Readme!

Please feel free to edit this template by adding / editing and removing slides. I've tried to cover most of what you might want to include but there are many ways of providing evidence so don't be afraid to delete slides that you don't need.

Slides with a green background should be used to show evidence of planning / project management. The slide with a purple background should be duplicated and used to show 'relevant implications' evidence.

The instructions in the grey boxes should be deleted once they have been read. Copies of the instructions are underneath the slides in the speaker notes (just in case they are needed at a later stage).

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 and Trello board are public**.

Trello board / Project Management:
https://trello.com/invite/b/xHIvpQnb/ATTI882ea659b518d107bf733b3e79d332a6B5491A3F/digital-storage-unit-convertor

The project on Github:
https://github.com/BingheLi/2024_Digital_Storage_Unit_Convertor

Final Project Filename: <ie: Completed_v3>

Addressing Relevant Implications

Use slides with this colour background to show that you have addressed the relevant implications. To do this easily, remove this instruction and then duplicate this slide.

Implications text goes here

Addressing Relevant Implications-Functionality

Functionality is crucial to my program because it ensures that the program operates as expected, with each component performing its intended tasks. Proper functionality is essential for providing a positive user experience, as any malfunction would prevent users from utilizing the program effectively. Each component should be able to handle unexpected or boundary inputs from users.To address this, I continuously tested each component to ensure it works as intended, and clearly display error message when results are unexpected, this ensure the user can continue using the program without the unexpected outcome breaking the program. I also addressed functionality by the implementation of the dropdown boxes to select unit. Users can easily select the units to convert from and to. This simplifies the selection process, as instead of having to type in all the units they want to convert, which could causes issues if units are entered incorrectly or in a format the program does not recognise. Having this dropdown menu allow user to easily and quickly to complete the conversion process. I have also addressed functionality with my history import/export component. The two components are two way communicated. Meaning that the exported file from history export component can be imported back with the history import component. This allow user to work with the conversion history in a easier way, as user don’t have to manually repeat all the previous conversion process.

Addressing Relevant Implications-Usability

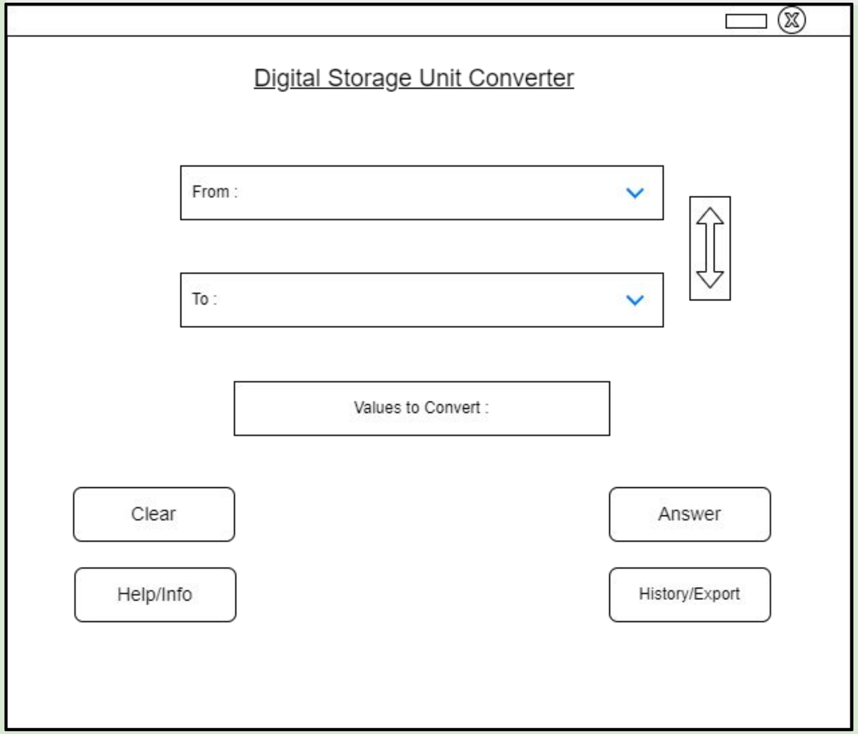
Another important aspect of my program is usability, which determines to how easily users can navigate and interact with the program. A clear and user-friendly interface is essential to guide users through the program and keep them engaged. Good usability enhances user-friendliness and makes the overall experience more enjoyable, as users won't have to spend time figuring out how to use the program, therefore avoiding confusion. I addressed this by using descriptive labels like ‘From:’ and ‘To:’ that can clearly communicate with the user and guide them through the conversion process. I also use large bold text with contrasting colours to its background in my program so that it can clearly communicate with the user, this allow users with vision problems such as colour-blindness or unclear vision to easily pick up on the content of my program, improving their user experience. I also addressed this by implementing clear error messages, for example if user try to convert with the empty entry grid or invalid value(non-numerical), a error message will show up and tell them what they did wrong and what to enter. I also addressed usability by implementing a instruction gui, where the user can click on the instruction button to bring up a new window with instruction explain the functionality of each button and how to use the program.

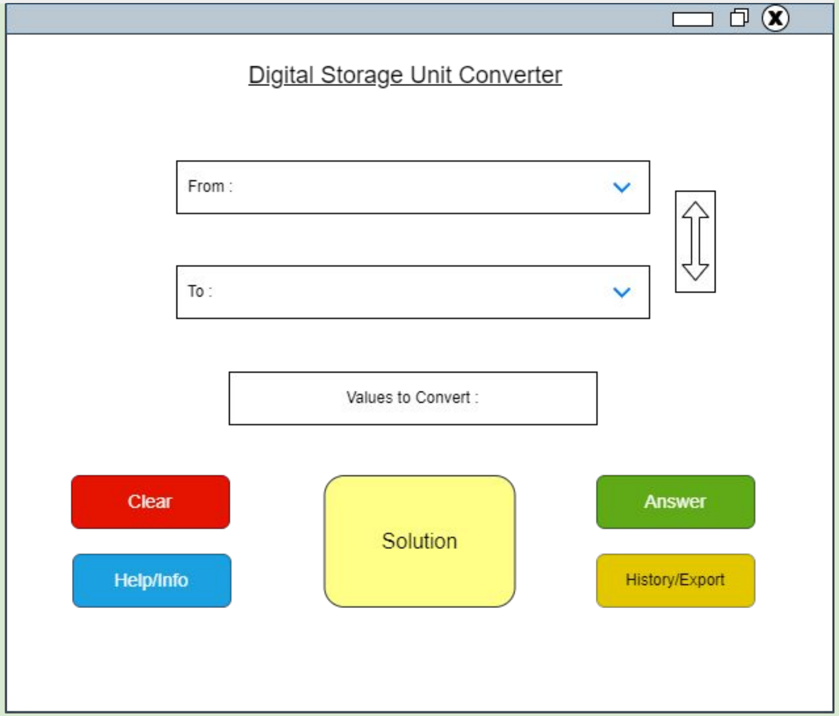
Addressing Relevant Implications-Aesthetics

Another important consideration is aesthetics, which is about the design and appearance of the program. This is crucial because it affects user experience and willingness to use the program. If the design is dull or uninspired, users may be less engaged, while an overly vibrant design can also be off-putting. I addressed this by maintaining a balance, keeping the program simple yet visually appealing. My approach to this is using high contrast colour theme but also not too overly colourful. I used Light cyan(#EAF6FF) and light black(#232528), the two colours are based of high contrasting white and black which suits the seriousness of a conversion tool, but both have a shade of blue in them, creating a contrasting harmony. This approach ensures the program isn't just filled with text, but has enough color and imagery to enhance the user experience. Layout of button are all centralled and symmetric, with buttons all aligned. This make the program more visually appealing and easier to navigate. I also use large sized bold text with contrasting colour to its background, this ensures the user can read the text properly and also makes all the components and conversion results stand out.

Graphical User Interface Design...

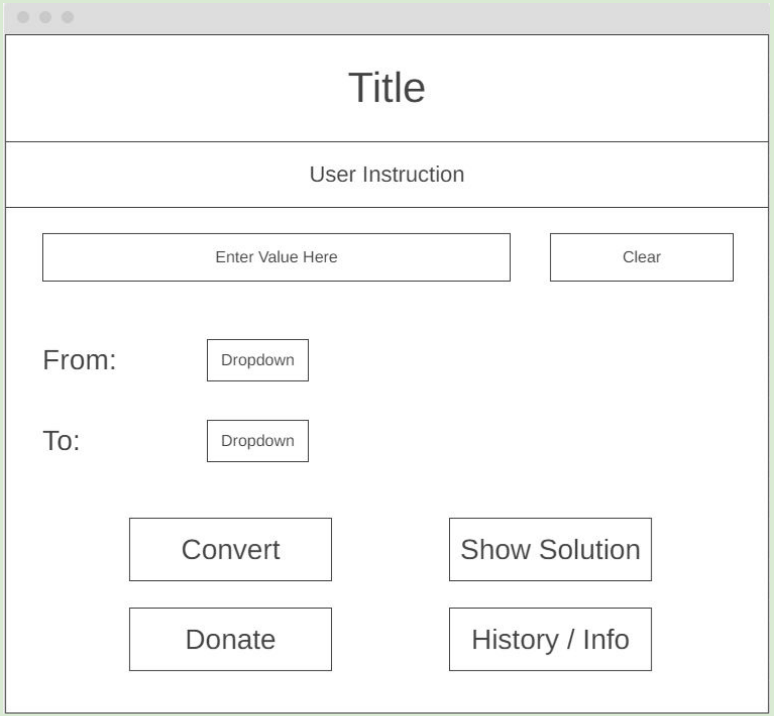
Main UI (scratch)





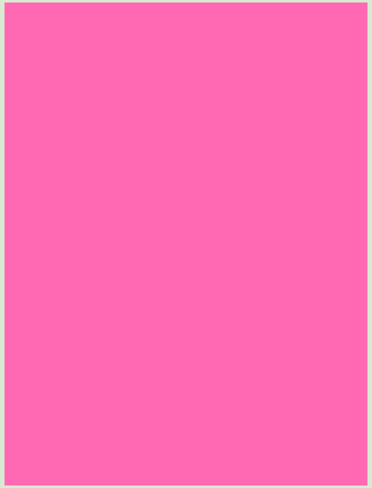
Graphical User Interface Design...

Main UI v2



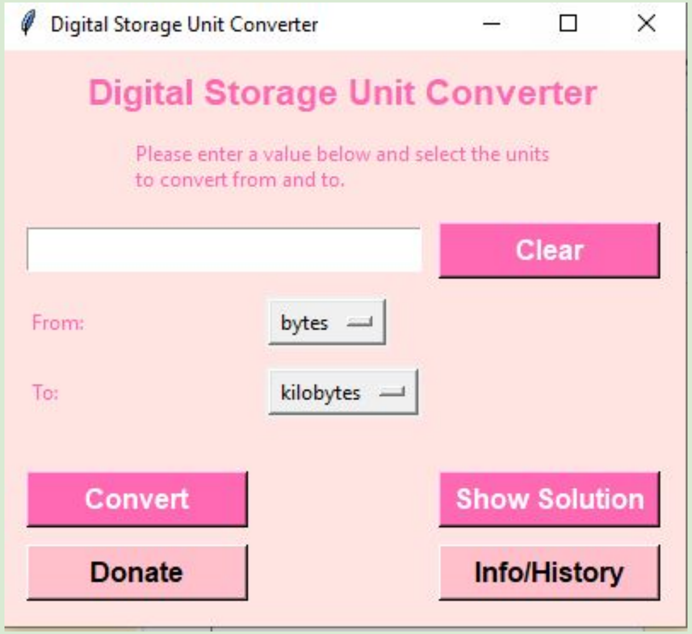
#FFE4E1

#FF69B4



Graphical User Interface Design...

Main UI v2 proof


Digital Storage Unit Converter

Digital Storage Unit Converter

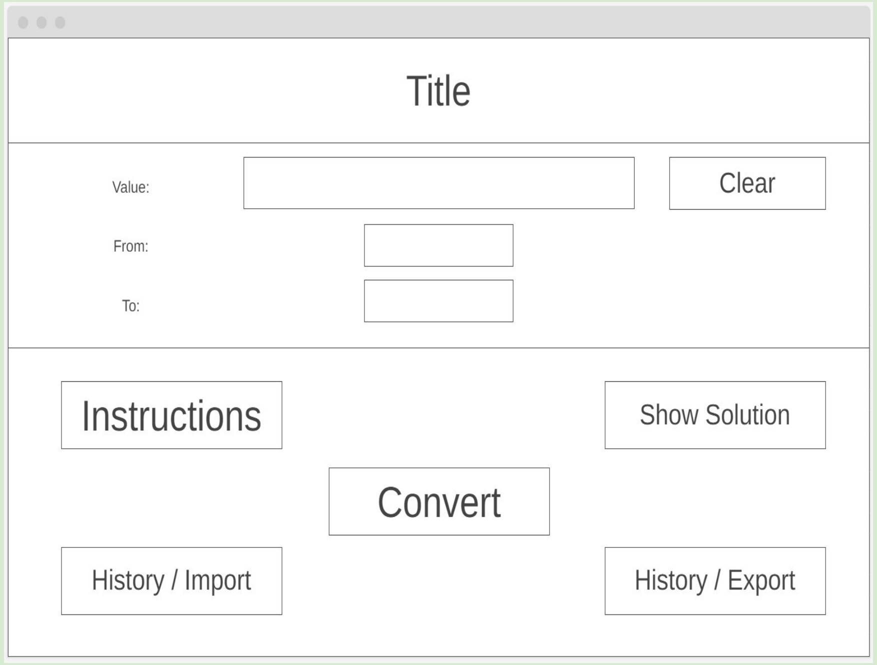
Please enter a value below and select the units
to convert from and to.

#FFE4E1 🟥

#FF69B4 🟥

Graphical User Interface Design...

Final user interface design

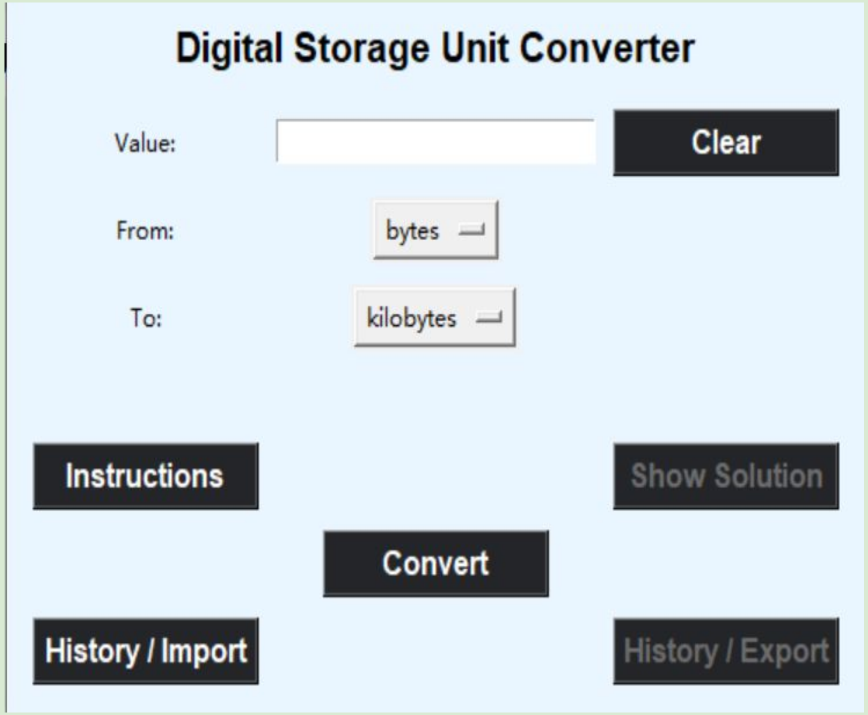


#eaf6ff

#232528

Graphical User Interface Design...

Final UI

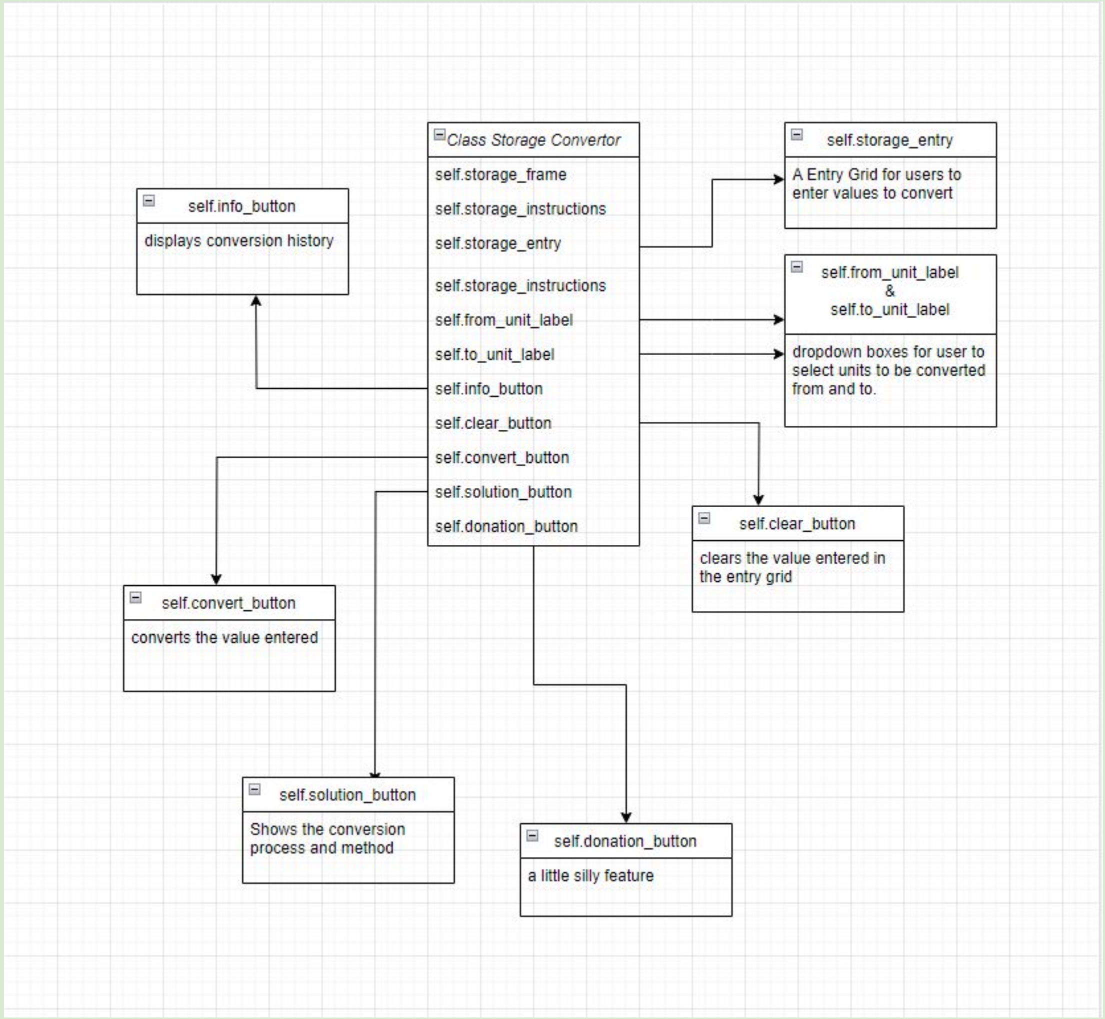


#eaf6ff

#232528

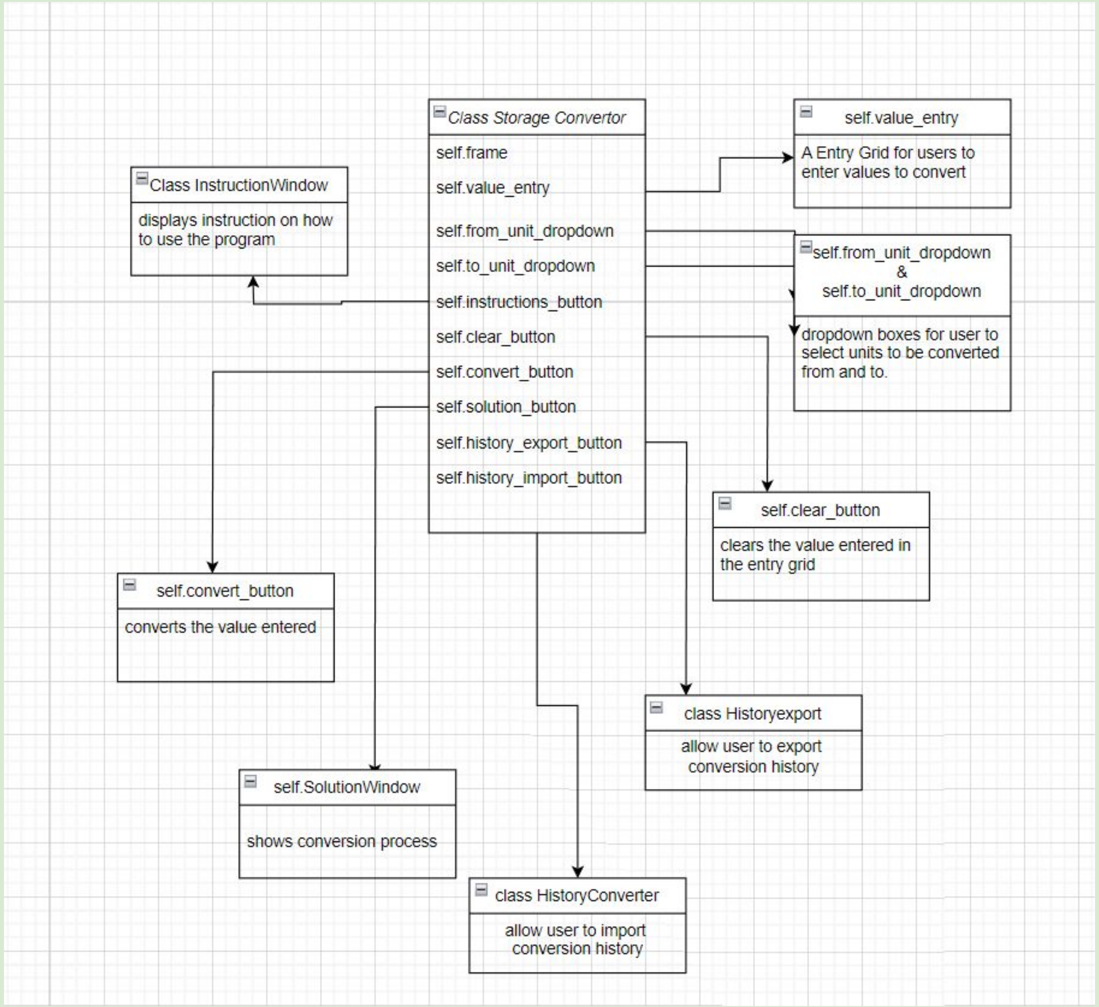
Program Structure

Program Structure Plan


Class Storage Convertor
self.storage_frame
self.storage_instructions
self.storage_entry
self.storage_instructions
self.from_unit_label
self.to_unit_label
self.info_button
self.clear_button
self.convert_button
self.solution_button
self.donation_button
self.storage_entry
A Entry Grid for users to enter values to convert
self.from_unit_label & self.to_unit_label
dropdown boxes for user to select units to be converted from and to.
self.info_button
displays conversion history
self.clear_button
clears the value entered in the entry grid
self.convert_button
converts the value entered
self.solution_button
Shows the conversion process and method
self.donation_button
a little silly feature

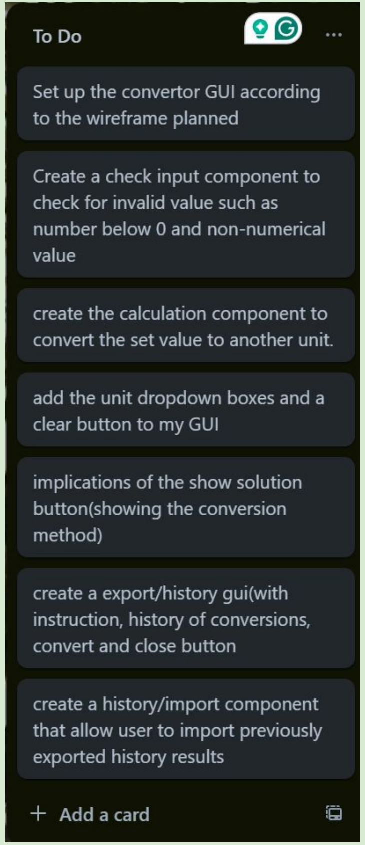
Program Structure

Program Structure - finished



Class Storage Convertor
self.frame
self.value_entry
self.from_unit_dropdown
self.to_unit_dropdown
self.instructions_button
self.clear_button
self.convert_button
self.solution_button
self.history_export_button
self.history_import_button
self.value_entry
A Entry Grid for users to enter values to convert
self.from_unit_dropdown & self.to_unit_dropdown
dropdown boxes for user to select units to be converted from and to.
Class InstructionWindow
displays instruction on how to use the program
self.clear_button
clears the value entered in the entry grid
self.convert_button
converts the value entered
class Historyexport
allow user to export conversion history
self.SolutionWindow
shows conversion process
class HistoryConverter
allow user to import conversion history

Problem Decomposition



To Do

Set up the convertor GUI according to the wireframe planned

Create a check input component to check for invalid value such as number below 0 and non-numerical value

create the calculation component to convert the set value to another unit.

add the unit dropdown boxes and a clear button to my GUI

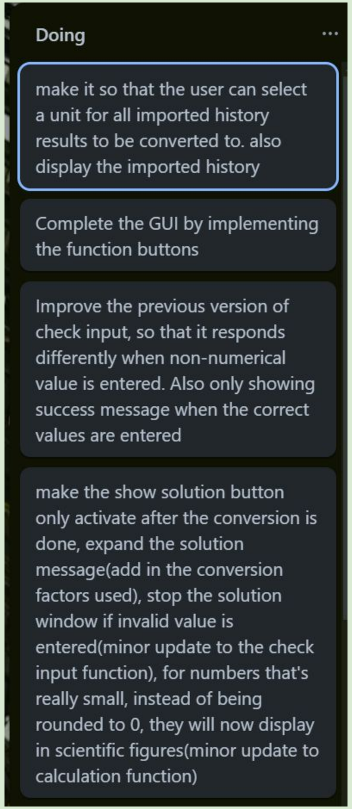
implications of the show solution button(showing the conversion method)

create a export/history gui(with instruction, history of conversions, convert and close button

create a history/import component that allow user to import previously exported history results

Initial decomposition of the program with all the planned components

Problem Decomposition



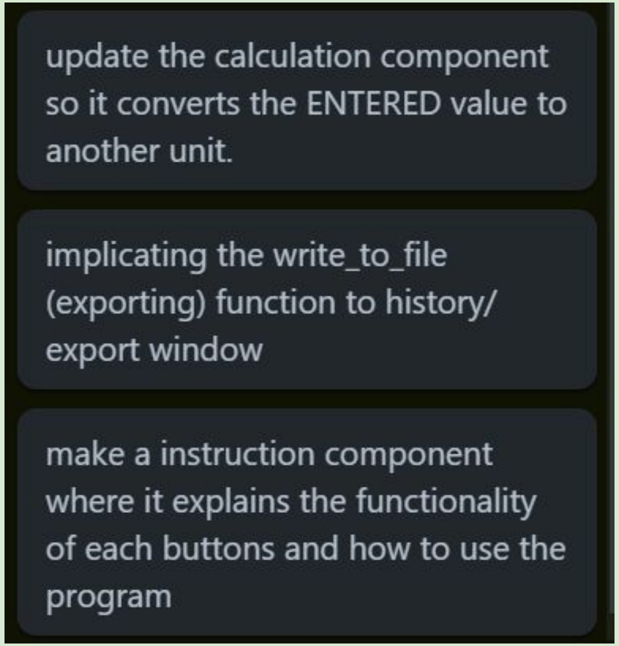
Doing

make it so that the user can select a unit for all imported history results to be converted to. also display the imported history

Complete the GUI by implementing the function buttons

Improve the previous version of check input, so that it responds differently when non-numerical value is entered. Also only showing success message when the correct values are entered

make the show solution button only activate after the conversion is done, expand the solution message(add in the conversion factors used), stop the solution window if invalid value is entered(minor update to the check input function), for numbers that's really small, instead of being rounded to 0, they will now display in scientific figures(minor update to calculation function)



update the calculation component so it converts the ENTERED value to another unit.

implicating the write_to_file (exporting) function to history/ export window

make a instruction component where it explains the functionality of each buttons and how to use the program

Improving and finishing up all my component to improve usability and functionality

Develop your Components!!

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

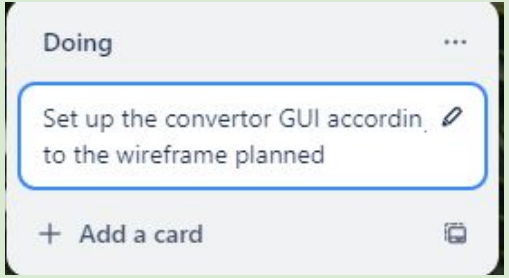
- A trello screenshot / evidence of ongoing use of your project management tools
- A test plan for the component that has been created BEFORE you start coding. Your plan should allow you to test all logical pathways for this component. It should also include test cases for relevant boundary and unexpected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

Component Planning...

Convertor GUI v1

- Purpose: To provide a basic interface for users to enter values and convert units.
- Design Choices:
  - Simple entry grid for user input.
- Techniques Trialed:
  - Basic text input and output functionality.
  - Outcome: The basic functionality was achieved but lacked user guidance



Doing

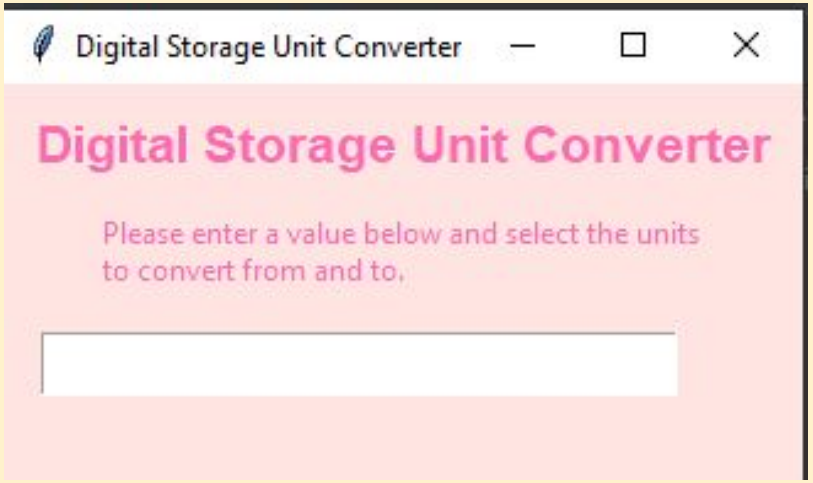
Set up the convertor GUI accordin
to the wireframe planned

Component Test Plan <Edit Me>

Test Case	Expected Outcome
Title of the GUI	Display on the top of the GUI
Instruction	Display below the Title
Entry Grid	A Entry Grid for user to enter a value

Component Trialling

The trailing results meets my expected outcome and programs runs according to plan.


Digital Storage Unit Converter

Digital Storage Unit Converter

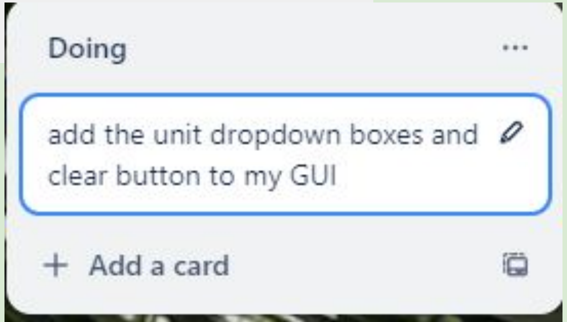
Please enter a value below and select the units to convert from and to.

Component Planning...

Convertor GUI v2

The initial version of the Convertor GUI featured a simple entry grid for user input. While it successfully handled basic text input and output, it was clear that additional user guidance were needed to improve functionality and user experience.

- Purpose: To enhance the user interface by adding dropdown menus for unit selection.
- Design Choices:
  - Added dropdown menus for selecting units to convert from and to.
- Techniques Trialed:
  - Enhanced user interface with dropdown menus.
  - Outcome: Improved user experience by simplifying unit selection


Doing

add the unit dropdown boxes and clear button to my GUI

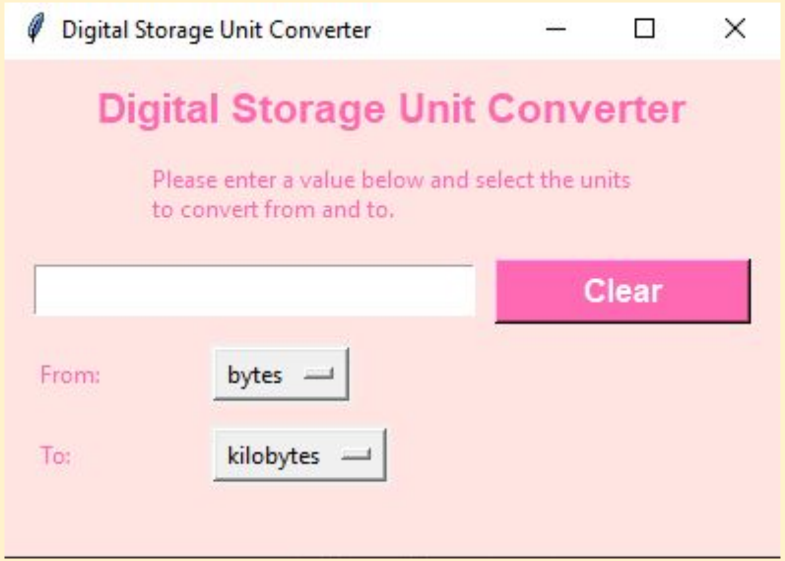
Component Test Plan <Edit Me>

Test Case	Expected Outcome
clear button	A clickable clear button display next to the entry grid
From Unit dropdown box	dropdown box that shows the units when clicked
To unit dropdown box	dropdown box that shows the units when clicked

The second version introduced dropdown menus for unit selection, simplifying the process for users and reducing input errors.

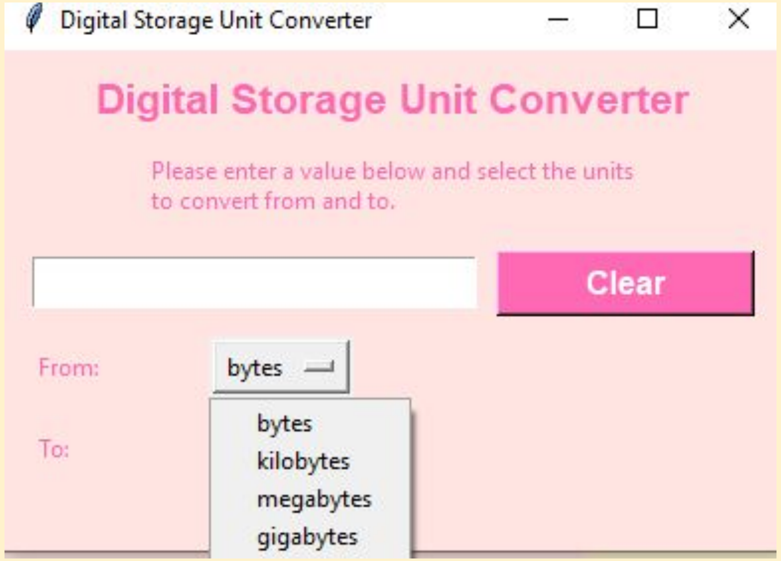
Component Trialling

The trailing results meets my expected outcome and programs runs according to plan.


Digital Storage Unit Converter

Digital Storage Unit Converter

Please enter a value below and select the units
to convert from and to.


Digital Storage Unit Converter

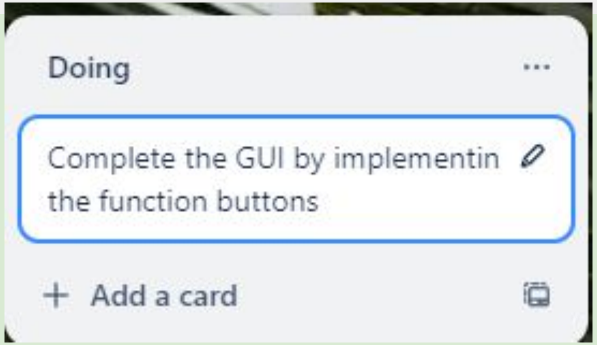
Digital Storage Unit Converter

Please enter a value below and select the units
to convert from and to.

Component Planning...

Convertor GUI v3

- Purpose: To implement functional buttons
- Design Choices:
  - Enhanced layout and user guidance features.
- Techniques Trialed:
  - Outcome: This version provided full funcrtional button, significantly improving usability.


Doing

Complete the GUI by implementin the function buttons

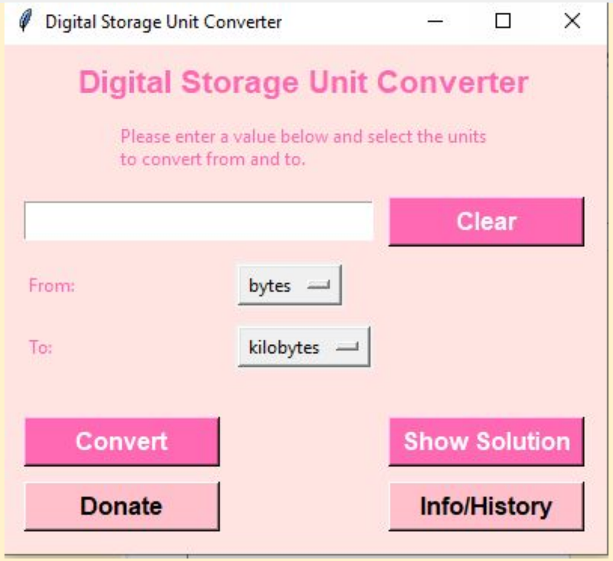
Component Test Plan <Edit Me>

Test Case	Expected Outcome
Convert	A clickable button
Show Solution	A clickable button
Donate	A clickable button
Info/History	A clickable button

In the third version, functional buttons are added to improve the overall usability and robustness of the interface. The layout was also enhanced for better user guidance.

Component Trialling

The trailing results meets my expected outcome and programs runs according to plan.


Digital Storage Unit Converter

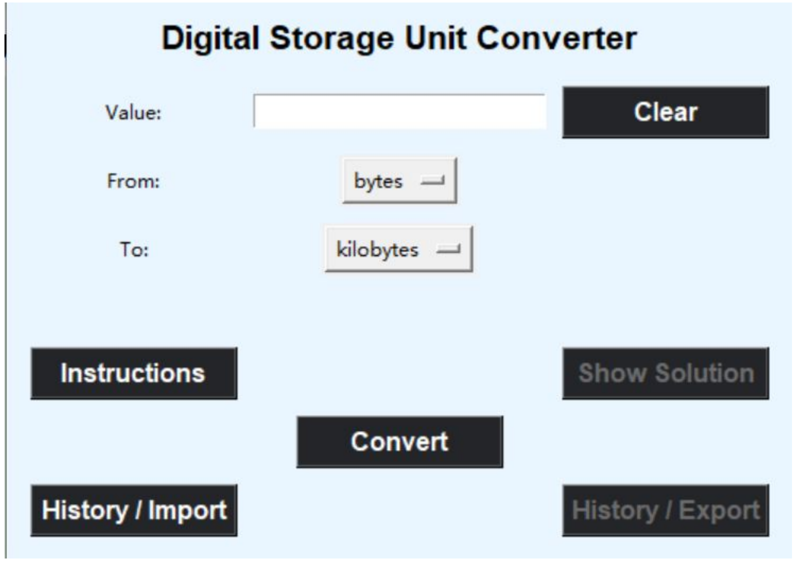
Digital Storage Unit Converter

Please enter a value below and select the units to convert from and to.

From: bytes

To: kilobytes

Final Converter GUI



For my final complete program, I made some minor changes to improve the Aesthetics and Usability of the program.

Aesthetics: layout of the program is now more centralled, colour theme will no longer be overly colourful pinky colour.

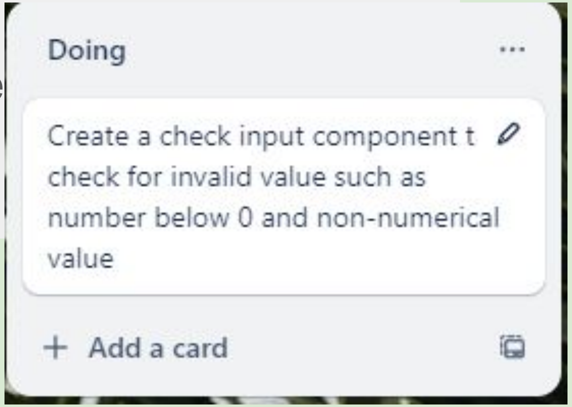
Usability: got rid of the brief instruction and replaced it with a separated and more detailed instruction, can be accessed by clicking the instruction button

Component Planning...

Check Input V1

Purpose: To validate user input before performing calculations.
Design Choices:
Basic input validation for numerical values.

Techniques Trialed:
Initial implementation of input validation.
Outcome: Successfully validated basic inputs but ne
comprehensive validation.


Doing

Create a check input component t check for invalid value such as number below 0 and non-numerical value

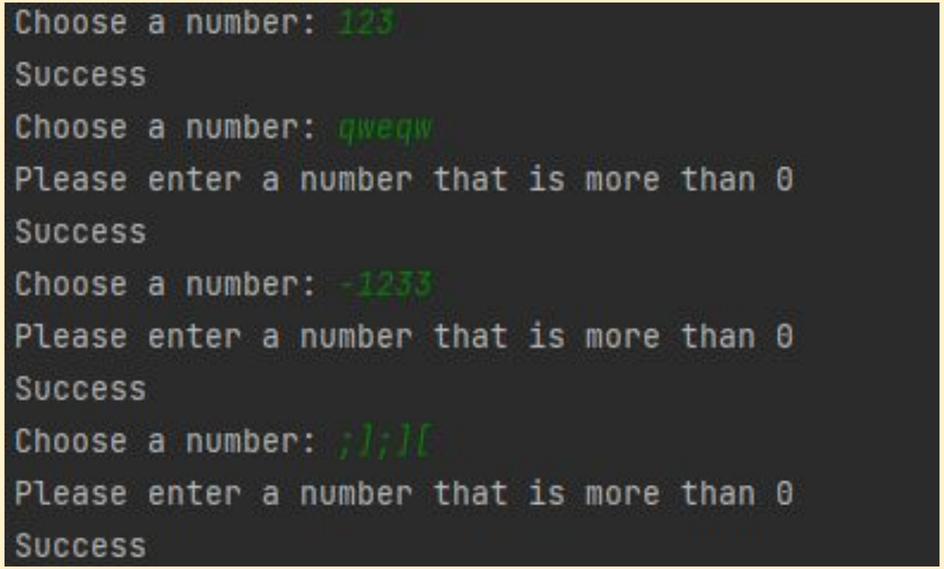
Component Test Plan <Edit Me>

Test Case	Expected Outcome
123	Success
qweqw	Please enter a number that is more than 0
-1233	Please enter a number that is more than 0
;];][	Please enter a number that is more than 0

The initial version of the Check Input component validated basic numerical inputs but required more comprehensive validation to handle a wider range of input scenarios.

Component Trialling

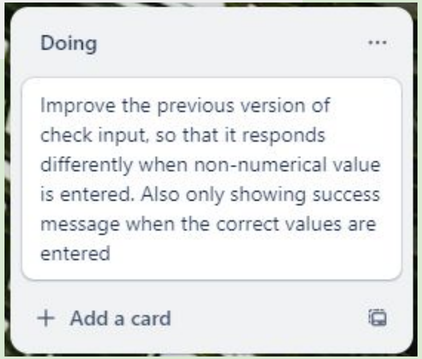
The trailing results are slightly different to my expected outcome, I wasn’t expecting the ‘success message to show after every test case. But it will be fixed in the next version.



```
Choose a number: 123
Success
Choose a number: qweqw
Please enter a number that is more than 0
Success
Choose a number: -1233
Please enter a number that is more than 0
Success
Choose a number: ;];][
Please enter a number that is more than 0
Success
```

Component Planning...

Check Input V2
Purpose: To enhance input validation and handle a wider range of input scenarios.
Design Choices:
Improved validation logic to handle non-numerical and boundary inputs.
Techniques Trialed:
Enhanced input validation.
Outcome: Provided more robust validation, improving the overall reliability of the program.



Improve the previous version of check input, so that it responds differently when non-numerical value is entered. Also only showing success message when the correct values are entered

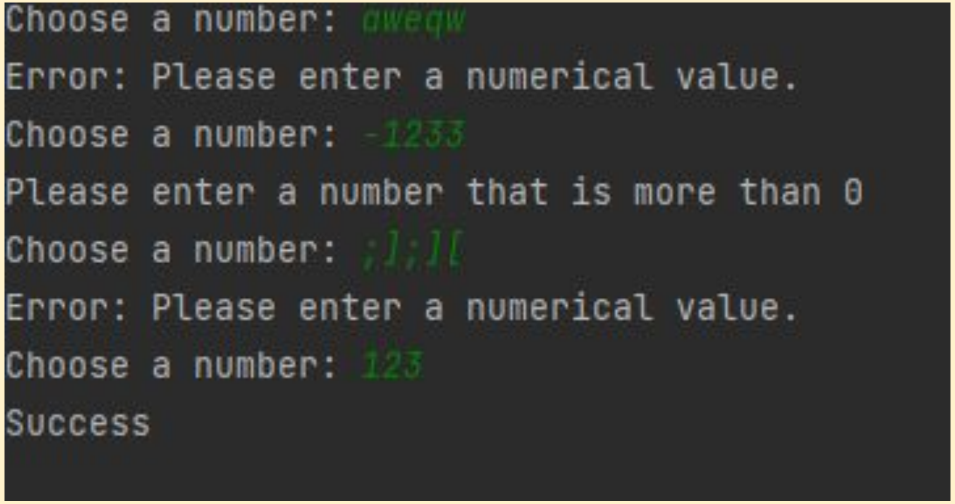
Component Test Plan <Edit Me>

Test Case	Expected Outcome
123	Success
qweqw	Error: Please enter a numerical value.
-1233	Please enter a number that is more than 0
;];][	Error: Please enter a numerical value.

The second version of the Check Input component enhanced the validation logic to handle non-numerical and boundary inputs, making the program more robust and reliable.

Component Trialling

The trailing results expected.



```
Choose a number: aweqw
Error: Please enter a numerical value.
Choose a number: -1233
Please enter a number that is more than 0
Choose a number: ;];][
Error: Please enter a numerical value.
Choose a number: 123
Success
```

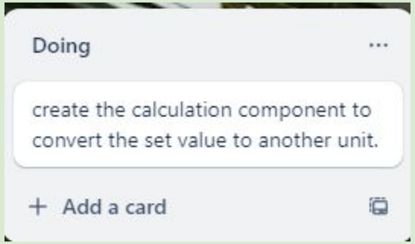
Check Input

The present check input component has been abandoned from the final finished program. It has been integrated into the calculation component.

Details on slide 41

Component Planning...

Calculation GUI v1
Purpose: To perform the actual conversion calculations.
Design Choices:
Basic calculation logic for converting units.
Techniques Trialed:
Initial implementation of calculation logic.
Outcome: Successfully performed conversions but needed refinement for edge cases.


Doing

create the calculation component to convert the set value to another unit.

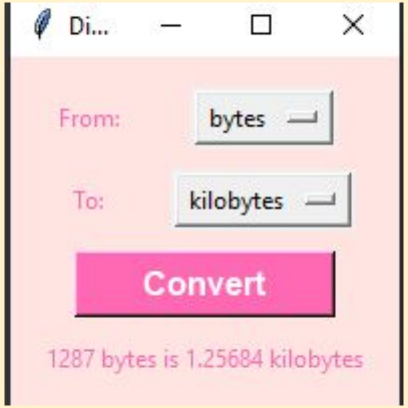
Component Test Plan <Edit Me>

Test Case	Expected Outcome
1287 bytes to kilobytes	1.25684 kilobytes
1287 bytes to megabytes	0.00123 megabytes
1287 bytes to Gigabytes	1.287E-6 Gigabytes

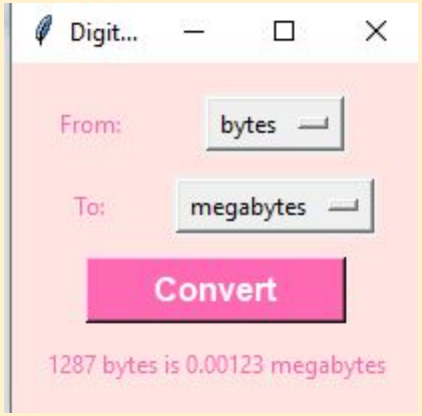
The initial version of the Calculation component successfully performed basic conversions but required refinement to handle edge cases and improve accuracy.

Component Trialling

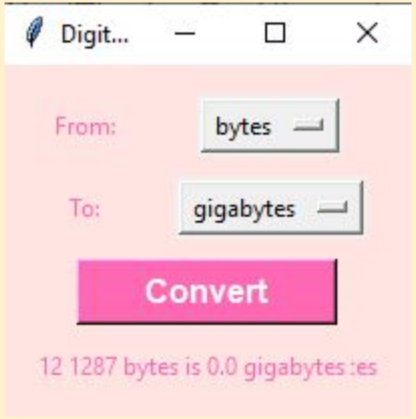
The trailing results are slightly different to my expected outcome, I wasn't expecting the '0.0 gigabytes' result, However it makes sense as converting from bytes to gigabyte is too much a difference. And my program is set to round to 5th decimal places. So it rounds the result to 0.



1287 bytes is 1.25684 kilobytes



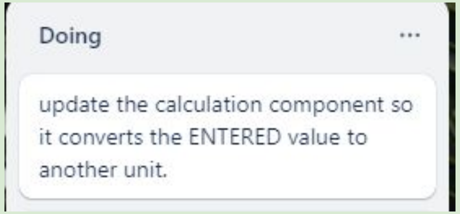
1287 bytes is 0.00123 megabytes



12 1287 bytes is 0.0 gigabytes :es

Component Planning...

Calculation GUI v2
Purpose: To refine the calculation logic allowing user to enter the value to convert and handle edge cases.
Design Choices:
Improved logic for handling edge cases and rounding errors.
Techniques Trialed:
Refined calculation logic.
Outcome: Provided more accurate and reliable conversion results.


Doing

update the calculation component so it converts the ENTERED value to another unit.

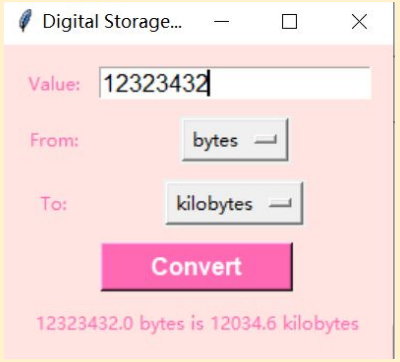
Component Test Plan <Edit Me>

Test Case	Expected Outcome
12323432 bytes to kilobytes	12034.6 kilobytes
5123345 bytes to megabytes	4.89 megabytes
74234325 bytes to Gigabytes	0.7 Gigabytes
1234 kilobytes to bytes	1263616 kilobytes

The second version of the Calculation component refined the logic to handle edge cases and rounding errors, providing more accurate and reliable conversion results. Also allowing user to enter the value to convert

Component Trialling

The trailing results are the same as my expected outcome.


Digital Storage...

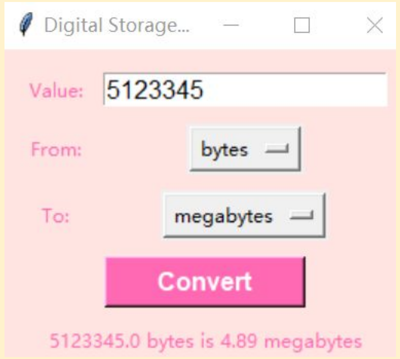
Value: 12323432

From: bytes

To: kilobytes

Convert

12323432.0 bytes is 12034.6 kilobytes


Digital Storage...

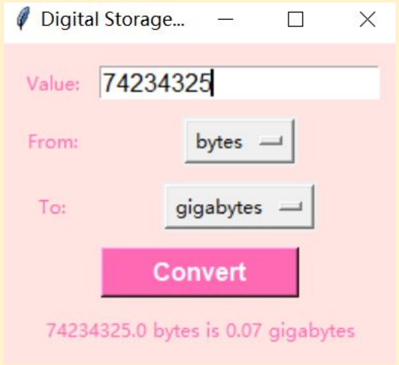
Value: 5123345

From: bytes

To: megabytes

Convert

5123345.0 bytes is 4.89 megabytes


Digital Storage...

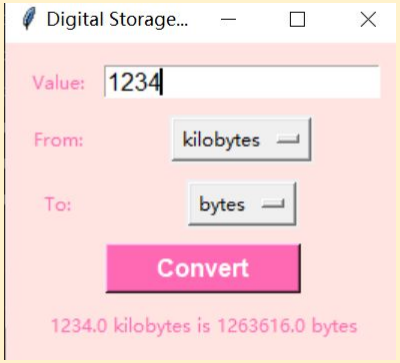
Value: 74234325

From: bytes

To: gigabytes

Convert

74234325.0 bytes is 0.07 gigabytes


Digital Storage...

Value: 1234

From: kilobytes

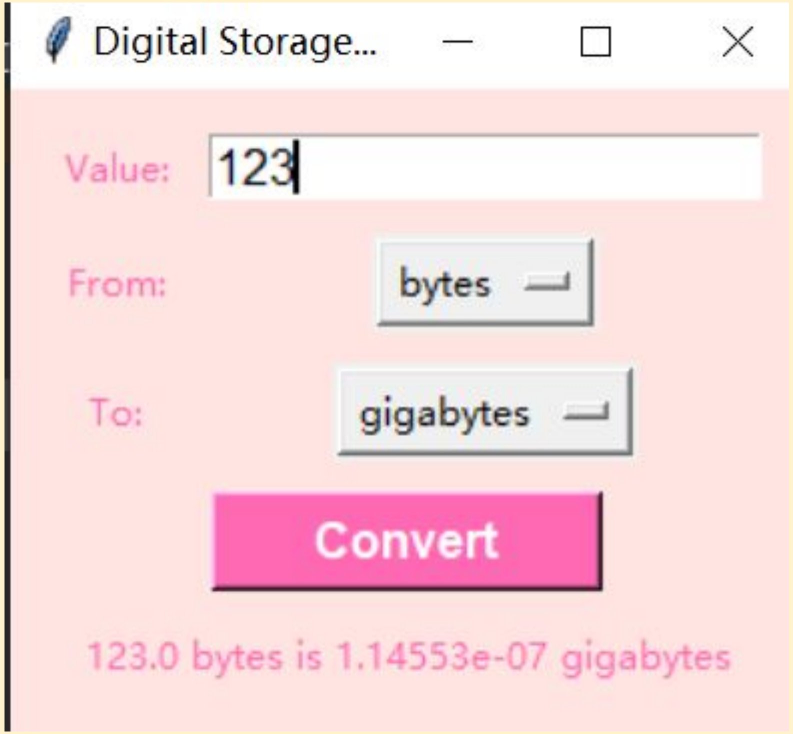
To: bytes

Convert

1234.0 kilobytes is 1263616.0 bytes

Extra minor update(from show solution function)

Values that are smaller than 5 decimal places will no longer be rounded to 0, instead they will display in scientific figures



Value: 123

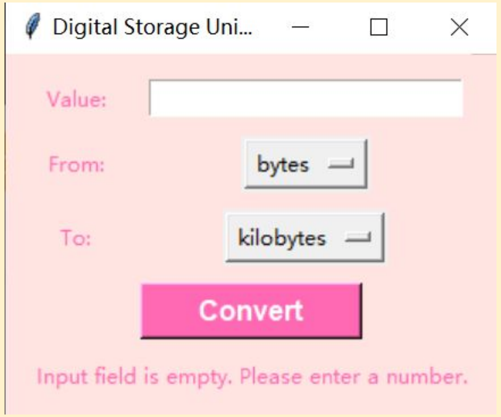
From: bytes

To: gigabytes

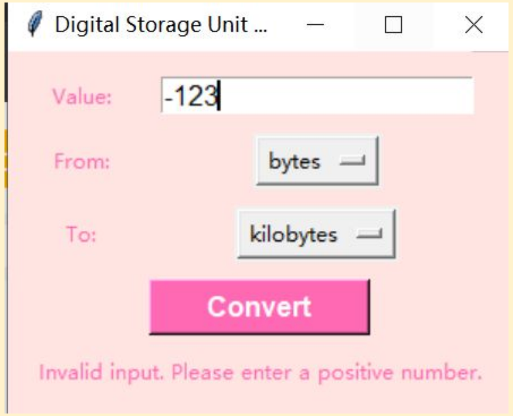
```
123.0 bytes is 1.14553e-07 gigabytes
```

Integration with check_input

Test Case	Expected Outcome
Empty entry grid	Input field is empty. Please enter a number.
Negative number	Invalid input. Please enter a positive number.
Non-numerical value	Invalid input. Please enter a valid number.

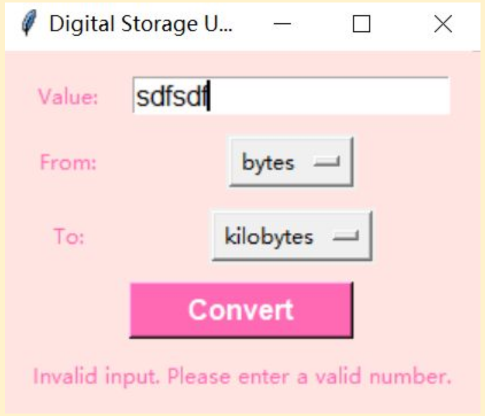


Input field is empty. Please enter a number.



-123

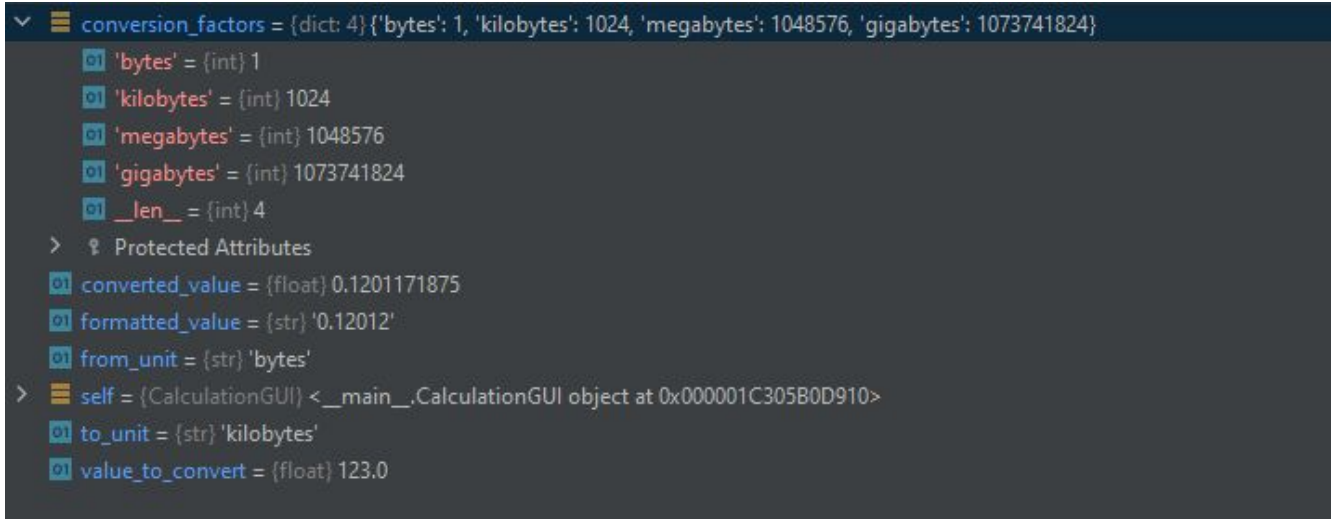
Invalid input. Please enter a positive number.



sdfsdf

Invalid input. Please enter a valid number.

Debugging of calculation GUI



```
conversion_factors = {dict: 4}{'bytes': 1, 'kilobytes': 1024, 'megabytes': 1048576, 'gigabytes': 1073741824}
    'bytes' = {int} 1
    'kilobytes' = {int} 1024
    'megabytes' = {int} 1048576
    'gigabytes' = {int} 1073741824
    __len__ = {int} 4
  > Protected Attributes
  converted_value = {float} 0.1201171875
  formatted_value = {str} '0.12012'
  from_unit = {str} 'bytes'
> self = {CalculationGUI} <__main__.CalculationGUI object at 0x000001C305B0D910>
  to_unit = {str} 'kilobytes'
  value_to_convert = {float} 123.0
```

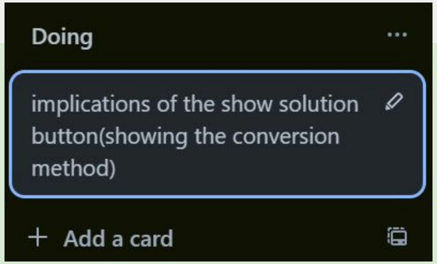
During the debugging process I can see how the program identify the units selected and use its relative conversion factor to convert the entered value

In this case:

value to convert=123
From unit=bytes
To units=kilobytes
Converted value=0.1201171875
Formatted value(results displayed)=0.12012

Component Planning...

Show Solution v1
Purpose: To display the steps and conversion factors used in the conversion process.
Design Choices:
Basic display of conversion factors and steps.
Techniques Trialed:
Initial implementation of the show solution feature.
Outcome: Successfully displayed the conversion factors but needed better formatting and user guidance.


Doing

implications of the show solution button(showing the conversion method)

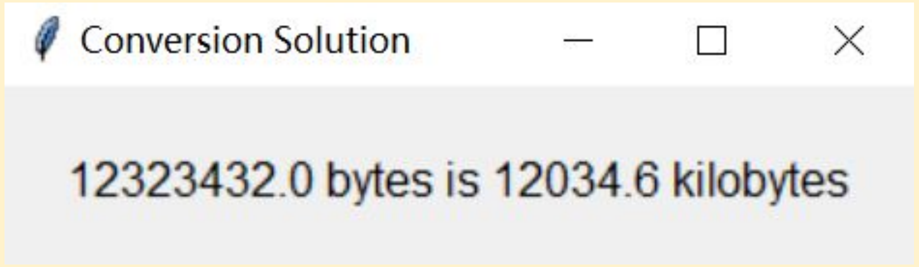
Component Test Plan <Edit Me>

Test Case	Expected Outcome
12323432 bytes to kilobytes	12034.6 kilobytes
5123345 bytes to megabytes	4.89 megabytes

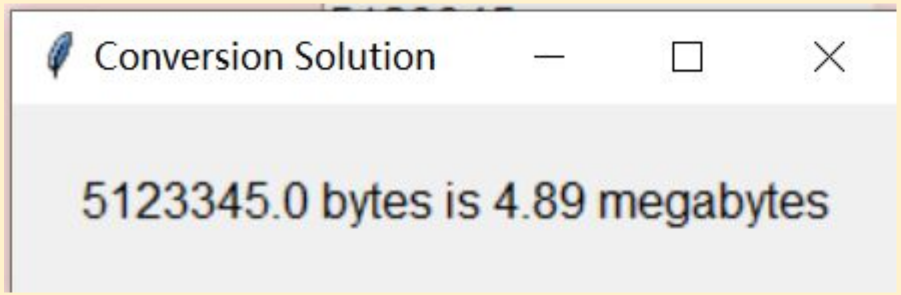
The initial version of the Show Solution component successfully displayed the conversion factors but required better formatting and user guidance to enhance usability.

Component Trialling

The trailing results are the same as my expected outcome.


Conversion Solution

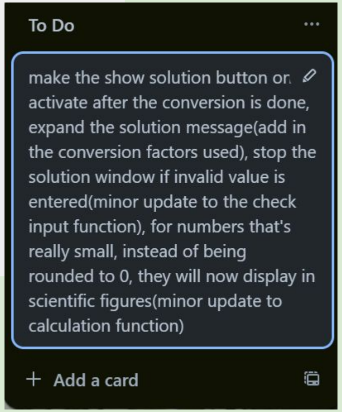
```
12323432.0 bytes is 12034.6 kilobytes
```


Conversion Solution

```
5123345.0 bytes is 4.89 megabytes
```

Component Planning...

Show Solution v2
Huge update woohoo!
Purpose: To improve the display format, flexible state of functional button and add user guidance.
Design Choices:
Enhanced formatting and added instructional text.
Techniques Trialed:
Improved display and user guidance.
Outcome: Provided a clearer and more user-friendly display of the conversion steps and factors.



make the show solution button on activate after the conversion is done, expand the solution message(add in the conversion factors used), stop the solution window if invalid value is entered(minor update to the check input function), for numbers that's really small, instead of being rounded to 0, they will now display in scientific figures(minor update to calculation function)

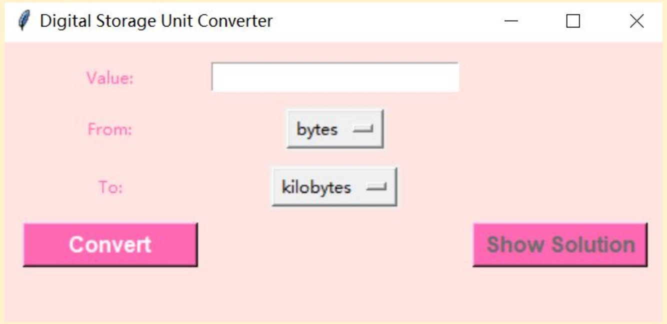
Component Test Plan <Edit Me>

The second version improved the Show Solution component by enhancing the formatting and adding instructional text, solution now only enable after conversion.making the information clearer and more user-friendly and the the functional of solution button more logical.

Test Case	Expected Outcome
Before conversion	Show solution button disabled
After conversion	Show solution button activated
Entry grid empty	Display error message, disable show solution button
Converted value smaller than 5 decimal places	Display in scientific figures
Show solution window	Show conversion factors used

Component Trialling

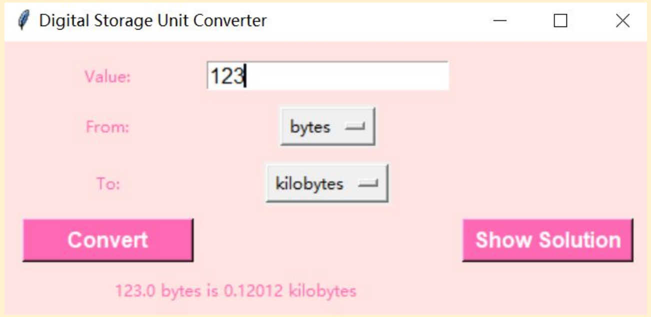
The trailing results are the same as my expected outcome.


Digital Storage Unit Converter

Value:

From: bytes

To: kilobytes

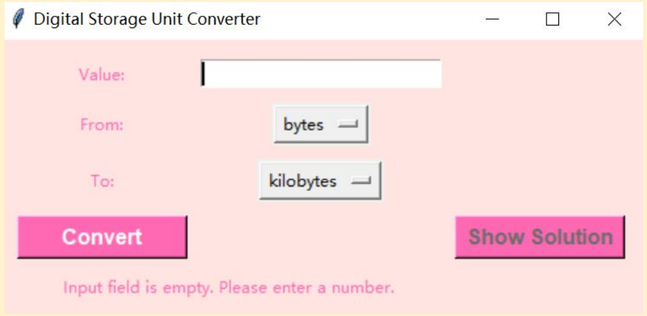

Digital Storage Unit Converter

Value: 123

From: bytes

To: kilobytes

123.0 bytes is 0.12012 kilobytes

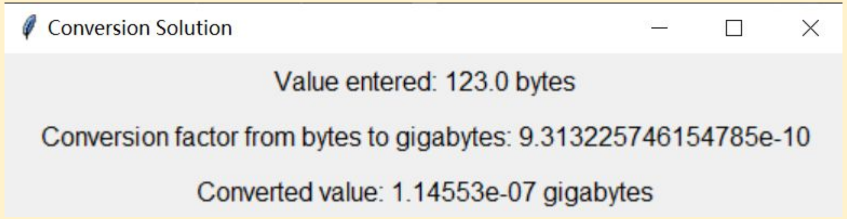

Digital Storage Unit Converter

Value:

From: bytes

To: kilobytes

Input field is empty. Please enter a number.

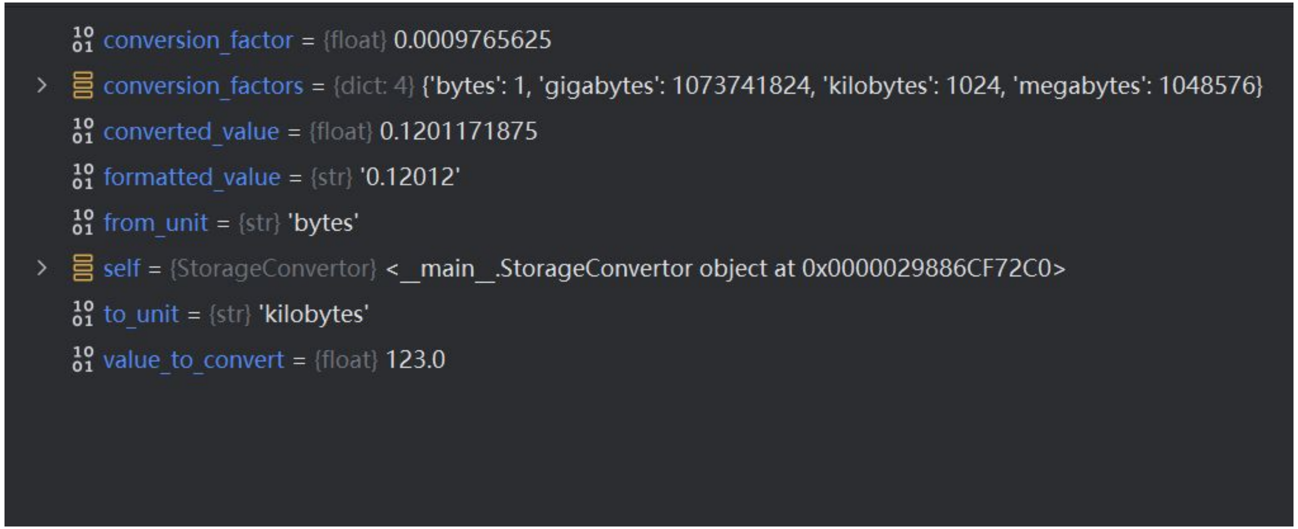

Conversion Solution

Value entered: 123.0 bytes

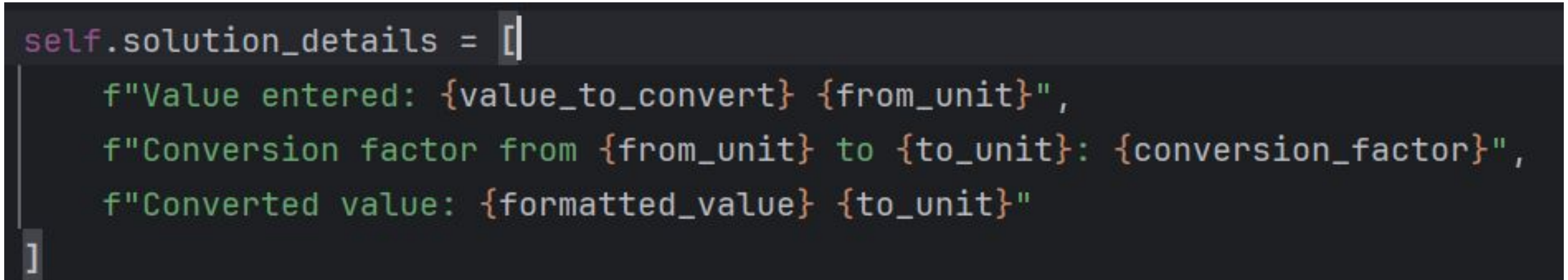
Conversion factor from bytes to gigabytes: 9.313225746154785e-10

Converted value: 1.14553e-07 gigabytes

Debugging of show solution gui



```
10
01 conversion_factor = {float} 0.0009765625
> conversion_factors = {dict: 4} {'bytes': 1, 'gigabytes': 1073741824, 'kilobytes': 1024, 'megabytes': 1048576}
10
01 converted_value = {float} 0.1201171875
10
01 formatted_value = {str} '0.12012'
10
01 from_unit = {str} 'bytes'
> self = {StorageConvertor} <__main__.StorageConvertor object at 0x0000029886CF72C0>
10
01 to_unit = {str} 'kilobytes'
10
01 value_to_convert = {float} 123.0
```



```
self.solution_details = [
    f"Value entered: {value_to_convert} {from_unit}",
    f"Conversion factor from {from_unit} to {to_unit}: {conversion_factor}",
    f"Converted value: {formatted_value} {to_unit}"
]
```

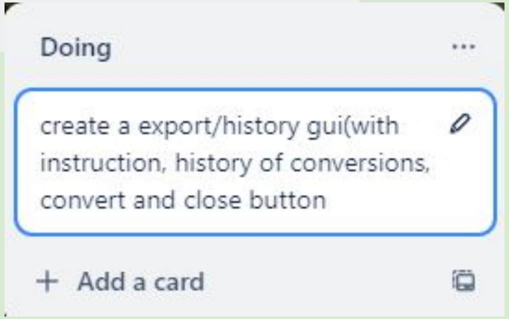
During the debugging process I can see how the program identify the variable that is going to display on the solution window

In this case:

value to convert=123
From unit=bytes
To units=kilobytes
Formatted value(results displayed)=0.12012
Conversion_factor = 0.0009765625

Component Planning...

History/ Export_v1
Purpose: To provide basic functionality for exporting conversion history.
Design Choices:
Simple export functionality without the write to file function yet.
Basic history export.
Outcome: Functionality worked



create a export/history gui(with instruction, history of conversions, convert and close button

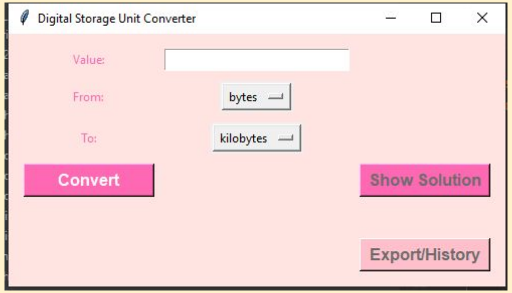
Component Test Plan <Edit Me>

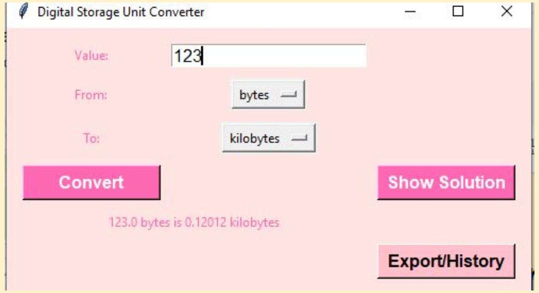
The initial version of the history export component consist of basic button function without the write to file function.

Test Case	Expected Outcome
Before conversion	History/ export button disabled
After conversion	History/ export button activated Conversion history displayed in history export window
Entry grid empty and click convert button (history/export window)	Show error message “file name cannot be empty”
Dismiss button	Closes the history export window
Entry grid filled and click convert button(history/export window)	Show success message “history exported successfully”

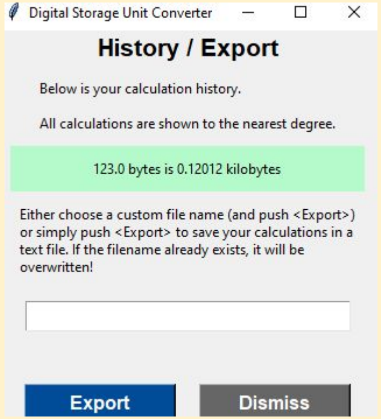
Component Trialling

The trailing results are the same as my expected outcome.





123.0 bytes is 0.12012 kilobytes



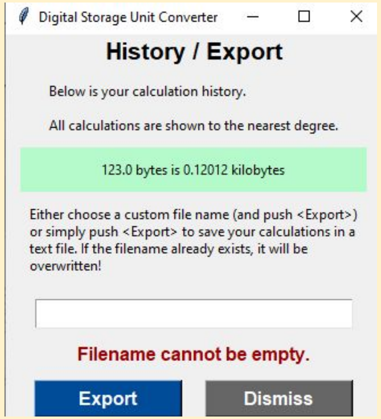
History / Export

Below is your calculation history.

All calculations are shown to the nearest degree.

123.0 bytes is 0.12012 kilobytes

Either choose a custom file name (and push <Export>) or simply push <Export> to save your calculations in a text file. If the filename already exists, it will be overwritten!



History / Export

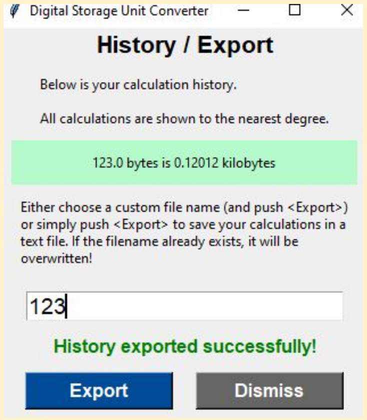
Below is your calculation history.

All calculations are shown to the nearest degree.

123.0 bytes is 0.12012 kilobytes

Either choose a custom file name (and push <Export>) or simply push <Export> to save your calculations in a text file. If the filename already exists, it will be overwritten!

Filename cannot be empty.



History / Export

Below is your calculation history.

All calculations are shown to the nearest degree.

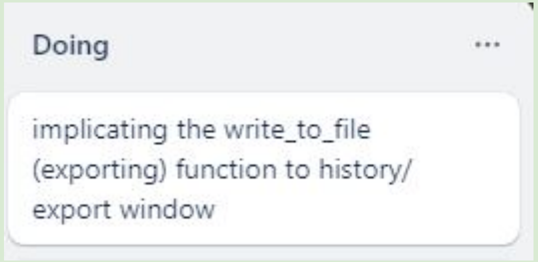
123.0 bytes is 0.12012 kilobytes

Either choose a custom file name (and push <Export>) or simply push <Export> to save your calculations in a text file. If the filename already exists, it will be overwritten!

History exported successfully!

Component Planning...

History/ Export_v2
Purpose: To enhance the import functionality and include write to file function.
Design Choices:
Complete the function of the component
Techniques Trialed:
Enhanced import functionality with better user guidance.
Outcome: Improved usability and functionality by implicatingh the write to file function


Doing

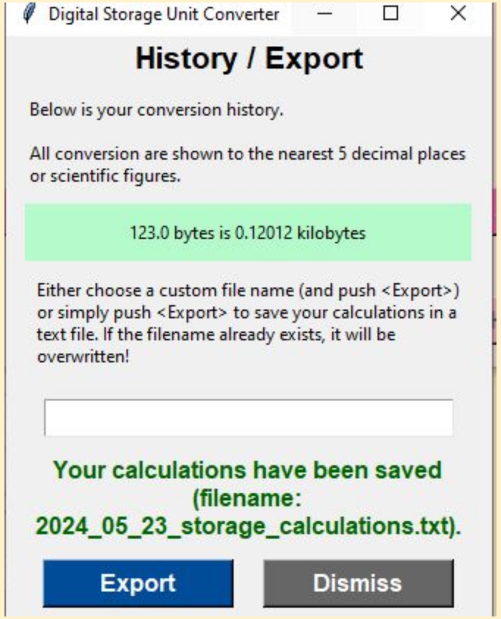
implicating the write_to_file (exporting) function to history/ export window

Component Test Plan <Edit Me>

Test Case	Expected Outcome
Entry grid filled and click convert button (history/export window)	Success message “ Your calculations have been saved(filename: whatever the user entered in entry grid.txt)
Entry grid empty and click convert button (history/export window)	Success message “ Your calculations have been saved(filename: folder location.txt)
After click export button	History txt file saved in the folder History txt file show time and date the file is saved, and all conversion results are all aligned

Component Trialling

The trailing results are the same as my expected outcome.



History / Export

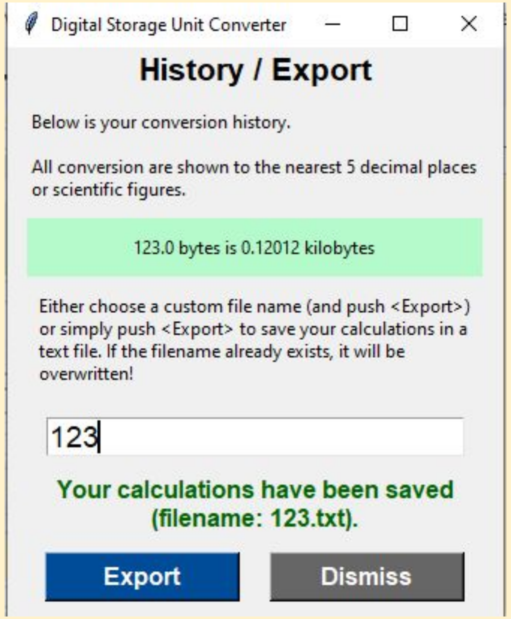
Below is your conversion history.

All conversion are shown to the nearest 5 decimal places or scientific figures.

123.0 bytes is 0.12012 kilobytes

Either choose a custom file name (and push <Export>) or simply push <Export> to save your calculations in a text file. If the filename already exists, it will be overwritten!

Your calculations have been saved (filename: 2024_05_23_storage_calculations.txt).



History / Export

Below is your conversion history.

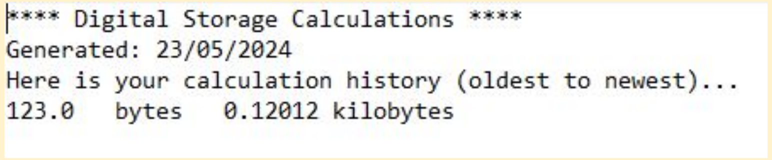
All conversion are shown to the nearest 5 decimal places or scientific figures.

123.0 bytes is 0.12012 kilobytes

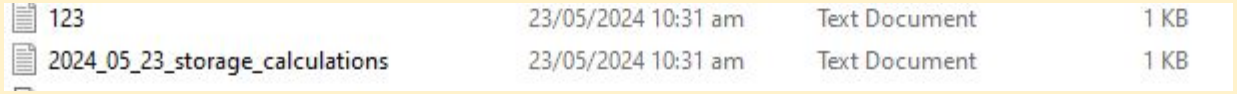
Either choose a custom file name (and push <Export>) or simply push <Export> to save your calculations in a text file. If the filename already exists, it will be overwritten!

123

Your calculations have been saved (filename: 123.txt).



```
**** Digital Storage Calculations ****
Generated: 23/05/2024
Here is your calculation history (oldest to newest)...
123.0   bytes   0.12012 kilobytes
```



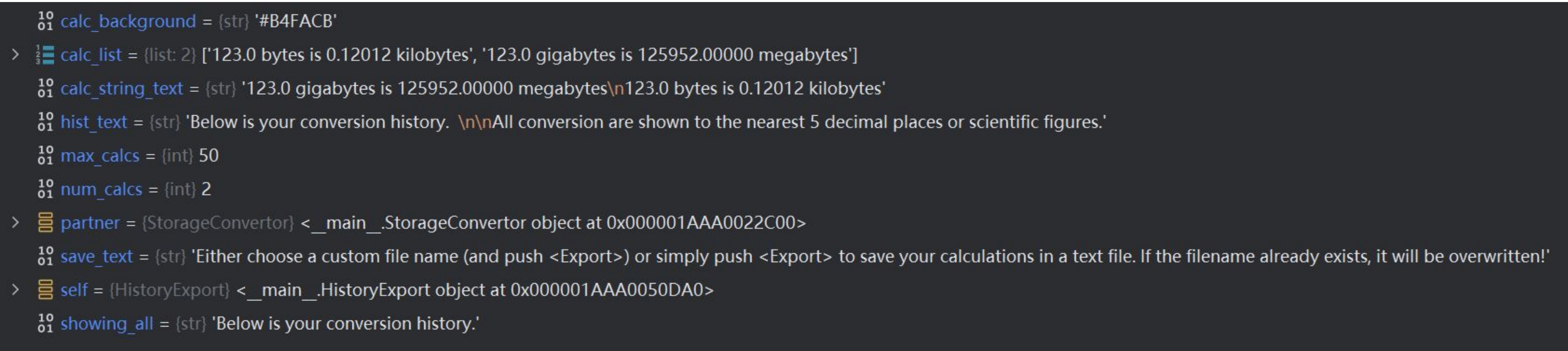
123	23/05/2024 10:31 am	Text Document	1 KB
2024_05_23_storage_calculations	23/05/2024 10:31 am	Text Document	1 KB

Debugging of show history/export gui

During the debugging process I can see how the program stores the conversion and how many has been made .

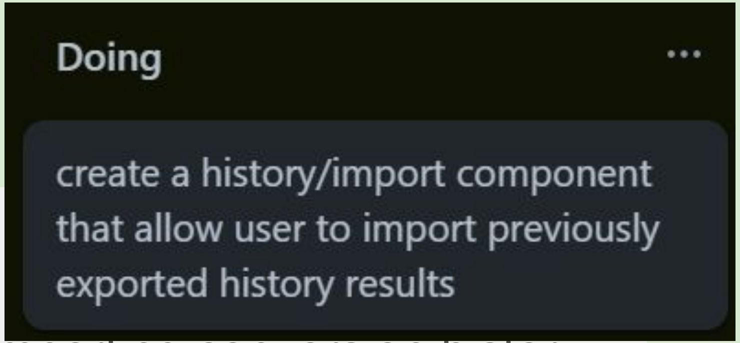
In this case:

- Calc_list = 123.0 bytes is 0.12012 kilobytes',
    - '123.0 gigabytes is 125952.00000 megabytes
- Max_cals = 50
- Num_cals = 2



```
calc_background = {str} '#B4FACB'
> calc_list = {list: 2} ['123.0 bytes is 0.12012 kilobytes', '123.0 gigabytes is 125952.00000 megabytes']
calc_string_text = {str} '123.0 gigabytes is 125952.00000 megabytes\n123.0 bytes is 0.12012 kilobytes'
hist_text = {str} 'Below is your conversion history.  \n\nAll conversion are shown to the nearest 5 decimal places or scientific figures.'
max_calcs = {int} 50
num_calcs = {int} 2
> partner = {StorageConvertor} <__main__.StorageConvertor object at 0x000001AAA0022C00>
save_text = {str} 'Either choose a custom file name (and push <Export>) or simply push <Export> to save your calculations in a text file. If the filename already exists, it will be overwritten!'
> self = {HistoryExport} <__main__.HistoryExport object at 0x000001AAA0050DA0>
showing_all = {str} 'Below is your conversion history.'
```

Component Planning...


Doing

create a history/import component that allow user to import previously exported history results

History/ Import_v1

Purpose: To provide basic functionality for importing conversion history.
Design Choices:
Simple import functionality without advanced features.
Techniques Trialed:
Basic history import.
Outcome: Functionality worked but lacked error handling and advanced features.

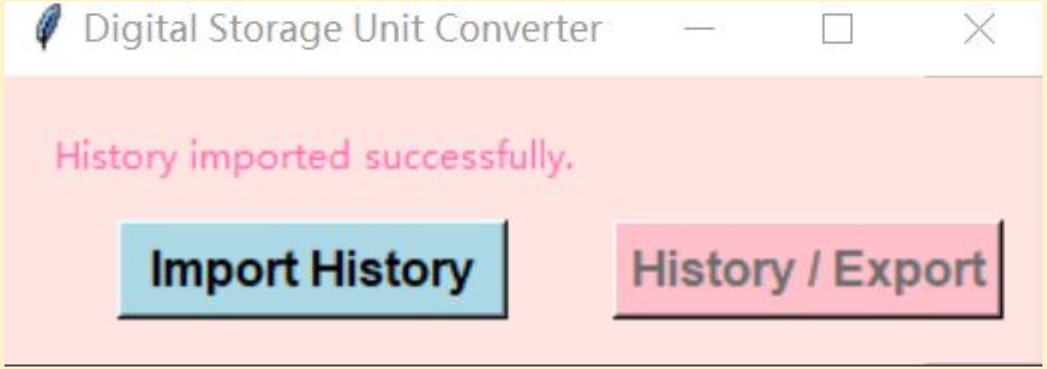
Component Test Plan <Edit Me>

Test Case	Expected Outcome
Click on the history/import button	File explorer opens and ask the user to select the history file to be imported
After import	Show success message"history imported successfully" Imported history also display in the history grid in the history export gui

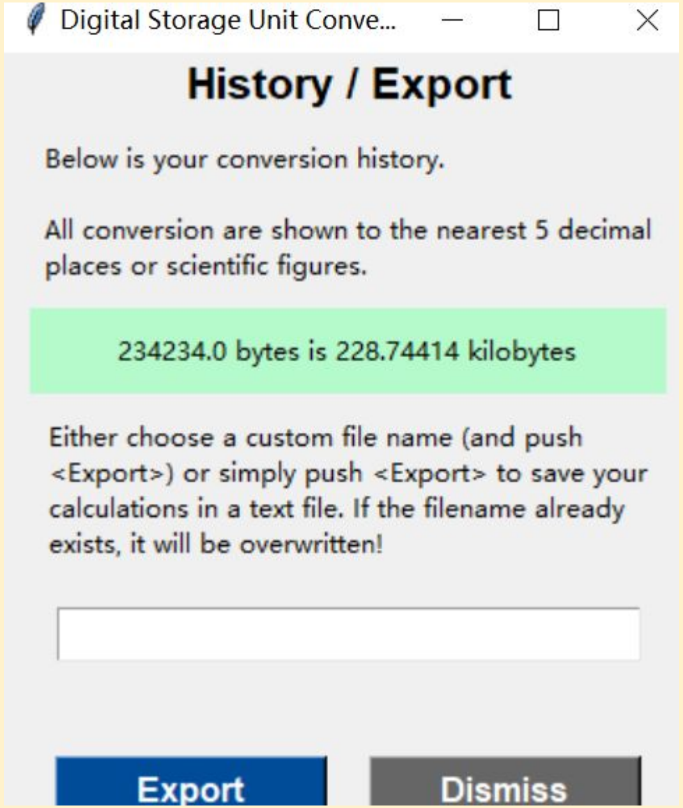
The initial version of the history import component allowed basic importing of conversion history. It worked as expected but needed more advanced features and error handling.

Component Trialling

The trailing results are the same as my expected outcome.


Digital Storage Unit Converter

History imported successfully.


Digital Storage Unit Conve...

History / Export

Below is your conversion history.

All conversion are shown to the nearest 5 decimal places or scientific figures.

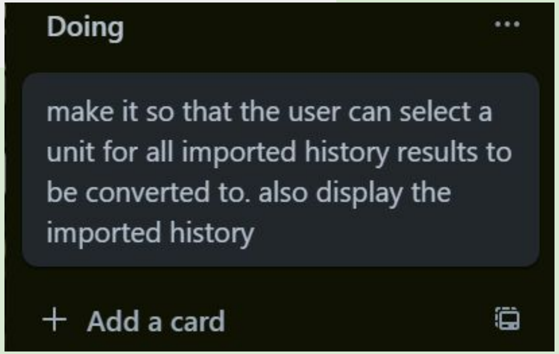
234234.0 bytes is 228.74414 kilobytes

Either choose a custom file name (and push <Export>) or simply push <Export> to save your calculations in a text file. If the filename already exists, it will be overwritten!

https://drive.google.com/file/d/19X39xRfxVNmznlsIus9gFcTl-596dM68/view?usp=sharing

Component Planning...

History/ Import_v2
Purpose: To enhance the import functionality and include large quantity conversion .
Design Choices:
Improved user interface and added function to large quantity conversion
Techniques Trialed:
Enhanced import functionality with better user guidance.
Outcome: Improved usability and functionality by adding function to large quantity conversion


Doing

make it so that the user can select a unit for all imported history results to be converted to. also display the imported history

Component Test Plan <Edit Me>

The second version improved the history import functionality by adding function to large quantity conversion and enhancing the user interface, making it more robust and user-friendly.

Test Case	Expected Outcome
Click on the history/import button	Open history/import gui
Click on import button	File explorer opens and ask the user to select the history file to be imported
Click on export button	File explorer opens and ask the user to enter the name of the txt file to be saved as History txt file saved in the folder History txt file show time and date the file is saved, and all conversion results are all aligned(same format as the history export component)

Component Trialling

The trailing results are the same as my expected outcome.

https://drive.google.com/file/d/1LaoN9PSwA3AnPZFQDvAuP5PZMBONuP1L/view?usp=sharing

Component Testing Evidence <Edit Me>

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_v1

From the testing of the complete program_v1 and from user feedback i got from my peers, I was told the visual appearance of my program doesn't suit the purpose of a Digital Storage Unit Convertor, that the pinky colour theme seems to playful and unserious for a tool. So I decided to use high contrast colour theme but also not too overly colourful. I used Light cyan(#EAF6FF) and light black(#232528), the two colours are based of high contrasting white and black which suits the seriousness of a conversion tool, but both have a shade of blue in them, creating a contrasting harmony.

Complete Program_v1 <test plan>

Test plan:

- Run the program and everything should load as expected(buttons, entry boxes, dropdown boxes)
- Entry grid should allow user to enter any values
- When click on the convert button, if entry grid is empty or filled with non numerical values, a error message should be displayed. Only if a numeric value is entered the conversion result would display
- Both dropdown boxes should allow user to freely select the units they want
- Show solution button should be disabled before any conversion is made, and enabled after a conversion is made. When click on it, a new window should appear and shows the step process to the conversion.
- History/export should be disabled before any conversion is made, and enabled after a conversion is made. When click on it, a new window should appear and shows the previous conversion history. And below you can enter a name in the entry grid for the exported history txt file. There is also a dismiss button where you can use it to close the window.
- When click on the history/import button, a new window should appear, on this window you can import back in your previous exported history txt file by clicking on the import history button. On the right of this button, there is a drop down box where you can select a unit for all the results you just imported back in, this will allow you to convert all of the results you just imported back in them to another unit with just one click on the convert and export button.

Complete Program_v1 <testing evidence>

https://drive.google.com/file/d/1Pozyp-3diYfQgX7NZv4B4DxhpZ6X82iG/view?usp=sharing

Complete Program_v2

From the testing of the complete program_v1 and from user feedback i got from my peers, I was told that there weren’t enough instructions or informations that tells him how to use the program. So i decide to add a instruction button, and when you click on the button it will open a instruction window that explains the functionality of each button and how to use the program

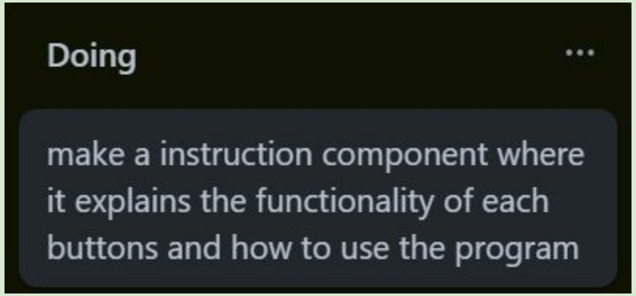
Complete Program_v2 <test plan>

Test plan:

- Run the program and everything should load as expected(buttons, entry boxes, dropdown boxes)
- Entry grid should allow user to enter any values
- when you click on the instruction button it will open a instruction window that explains the functionality of each button and how to use the program
- When click on the convert button, if entry grid is empty or filled with non numerical values, a error message should be displayed. Only if a numeric value is entered the conversion result would display
- Both dropdown boxes should allow user to freely select the units they want
- Show solution button should be disabled before any conversion is made, and enabled after a conversion is made. When click on it, a new window should appear and shows the step process to the conversion.
- History/export should be disabled before any conversion is made, and enabled after a conversion is made. When click on it, a new window should appear and shows the previous conversion history. And below you can enter a name in the entry grid for the exported history txt file. There is also a dismiss button where you can use it to close the window.
- When click on the history/import button, a new window should appear, on this window you can import back in your previous exported history txt file by clicking on the import history button. On the right of this button, there is a drop down box where you can select a unit for all the results you just imported back in, this will allow you to convert all of the results you just imported back in them to another unit with just one click on the convert and export button.

Component Planning...

Instruction gui
Purpose: To provide detailed instructions and improve layout.
Design Choices:
Detailed instructions and improved layout for better user guidance.
Techniques Trialed:
Comprehensive instructions and enhanced layout.
Outcome: Significantly improved usability by providing clear and detailed guidance.


Doing

make a instruction component where it explains the functionality of each buttons and how to use the program

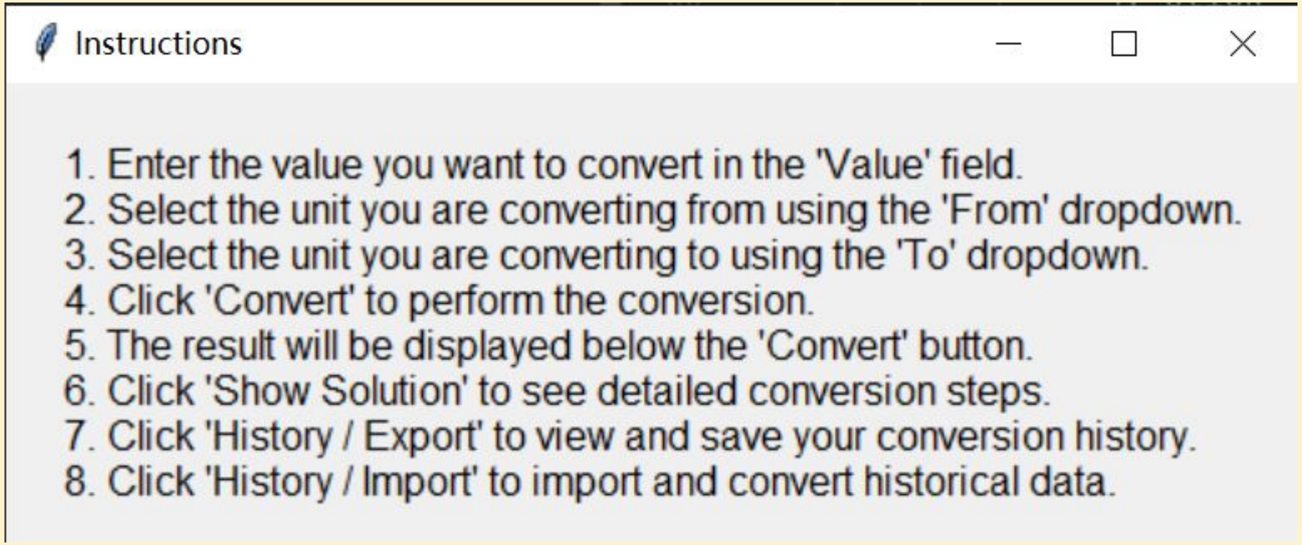
Component Test Plan <Edit Me>

Test Case	Expected Outcome
Click on the instruction button	Display a new window that has all the instruction on there

instruction GUI included detailed instructions and an improved layout, making it much easier for users to understand and navigate the program.

Component Trialling

The trailing results are the same as my expected outcome.


Instructions

1. Enter the value you want to convert in the 'Value' field.
2. Select the unit you are converting from using the 'From' dropdown.
3. Select the unit you are converting to using the 'To' dropdown.
4. Click 'Convert' to perform the conversion.
5. The result will be displayed below the 'Convert' button.
6. Click 'Show Solution' to see detailed conversion steps.
7. Click 'History / Export' to view and save your conversion history.
8. Click 'History / Import' to import and convert historical data.

Complete Program_v2 <testing evidence>

https://drive.google.com/file/d/1_2V8FlRwN8lcYpyJUU55ShbGs9_-CFy0/view?usp=sharing

Complete Program_v3

From the testing of the complete program_v2 and from user feedback i got from my peers, I found out that my history/export gui won‘t allow certain keywords that reserved for system file of Windows(con, con1, etc). so I will make some changes to the history export component that detects those windows reserved keyword, and display error messages when these keywords are entered. I was also told that most of my component gui can be opened many times, resulting a lot of component window being opened and slowing down the program. This will significantly affect the usability and functionality of the program. So i will make some changes to all the GUIs and only allow one to be opened.

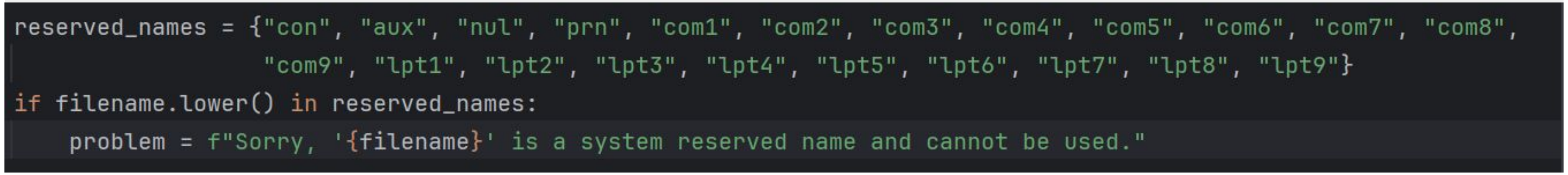
Complete Program_v3 <test plan>

Test plan:

- Run the program and everything should load as expected(buttons, entry boxes, dropdown boxes)
- Entry grid should allow user to enter any values
- when you click on the instruction button it will open a instruction window that explains the functionality of each button and how to use the program
- When click on the convert button, if entry grid is empty or filled with non numerical values, a error message should be displayed. Only if a numeric value is entered the conversion result would display
- Both dropdown boxes should allow user to freely select the units they want
- Show solution button should be disabled before any conversion is made, and enabled after a conversion is made. When click on it, a new window should appear and shows the step process to the conversion.
- History/export should be disabled before any conversion is made, and enabled after a conversion is made. When click on it, a new window should appear and shows the previous conversion history. And below you can enter a name in the entry grid for the exported history txt file, except special symbols and windows reserved keywords. There is also a dismiss button where you can use it to close the window.
- When click on the history/import button, a new window should appear, on this window you can import back in your previous exported history txt file by clicking on the import history button. On the right of this button, there is a drop down box where you can select a unit for all the results you just imported back in, this will allow you to convert all of the results you just imported back in them to another unit with just one click on the convert and export button.
- all component GUIs can only have one opened at a time.

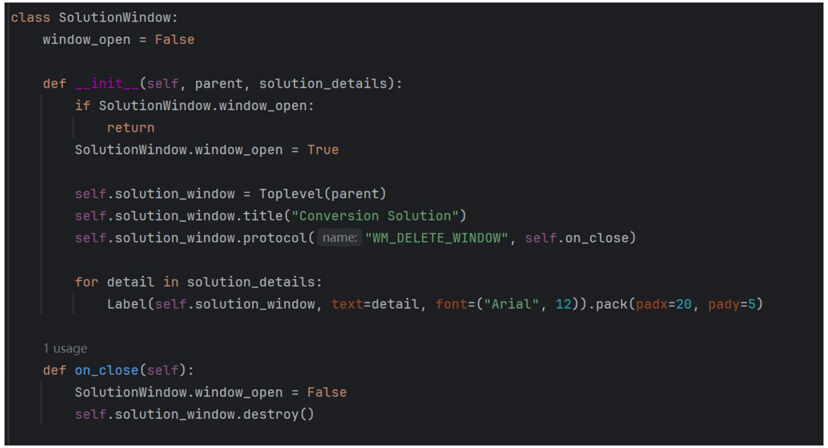
Complete Program_v3 changes

I listed the reserved names, and set that if the filename is one of the reserved names, the program will return a error message that say “sorry, {file name} is a system reserved name and cannot be used.



```python
reserved_names = {"con", "aux", "nul", "prn", "com1", "com2", "com3", "com4", "com5", "com6", "com7", "com8",
                  "com9", "lpt1", "lpt2", "lpt3", "lpt4", "lpt5", "lpt6", "lpt7", "lpt8", "lpt9"}
if filename.lower() in reserved_names:
    problem = f"Sorry, '{filename}' is a system reserved name and cannot be used."
```

Complete Program_v3 changes



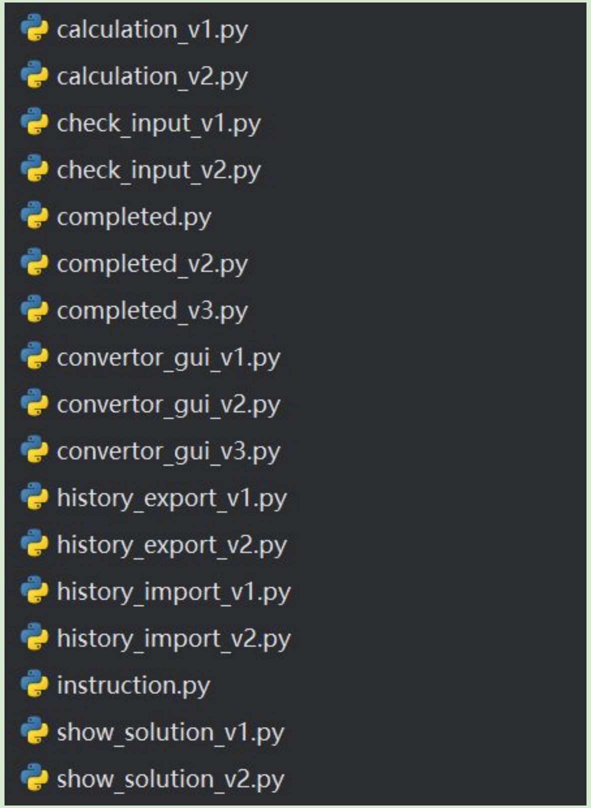
```
class SolutionWindow:
    window_open = False

    def __init__(self, parent, solution_details):
        if SolutionWindow.window_open:
            return
        SolutionWindow.window_open = True

        self.solution_window = Toplevel(parent)
        self.solution_window.title("Conversion Solution")
        self.solution_window.protocol( name: "WM_DELETE_WINDOW", self.on_close)

        for detail in solution_details:
            Label(self.solution_window, text=detail, font=("Arial", 12)).pack(padx=20, pady=5)

    1 usage
    def on_close(self):
        SolutionWindow.window_open = False
        self.solution_window.destroy()
```

This code introduces class-level flags to ensure that each type of window can only be opened once at a time. The flags are set to ‘true’ when a window is opened and reset to ‘false’ when it is closed, preventing multiple instances of the same window from being opened simultaneously.

Complete Program_v3 <testing evidence>

https://drive.google.com/file/d/1iG61lBguPeqkIfSIz4K3ROHrLq8LdVpp/view?usp=sharing

Version Control


calculation_v1.py
calculation_v2.py
check_input_v1.py
check_input_v2.py
completed.py
completed_v2.py
completed_v3.py
convertor_gui_v1.py
convertor_gui_v2.py
convertor_gui_v3.py
history_export_v1.py
history_export_v2.py
history_import_v1.py
history_import_v2.py
instruction.py
show_solution_v1.py
show_solution_v2.py

I have uploaded all versions of my component and main program onto github. Doing so helps me to spot and fix up errors I made during the development process. Most importantly, these different versions of components shows how my program has been improved.

I made individual slides about major changes I made in different version of components, With each version, I added new functionality and eliminated errors or problems that occurred during testing. In the final version, the program is fully refined and functions perfectly, handling both expected and unexpected cases.

Relevant Implications Addressed

You might wish to use this slide to explain how you have addressed the relevant implications if you have not done so in previous slides.

Comments go here

Complex Processes - Discussion

In the planning phase of my program, I explored multiple ideas that I could expand on with the components I learned from tutorials and decided on making a Digital Storage Unit Converter. I first created a program structure diagram using wireframe.cc, which provided a clear visual of the complete program. This diagram helped me set clear goals and understand the components I needed to develop. Additionally, I used a Trello board to plan out and record tasks for each component, which was instrumental in keeping track of my progress and reminding me what to do next. By breaking down the project into manageable tasks and setting deadlines, I could efficiently manage the trialling, testing, and refinement processes.

During the initial development phase, I integrated components I was familiar with from tutorials to ensure basic functionality. These components were tested and debugged during the tutorial phase, which gave me confidence in their reliability. This approach streamlined the development process, reducing the time needed for research and allowing me to focus more on testing and refining new components. Implementing known components not only enhanced the program's functionality but also improved usability by allowing me to dedicate more time to making these components user-friendly with clear instructions.

Complex Processes - Discussion

As I developed the program, I also researched and integrated new components, such as dropdown boxes for unit selection. There were many instances where integrating new components with the pre-existing ones posed challenges. For example, ensuring that the dropdown selections interacted correctly with the calculation functions required continuous testing and refinement. I used a combination of debugging tools and user testing to identify and fix issues, ensuring each component's functionality and user-friendliness.

User feedback was crucial in refining the program. During the user testing phase, a peer pointed out that the program lacked clear instructions, which left them confused about the purpose of each button. Based on this feedback, I implemented a detailed instruction GUI that explains the function of each button, significantly improving the program's usability. This addition ensured that future users would have a better understanding of how to navigate and use the program effectively. Additionally, I refined the color theme of the program based on user suggestions to enhance its aesthetics, making it more visually appealing and easier to use.My peer also pointed out that the history/export GUI does not permit certain reserved keywords used by Windows system files (e.g., con, con1). To address this issue, I updated the history export component to detect these reserved keywords and display appropriate error messages when such keywords are entered. Additionally, I was informed that multiple instances of the GUI components can be opened simultaneously, leading to performance issues and reducing the program's usability and functionality. To resolve this, I modified all GUI components to ensure only one instance can be opened at a time. These changes aim to enhance the overall user experience and ensure the program functions smoothly.

Complex Processes - Discussion

Reflecting on my development process, I realized that the planning phase could have been more efficient. Initially, I experimented with ideas like a donation function and other features that were ultimately unnecessary and time-consuming. These trials did not contribute to the final product and diverted my focus from more critical components. If I could redo this project, I would use that time to work on eye-catching and useful components or conduct more testing to further improve the functionality and usability of the program.

Another aspect I would change if I had more time would be to consolidate all the GUI elements into a single window. My current program consists of multiple GUI windows for different components, which can be confusing and annoying for users who have to deal with numerous pop-up windows. Combining these into a single window would streamline the user experience, making it more intuitive and enhancing the overall aesthetics of the program.

Despite these challenges, something I did well in the development process was the creation and integration of each component. Each component was thoroughly tested individually to ensure smooth operation, which made the final integration process straightforward. Consistent commenting in my code and comprehensive testing of each version of the components helped in quickly identifying and resolving any issues that arose during integration. This preparation ensured that combining the components into a fully functional program was a simple process of integrating defined functions into a shared class.

Complex Processes - Discussion

The thorough planning and use of project management tools like Trello and wireframe.cc played a crucial role in managing the project efficiently. They helped in organizing tasks, tracking progress, and incorporating user feedback effectively. The integration of user suggestions and continuous testing and refinement of components ensured the development of a high-quality digital technology outcome. Reflecting on my process, while there were areas for improvement, the overall project was successful due to the thorough planning, testing, and incorporation of feedback. These steps ensured that the final product was functional, user-friendly, and aesthetically pleasing.

Reflecting on my development process, I recognize that the iterative approach of planning, testing, and refining was crucial in achieving a high-quality outcome. The use of project management tools and the incorporation of user feedback were particularly effective in addressing usability and functionality issues. While there are always areas for improvement, such as consolidating GUI elements and refining the initial planning phase, the overall project demonstrates a successful application of digital technology development processes.

Complex Processes - Discussion

Aesthetics were also a key consideration. I used a high-contrast color theme to ensure readability and visual appeal. The color scheme consisted of light cyan (#EAF6FF) and light black (#232528), chosen for their high contrast and harmonious look. Buttons were centrally aligned and symmetric, enhancing the visual structure of the program. Large, bold text with contrasting colors was used to make the content easily readable, which is particularly important for users with visual impairments.

Functionality was a primary focus throughout the development process. I continuously tested each component to ensure it performed its intended tasks correctly. For example, the dropdown boxes for selecting units simplified the user experience by eliminating the need for manual input, reducing the risk of errors. The history import/export components were designed to allow users to easily save and retrieve their conversion history, enhancing the program's functionality and user convenience.

To address the usability aspect, I made sure the program was easy to navigate. I used descriptive labels like ‘From:’ and ‘To:’ to guide users through the conversion process. I also implemented clear error messages, such as notifying the user if they try to convert with empty entry fields or invalid values. These error messages helped users understand what went wrong and how to correct their input, improving their overall experience with the program.