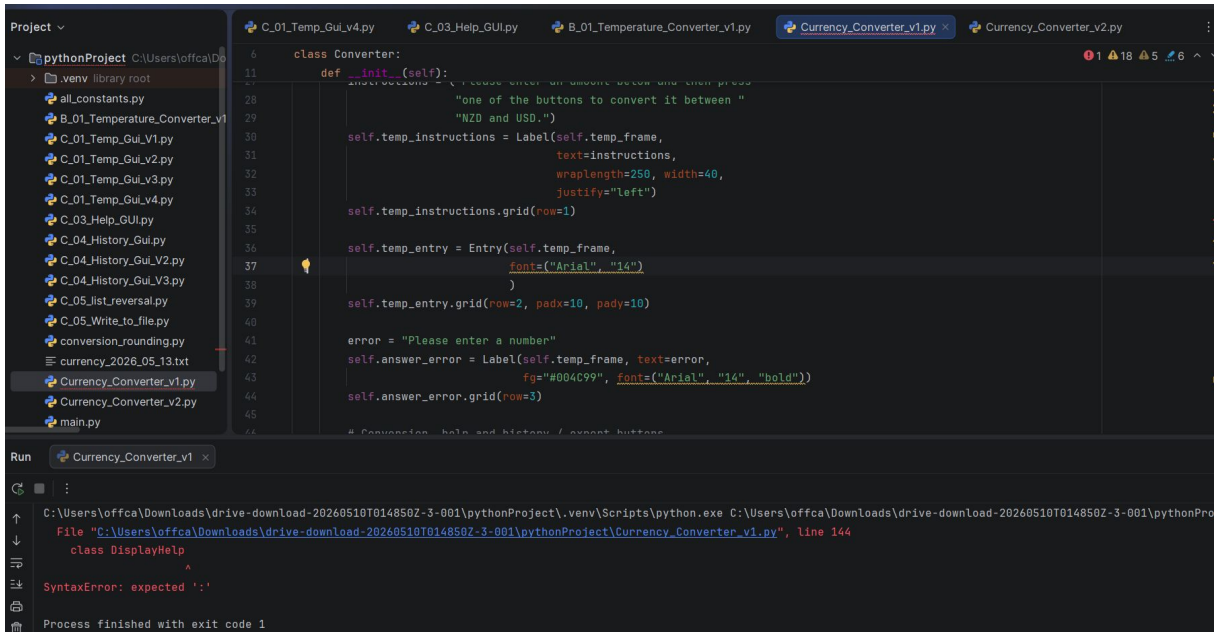
Component Trialling

Duplicate this slide to show you have trialled several ways users can input data .

History v1-v3 what is the difference,

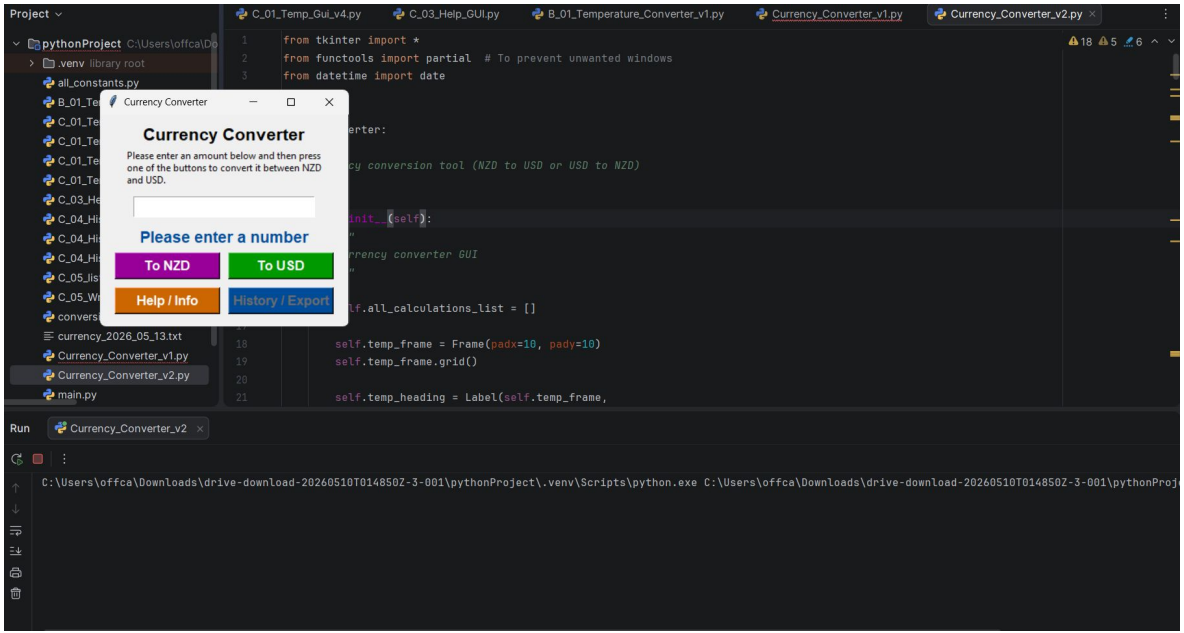
Debugging Evidence

I noticed my code wasn't working and it said that it had a syntax error, so i checked what it was and there was a semicolon missing.



Debugging Evidence

I noticed my code wasn’t working and it said that it had a syntax error, so i checked what it was and there was a semicolon missing.



Complete Program...

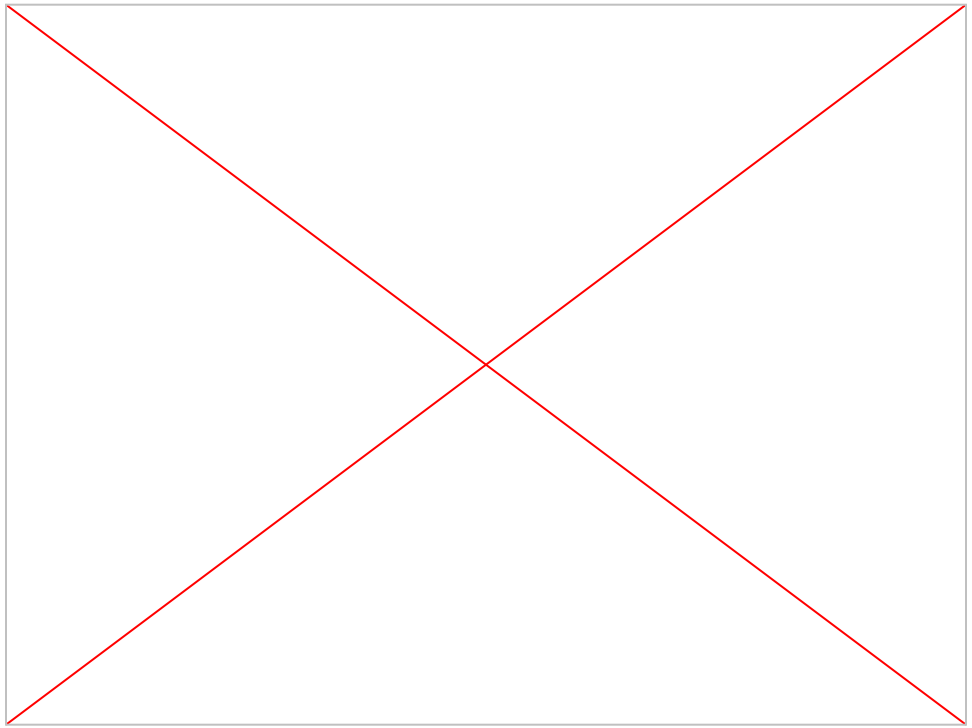
Assemble your components into a working program. On the slides that follow, please provide a test plan and evidence that your program works as expected.

If you are going for M / E, you also need to create extra slides showing how you have used your testing to improve the functionality of your program. This could mean having multiple test plans (and screenshots) showing several iterations of the assembled program.

Complete Program <test plan>

Data input	Expected output
Press “help/info”	Help and information pop-up appears
Press “dismiss”	Pop-up disappears
Data input	Expected output
Press “To NZD”	Enter a valid number
Press “To USD”	Enter a valid number
Data input	Expected output
xyz	Enter a valid number
0	$0.0 USD is $0.0 NZD

Complete Program <testing evidence>



Complex Processes - Discussion

- I used planning, testing, and trialling to improve the quality of my currency converter program. After testing the layout and getting feedback, I moved the help and history sections into separate pop-ups as it made the interface simpler. Testing also showed that the program could crash if users entered letters, blank values, or negative numbers, so I created the check_amount() function to stop invalid input. I also added a conversion history and export feature after trialling showed users wanted to save previous calculations, which made the program more useful and user-friendly.