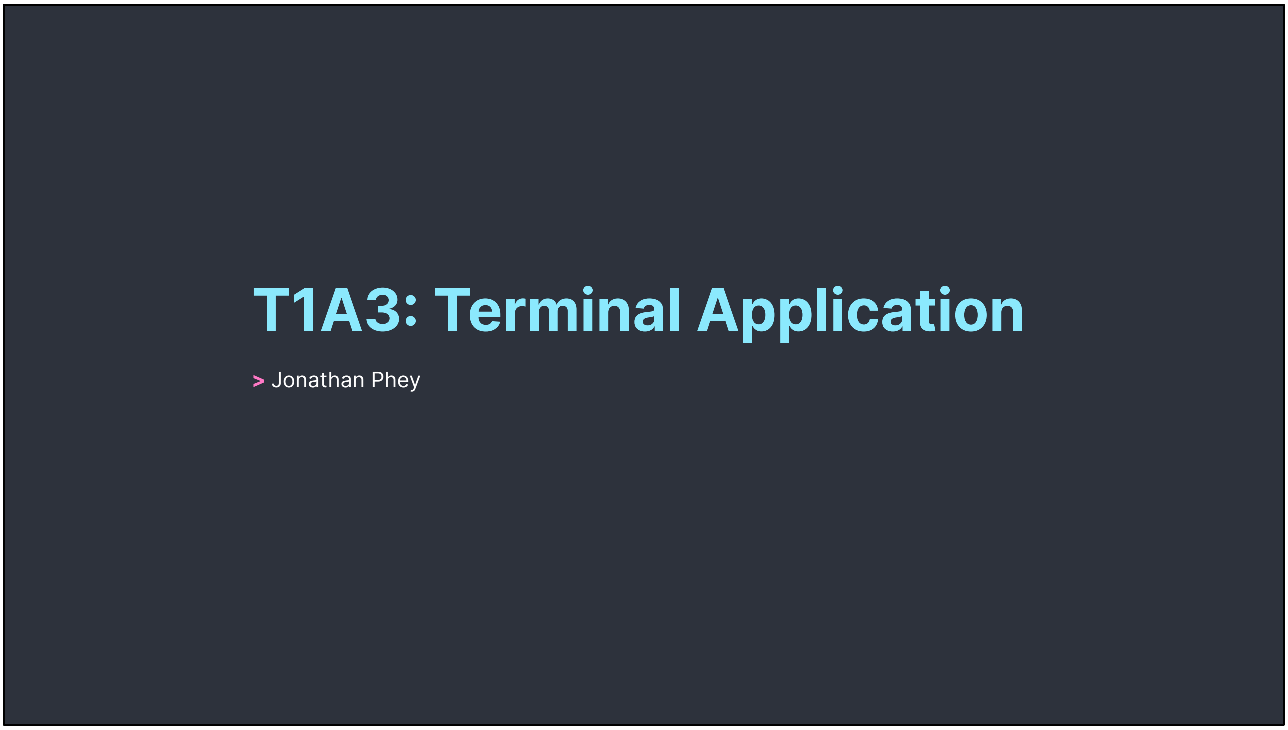
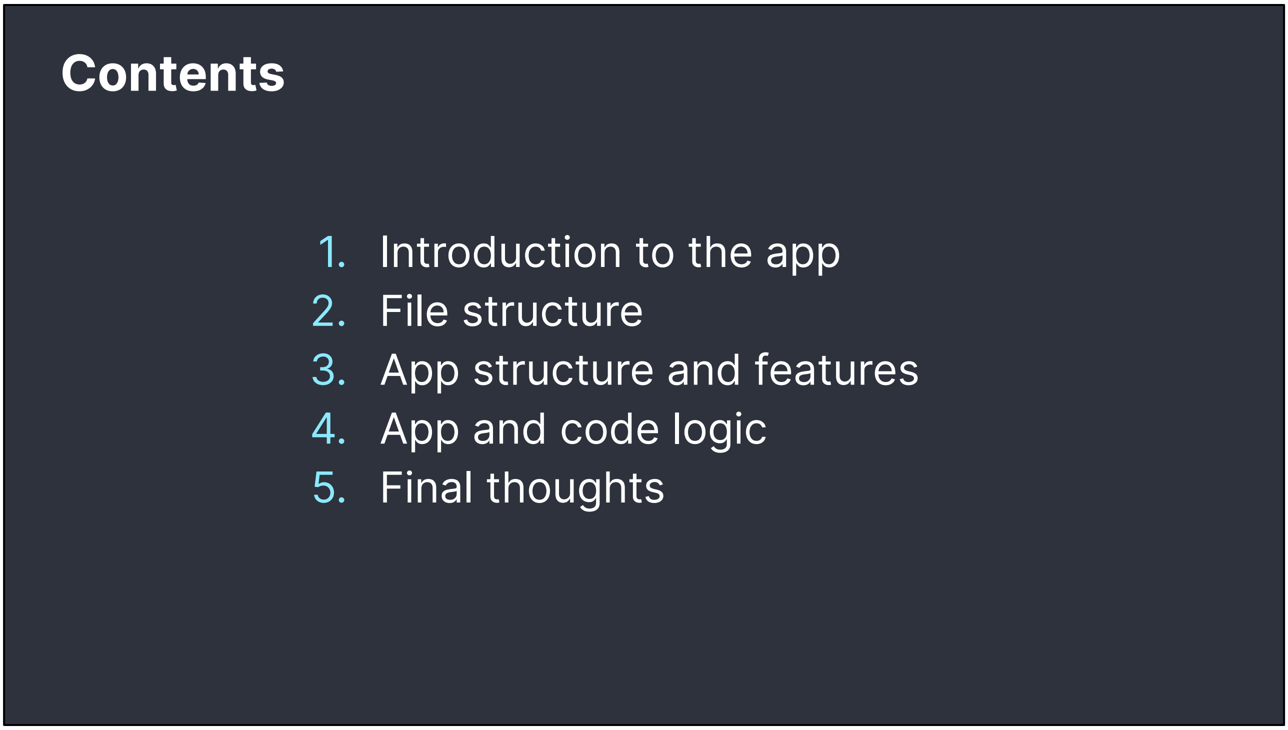


T1A3: Terminal Application

> Jonathan Phey

Introduction

- Hi! I’m Jonathan Phey, or Jon Phey, and welcome to my slides and presentation of my Terminal Application assignment
- In these slides, I’m going to take you through my terminal application, its features, the logic behind the code and a review of my build process



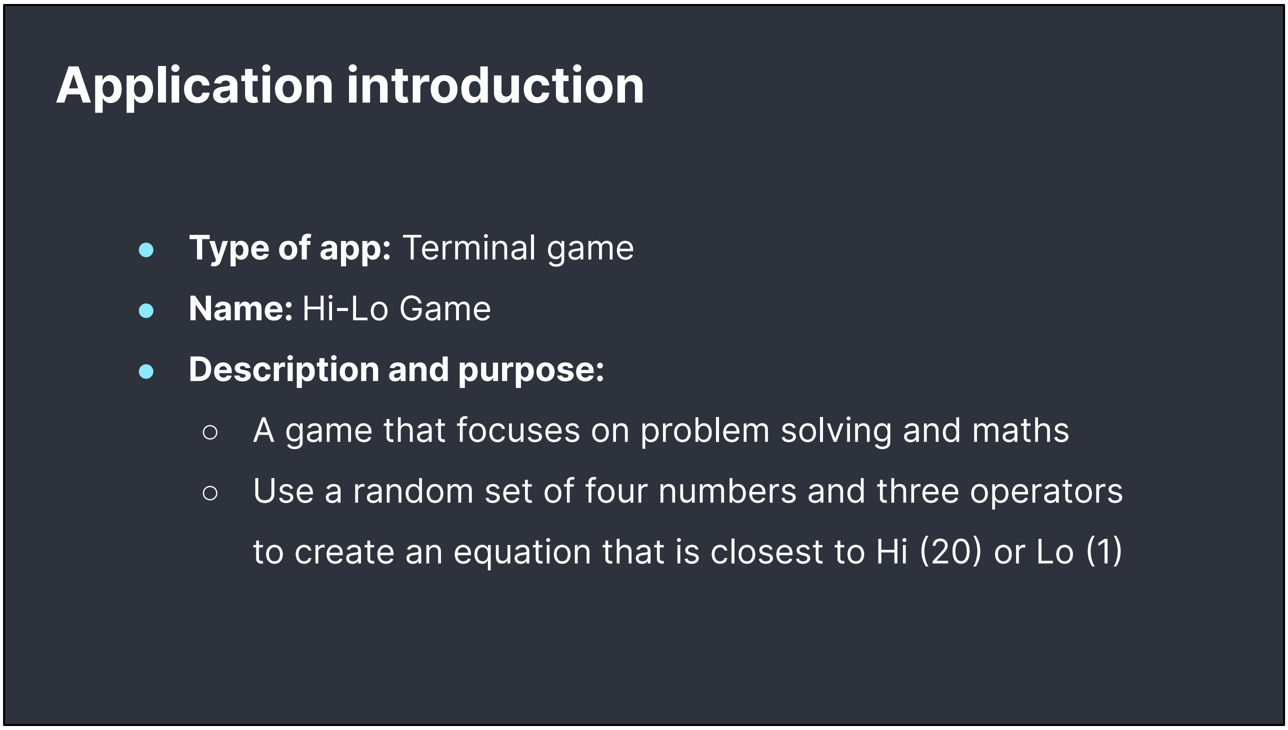
Contents

1. Introduction to the app
2. File structure
3. App structure and features
4. App and code logic
5. Final thoughts

Contents / Agenda

As a quick overview of what I'll be going through:

1. First, I'll **introduce the app**, where I'll give you a quick overview of what the app is and a short description
2. Secondly, I'll give a brief overview of the **file structure** for the directory and the provided files
3. Next, the **app structure** will provide an insight into how the app and app files are structured
4. Then, I'll provide an overview of the **app features** and how they're used
5. The **app and code logic** portion is where I'll walkthrough the important parts of my code, and detail where I've placed crucial application logic
6. And then I'll end on my **final thoughts**, which will wrap up with a conclusion and sharing of my favourite parts of the build process and some of the challenges I faced



Application introduction

- **Type of app:** Terminal game
- **Name:** Hi-Lo Game
- **Description and purpose:**
  - A game that focuses on problem solving and maths
  - Use a random set of four numbers and three operators to create an equation that is closest to Hi (20) or Lo (1)

Application introduction

Type of app:

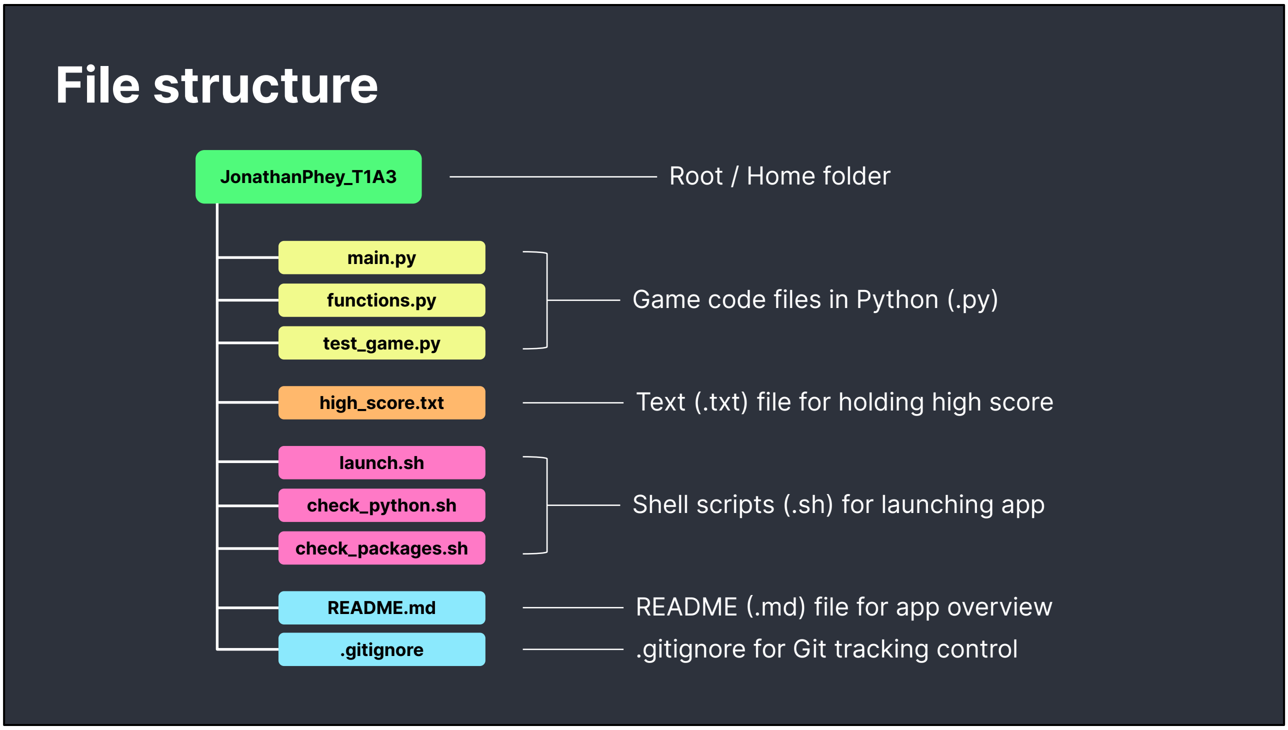
- For my terminal assignment application, I decided to build a game that would be played in the terminal

Name:

- The name of the game is ‘Hi-Lo’ (which is a shortened version of High - Low)

Description and purpose:

- The inspiration for the game came from a survival game show I recently watched, where they played this game against each other, and I wanted to turn it into an application that everyone could play in their own time
- Hi-Lo is a game that focuses on problem solving and mathematics, using the hand of cards that have been dealt
- Every round, the player will be given a hand of 7 cards, which consist of 4 number cards and 3 operator cards
- Using those 7 cards, the player must create a valid equation that is closest or equal to Hi (which represents the value 20) or Lo (which represents the value 1), depending on which option they chose
- The closer the player gets to making an equation equalling to the chosen Hi or Lo, the more points they will score
  - Each round, the player will receive a score depending on how close they got to Hi or Lo, and a high score will be maintained



File structure

JonathanPhey_T1A3 — Root / Home folder

- main.py
- functions.py
- test_game.py

Game code files in Python (.py)

- high_score.txt — Text (.txt) file for holding high score

- launch.sh
- check_python.sh
- check_packages.sh

Shell scripts (.sh) for launching app

- README.md — README (.md) file for app overview
- .gitignore — .gitignore for Git tracking control

File structure

- For the files that are included in the whole package or .zip file, I've used one home folder (**JonathanPhey_T1A3**) that houses all of the files that were required for the build and are required for running and launching the application

Python files (.py)

- **main.py** is the main Python script for the Hi-Lo game application. It contains the core logic and functionality of the game, and imports the necessary Python packages required.
- **functions.py** holds the various functions and classes used in my game, including the card-related classes and game logic functions, such as the equation and score logic.
- **test_game.py** is the main testing script, which allows one to test different parts of the code and ensures that it's working as expected. Players are not expected to have much interaction with this file, unless they decide to build out further tests to enhance the app.

high_score.txt

- **high_score.txt** is a simple .txt file that holds a player's high score as they play the game. This file holds a starting score of 0, will update whenever the player

- scores more than the high score, and will be remembered after subsequent playthroughs.

Shell scripts (.sh)

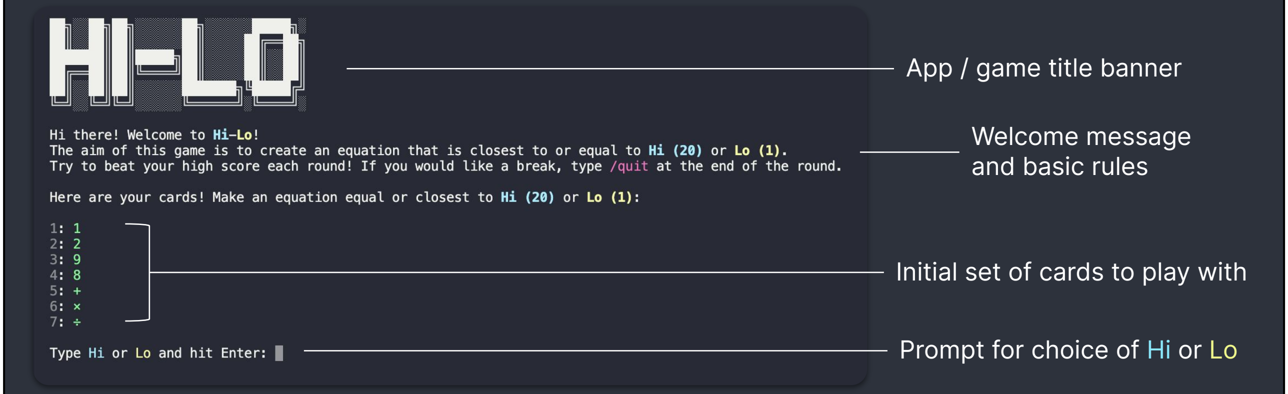
- **launch.sh** is the main shell script used to launch the game. It contains commands and other instructions to run the Python program, namely from…
    - **check_python.sh**, which includes commands to check if Python and the right Python version is installed, and prompts the player to install it if required and
    - **check_packages.sh,** which includes commands to check if the necessary Python packages and modules are installed.

Finally,

- **README.md** is a Markdown file that holds the main documentation for the app. It contains information about the app and game, how to run it, how it works, and additional details pertaining to the build process and attributions.
- And a **.gitignore** is utilised for Git version control, ensuring unnecessary files and directories are ignored and not tracked by Git. Players are expected to have minimal interaction with this file.

App structure and features

> LAUNCH SCREEN



```
HI-LO

Hi there! Welcome to Hi-Lo!
The aim of this game is to create an equation that is closest to or equal to Hi (20) or Lo (1).
Try to beat your high score each round! If you would like a break, type /quit at the end of the round.

Here are your cards! Make an equation equal or closest to Hi (20) or Lo (1):

1: 1
2: 2
3: 9
4: 8
5: +
6: ×
7: ÷

Type Hi or Lo and hit Enter: 
```

App / game title banner

Welcome message
and basic rules

Initial set of cards to play with

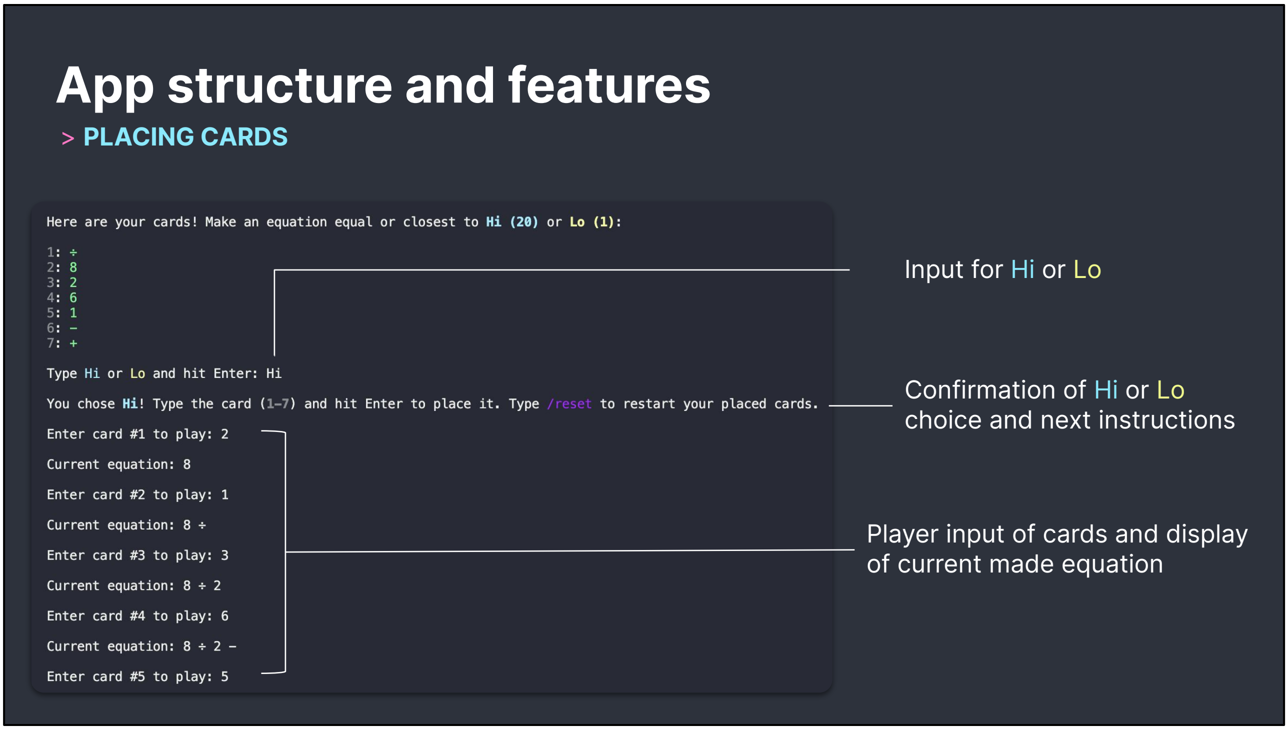
Prompt for choice of Hi or Lo

Application structure

When players first launch the app, they are met with this **launch screen.**

Launch screen

- The launch screen consists of a **banner** that introduces the name of the game, Hi-Lo, in large characters, which was made using the ‘art’ Python package
- Underneath the banner is a **welcome message** and a short set of **basic rules**, which briefly tells the player the aim of the game and how they can /quit out of the game after each round if needed
    - The coloured font for highlight words such as *Hi*, *Lo*, */quit* and the following coloured text were all made possible by using the ‘rich’ and ‘colorama’ Python packages
- The **initial set of cards** are displayed underneath the welcome message, and this is the player’s first set of 4 number cards and 3 operator cards that they are to use to play the game
    - Each card or value matches with a number that they can later input to play the corresponding card in order
- The first actual input prompt from the player is the **prompt for choice of Hi or Lo**, where the player will choose if they want to make an equation closest to Hi (20) or Lo (1)
    - Users can type hi / Hi or lo / Lo and hit Enter to continue with the game



App structure and features

> PLACING CARDS

```
Here are your cards! Make an equation equal or closest to Hi (20) or Lo (1):

1: ÷
2: 8
3: 2
4: 6
5: 1
6: −
7: +

Type Hi or Lo and hit Enter: Hi

You chose Hi! Type the card (1–7) and hit Enter to place it. Type /reset to restart your placed cards.

Enter card #1 to play: 2

Current equation: 8

Enter card #2 to play: 1

Current equation: 8 ÷

Enter card #3 to play: 3

Current equation: 8 ÷ 2

Enter card #4 to play: 6

Current equation: 8 ÷ 2 −

Enter card #5 to play: 5
```

Input for Hi or Lo

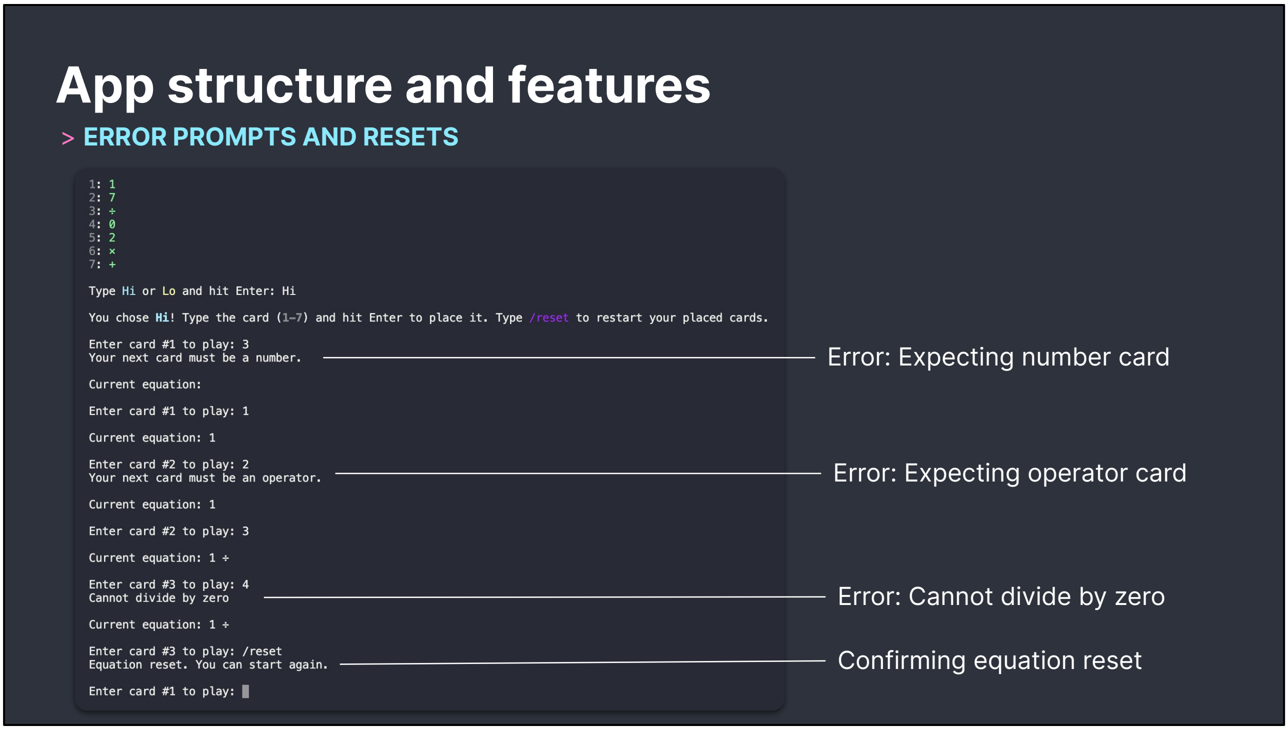
Confirmation of Hi or Lo choice and next instructions

Player input of cards and display of current made equation

Application structure

Placing cards

- The following screen capture shows the structure following the player entering the choice to play for Hi or Lo
- Once the player inputs ‘hi’ or ‘lo’ and hits Enter, a message confirms their choice with “You chose [hi]/[lo]!” and tells the player to start playing their cards by typing the corresponding card number (1-7) and hitting Enter
- I’ve shown a snippet or simulation of what the output appears as when the player begins to place their cards.
    - After each time a player places their card, the app will show the current equation to the player so that they can track what their currently made equation is
    - The prompt for a card to be played will continue until 7 cards are correctly placed and a valid equation has been created



App structure and features

> ERROR PROMPTS AND RESETS

```
1: 1
2: 7
3: ÷
4: 0
5: 2
6: ×
7: +

Type Hi or Lo and hit Enter: Hi

You chose Hi! Type the card (1-7) and hit Enter to place it. Type /reset to restart your placed cards.

Enter card #1 to play: 3
Your next card must be a number.

Current equation:

Enter card #1 to play: 1

Current equation: 1

Enter card #2 to play: 2
Your next card must be an operator.

Current equation: 1

Enter card #2 to play: 3

Current equation: 1 ÷

Enter card #3 to play: 4
Cannot divide by zero

Current equation: 1 ÷

Enter card #3 to play: /reset
Equation reset. You can start again.

Enter card #1 to play: 
```

Error: Expecting number card

Error: Expecting operator card

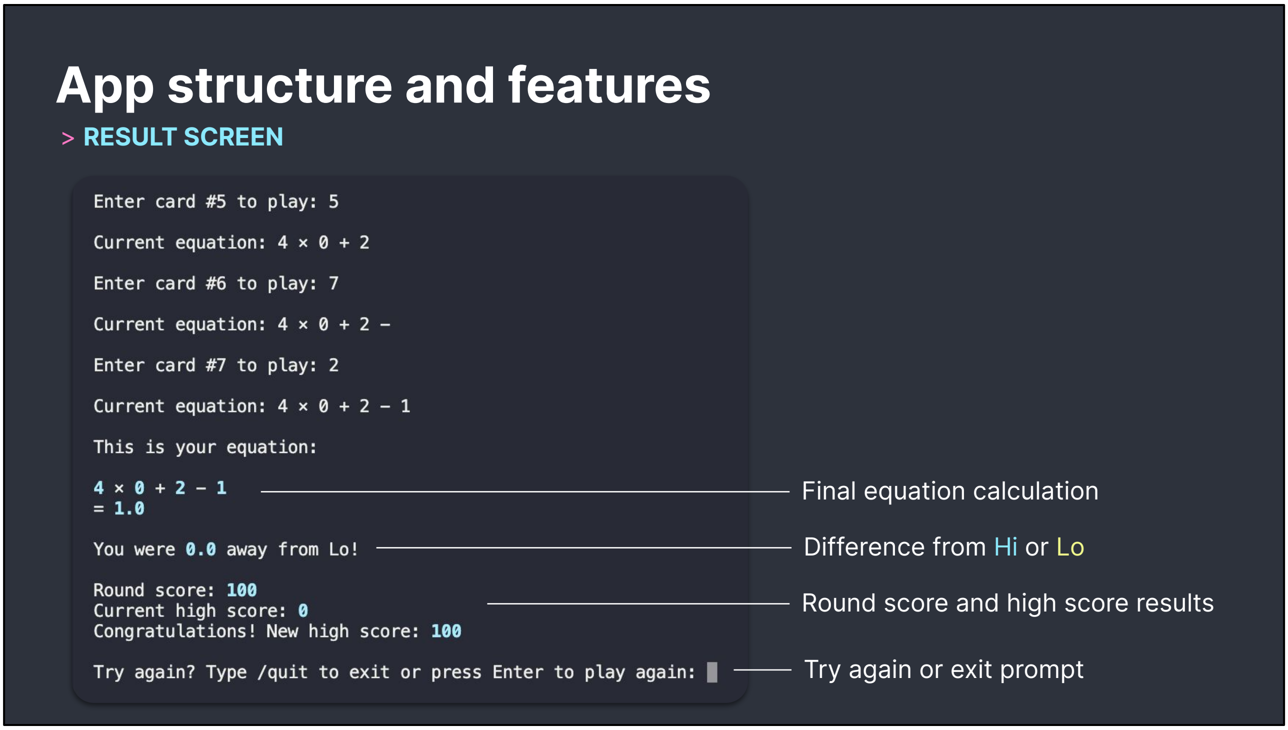
Error: Cannot divide by zero

Confirming equation reset

Application structure

Error prompts and resets

- Throughout the application, there are various error prompts shown to the player, usually when the app expects a certain type of input
- The goal of the error prompts are to guide the player and to ensure that the player plays an appropriate card or types out an appropriate input when playing a card
- The errors will inform the player of when they're not able to play a certain card, with a short reason of why not. For example,
    - An equation usually begins with a number, so playing an operator card first will result in an error
    - Similarly, in this game, a number must be followed by an operator card, so playing two numbers in a row is not allowed
- In general, dividing by zero is not allowed from a mathematics perspective, so an error appears when an attempt to place a '0' after a division operator
- In the case where the player decides to restart their equation, they can type '/reset' to start again
    - The option to type '/reset' also removes the possibility that the player gets stuck in a situation where the equation cannot be completed, such as when the last card remaining in the player's hand is a '0', but the card placed before it is a division operator



App structure and features

> RESULT SCREEN

```
Enter card #5 to play: 5

Current equation: 4 × 0 + 2

Enter card #6 to play: 7

Current equation: 4 × 0 + 2 -

Enter card #7 to play: 2

Current equation: 4 × 0 + 2 - 1

This is your equation:

4 × 0 + 2 - 1
= 1.0

You were 0.0 away from Lo!

Round score: 100
Current high score: 0
Congratulations! New high score: 100

Try again? Type /quit to exit or press Enter to play again: █
```

Final equation calculation

Difference from Hi or Lo

Round score and high score results

Try again or exit prompt

Application structure

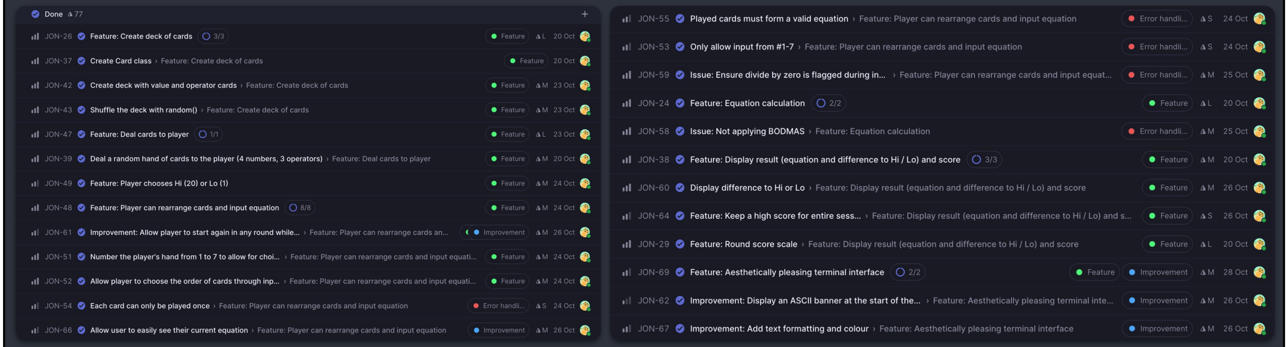
Result screen

- Following the player successfully placing down all seven cards from their hand and a valid equation has been created, the **result screen** will appear
- The result screen displays several parts, including:

- The player’s **final equation** that they made and what the equation equals
- The **difference from Hi (20) or Lo (1)**, which is in absolute values
- Their **round score** out of 100, with a round score of 100 meaning the player made an equation exactly equal to Hi or Lo. This round score decreases as the player’s equation gets further away from the intended Hi or Lo.
- The current **high score** is also shown, as well as a congratulatory note if the player’s current round score is higher than the previous high score. If that’s the case, the high score will be updated and maintained for future rounds.
- Finally, a prompt to try again will appear, and the player can stop playing by typing /quit

App structure and features

> FEATURE LIST



Done 77

JON-26	Feature: Create deck of cards	3/3		Feature	L	20 Oct
JON-37	Create Card class		Feature: Create deck of cards	Feature		20 Oct
JON-42	Create deck with value and operator cards		Feature: Create deck of cards	Feature	M	23 Oct
JON-43	Shuffle the deck with random()		Feature: Create deck of cards	Feature	M	23 Oct
JON-47	Feature: Deal cards to player	1/1		Feature	L	23 Oct
JON-39	Deal a random hand of cards to the player (4 numbers, 3 operators)		Feature: Deal cards to player	Feature	M	20 Oct
JON-49	Feature: Player chooses Hi (20) or Lo (1)			Feature	M	24 Oct
JON-48	Feature: Player can rearrange cards and input equation	8/8		Feature	M	24 Oct
JON-61	Improvement: Allow player to start again in any round while...		Feature: Player can rearrange cards an...	Improvement	M	26 Oct
JON-51	Number the player's hand from 1 to 7 to allow for choi...		Feature: Player can rearrange cards and input equati...	Feature	M	24 Oct
JON-52	Allow player to choose the order of cards through inp...		Feature: Player can rearrange cards and input equati...	Feature	M	24 Oct
JON-54	Each card can only be played once		Feature: Player can rearrange cards and input equation	Error handli...	S	24 Oct
JON-66	Allow user to easily see their current equation		Feature: Player can rearrange cards and input equation	Improvement	M	26 Oct
JON-55	Played cards must form a valid equation		Feature: Player can rearrange cards and input equation	Error handli...	S	24 Oct
JON-53	Only allow input from #1-7		Feature: Player can rearrange cards and input equation	Error handli...	S	24 Oct
JON-59	Issue: Ensure divide by zero is flagged during in...		Feature: Player can rearrange cards and input equat...	Error handli...	M	25 Oct
JON-24	Feature: Equation calculation	2/2		Feature	L	20 Oct
JON-58	Issue: Not applying BODMAS		Feature: Equation calculation	Error handli...	M	25 Oct
JON-38	Feature: Display result (equation and difference to Hi / Lo) and score	3/3		Feature	M	20 Oct
JON-60	Display difference to Hi or Lo		Feature: Display result (equation and difference to Hi / Lo) and score	Feature	M	26 Oct
JON-64	Feature: Keep a high score for entire sess...		Feature: Display result (equation and difference to Hi / Lo) and s...	Feature	S	26 Oct
JON-29	Feature: Round score scale		Feature: Display result (equation and difference to Hi / Lo) and score	Feature	L	20 Oct
JON-69	Feature: Aesthetically pleasing terminal interface	2/2		Feature Improvement	M	28 Oct
JON-62	Improvement: Display an ASCII banner at the start of the...		Feature: Aesthetically pleasing terminal inte...	Improvement	M	26 Oct
JON-67	Improvement: Add text formatting and colour		Feature: Aesthetically pleasing terminal interface	Improvement	M	26 Oct

Application features

When setting the foundation of my implementation plan and the crucial features of the app, I thought about the must-have features that the app must have to run successfully. I thought about what the game would need and how it would flow if a person was playing Hi-Lo with a physical deck of cards and created features, the feature descriptions and checklists based on that.

From the beginning, I created several milestone or parent features, and slotted in every feature, improvement and error handling feature underneath each as a checklist of things to build.

Feature: Create deck of cards

- I began with this parent feature first as I knew that without a defined deck of cards, there would be no Hi-Lo game
- I knew that the deck would consist of number cards, numbered 0 to 10, and four operator cards — add, subtract, multiply and divide
- And of course, I needed to shuffle the deck once it was created, so each playthrough would be completely random and unpredictable, adding to the replayability

Feature: Deal cards to player

- Once the cards were created, the app would deal a random set of cards to the player

- The key here was that it was random, but also that the player would always receive 4 number cards and 3 operator cards to create a valid equation

Feature: Player chooses Hi (20) or Lo (1)

- Once the cards are dealt, the first input required from the player is whether they will choose to make an equation closest to 20 (Hi) or 1 (Lo)
- By defining these variables and by storing the player's choice, this would allow future features such as calculating the difference between the equation and this choice easier

Feature: Player can rearrange cards and input equation

- Now that the player has cards and a goal, I built a feature that allowed them to place down their cards in their desired order to create an equation
- This feature had the biggest checklist of things to build as it was the most intensive from a logic point of view and required the most checks on the player's input
- Given this was a terminal app, there was no opportunity to tap or drag and drop cards, so I built out a way that would allow the user to choose the order of their cards based on corresponding inputted numbers
- Along the way, I would discover there were many considerations for this feature, such as how to handle illogical moves and how to gracefully handle the input and output as the lengthiest part of the app
- Furthermore, I needed to build a way to allow the user to restart the placement of their cards, in the chance that the player incorrectly places the wrong card, has a change of mind or makes a move that would result in a dead end

Feature: Equation calculation

- Once the player has played all of their cards, the app calculates the outcome of their made equation
- When I wrote this feature into my plan at the beginning of the project, I thought this would be one of the easier features, as I knew Python already had some level of BODMAS or order of operations built in
- That said, this feature would prove to be difficult for a number of reasons, such as replacing my custom division symbols with the division symbol recognised by Python
- Also, I ran into an issue where my app was not correctly applying BODMAS, which I eventually resolved by using the built in eval() function, after searching for the right module to use

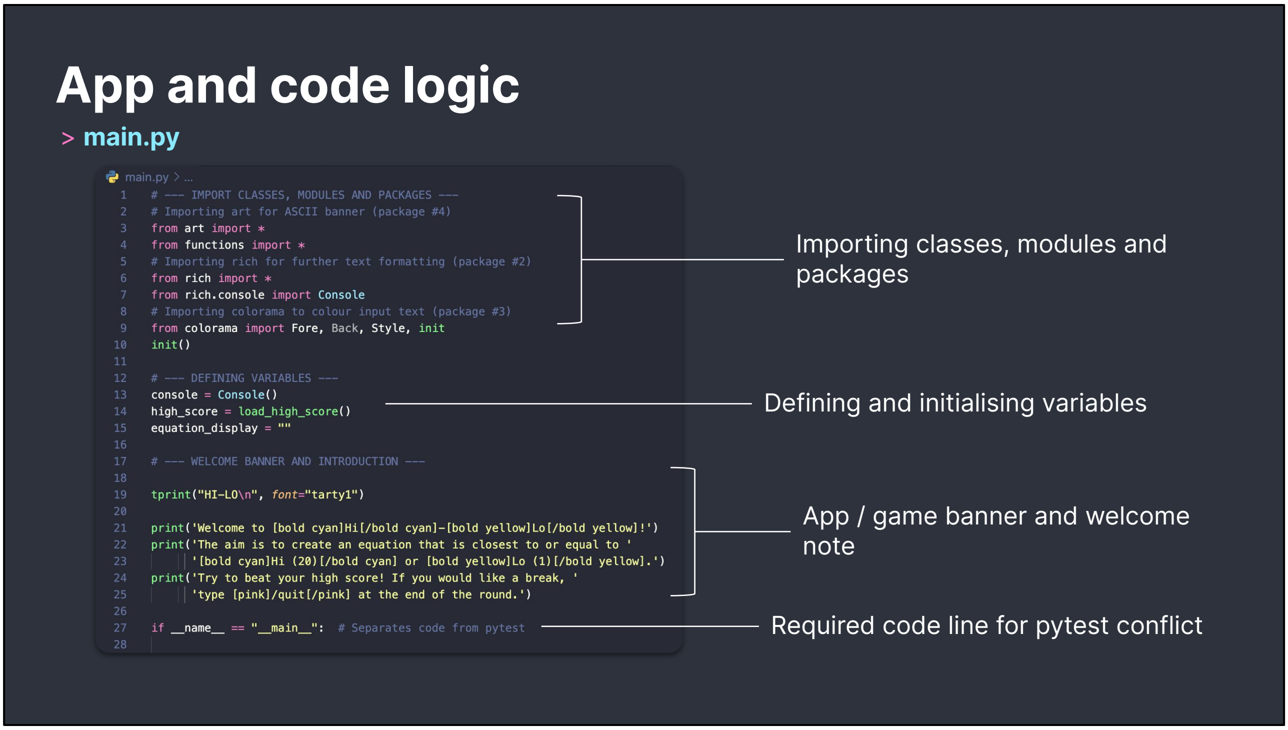
Feature: Display result and score

- Once the app calculated the equation correctly, it was time to display the player's result and score
- When my plan was created, I knew one of the first things I would need to figure out is how to appropriately scale the score given to the player
    - When I began, I still hadn't completely figured out how many rounds

    - the player would play or if I would just keep it to single rounds
    - As the build went on, I decided to keep it to a single round, as I figured it might not be right to lock a player into playing three rounds (for example) until they could get a score. This ensured that my game allowed for short and quick sessions.
- On this result screen, I built output to display how far off the player was from the target, their resulting score and the existing high score and if there was a new high score
- In order to keep the competitive side of things, I implemented a high score feature that would maintain the player's highest score achieved and would remain until exceeded
    - A consideration here is that the highest score that can be achieved is 100, whereas in a lot of games there are near endless amounts of high scores. This is something I can improve on in the future.
- Finally, I ensured there was a way to let the player play again or to take a break by typing /quit

Feature: Aesthetically pleasing terminal interface

- Given the world of interacting in terminals was still quite new to me, I was a little afraid of tackling this feature as I wasn't sure how I could achieve it in the given time. I was so used to seeing and using modern day GUIs in apps and browsers.
- However, there were a lot of learnings here from what could be achieved with what was built into Python, as well as the plethora of packages available on pypi
- While not as robust as a proper GUI, it was a great feature to learn from as I realised there's so much that can be tweaked and added aesthetically if time permits



App and code logic

> main.py

main.py > ...

```python
1   # --- IMPORT CLASSES, MODULES AND PACKAGES ---
2   # Importing art for ASCII banner (package #4)
3   from art import *
4   from functions import *
5   # Importing rich for further text formatting (package #2)
6   from rich import *
7   from rich.console import Console
8   # Importing colorama to colour input text (package #3)
9   from colorama import Fore, Back, Style, init
10  init()
11
12  # --- DEFINING VARIABLES ---
13  console = Console()
14  high_score = load_high_score()
15  equation_display = ""
16
17  # --- WELCOME BANNER AND INTRODUCTION ---
18
19  tprint("HI-LO\n", font="tarty1")
20
21  print('Welcome to [bold cyan]Hi[/bold cyan]-[bold yellow]Lo[/bold yellow]!')
22  print('The aim is to create an equation that is closest to or equal to '
23        '[bold cyan]Hi (20)[/bold cyan] or [bold yellow]Lo (1)[/bold yellow].')
24  print('Try to beat your high score! If you would like a break, '
25        'type [pink]/quit[/pink] at the end of the round.')
26
27  if __name__ == "__main__":  # Separates code from pytest
28
```

Importing classes, modules and packages

Defining and initialising variables

App / game banner and welcome note

Required code line for pytest conflict

App and code logic - main.py

Moving on to the explanation of the app's code and logic, starting with main.py, which houses the core logic of the app…

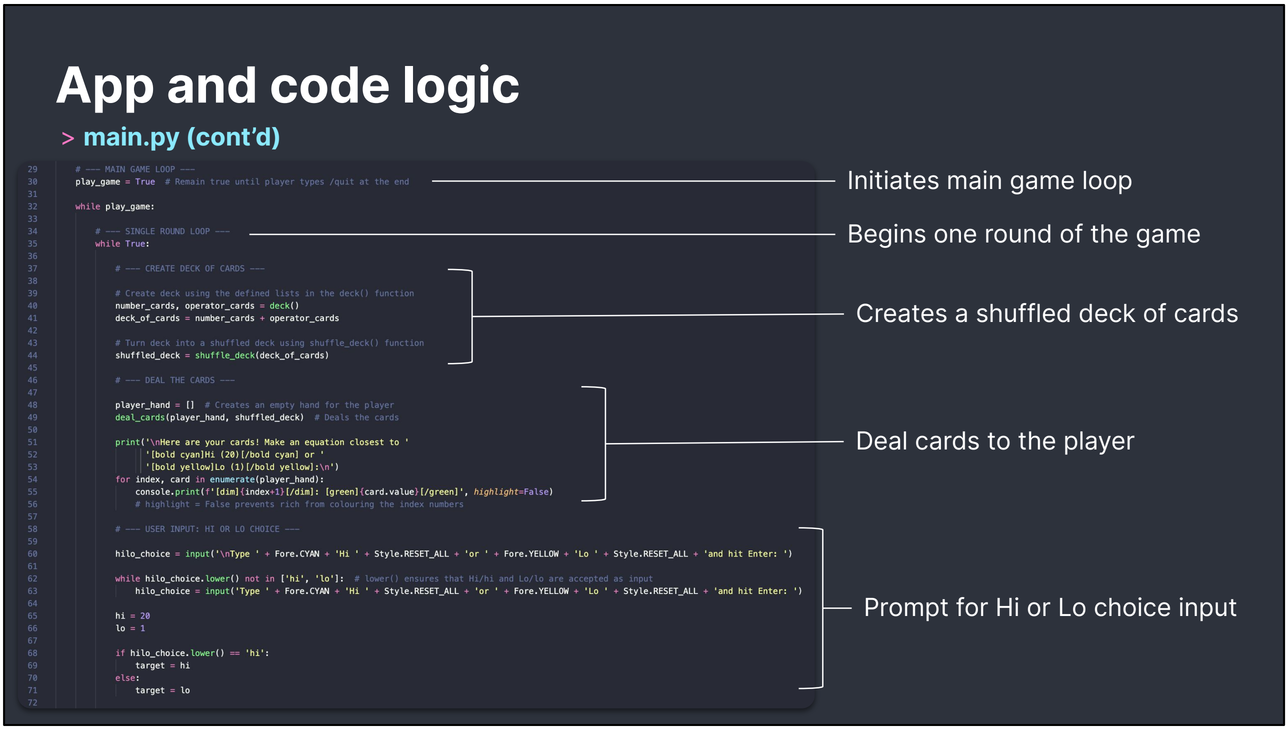
- Lines 1 to 10 involve importing all of the necessary **Python packages and modules** needed for the game, such as the core logic from functions.py.
    - The visual aspects of the app utilise the 'art', 'rich' and 'colorama' packages

- After importing the required modules, at lines 12 to 15, I define and **initialise the variables** required
    - The Console class is from the 'rich' package, which allows further customisation of text through console.Print()
    - The high_score variable keeps loads and keeps track of the ongoing high score for the player and
    - equation_display, while initially empty, is used to keep track of the current equation as the player places cards down

- Lines 17 to 25 are used to display the **app's banner and welcome notes**
    - Line 19 holds the welcome banner for Hi-Lo, using tprint() and the special font and ASCII art from the 'art' package
    - Lines 21 to 25 house a welcome message and explain the basic rules and have been separated to adhere to PEP8 character length

    - guidelines. Also, they contain syntax for colouring and styling certain words (e.g. [bold cyan] from the ‘rich’ package)

- Line 27 contains a line of code that allows the pytest to run without using `pytest -s`, ensuring that this code is only executed when the script is run directly (not imported as a module, such as in pytest).



App and code logic

> main.py (cont'd)

```
29      # --- MAIN GAME LOOP ---
30      play_game = True  # Remain true until player types /quit at the end
31
32      while play_game:
33
34          # --- SINGLE ROUND LOOP ---
35          while True:
36
37              # --- CREATE DECK OF CARDS ---
38
39              # Create deck using the defined lists in the deck() function
40              number_cards, operator_cards = deck()
41              deck_of_cards = number_cards + operator_cards
42
43              # Turn deck into a shuffled deck using shuffle_deck() function
44              shuffled_deck = shuffle_deck(deck_of_cards)
45
46              # --- DEAL THE CARDS ---
47
48              player_hand = []  # Creates an empty hand for the player
49              deal_cards(player_hand, shuffled_deck)  # Deals the cards
50
51              print('\nHere are your cards! Make an equation closest to '
52                    '[bold cyan]Hi (20)[/bold cyan] or '
53                    '[bold yellow]Lo (1)[/bold yellow]:\n')
54              for index, card in enumerate(player_hand):
55                  console.print(f'[dim]{index+1}[/dim]: [green]{card.value}[/green]', highlight=False)
56                  # highlight = False prevents rich from colouring the index numbers
57
58              # --- USER INPUT: HI OR LO CHOICE ---
59
60              hilo_choice = input('\nType ' + Fore.CYAN + 'Hi ' + Style.RESET_ALL + 'or ' + Fore.YELLOW + 'Lo ' + Style.RESET_ALL + 'and hit Enter: ')
61
62              while hilo_choice.lower() not in ['hi', 'lo']:  # lower() ensures that Hi/hi and Lo/lo are accepted as input
63                  hilo_choice = input('Type ' + Fore.CYAN + 'Hi ' + Style.RESET_ALL + 'or ' + Fore.YELLOW + 'Lo ' + Style.RESET_ALL + 'and hit Enter: ')
64
65              hi = 20
66              lo = 1
67
68              if hilo_choice.lower() == 'hi':
69                  target = hi
70              else:
71                  target = lo
72
```

Initiates main game loop

Begins one round of the game

Creates a shuffled deck of cards

Deal cards to the player

Prompt for Hi or Lo choice input

App and code logic - main.py (cont'd)

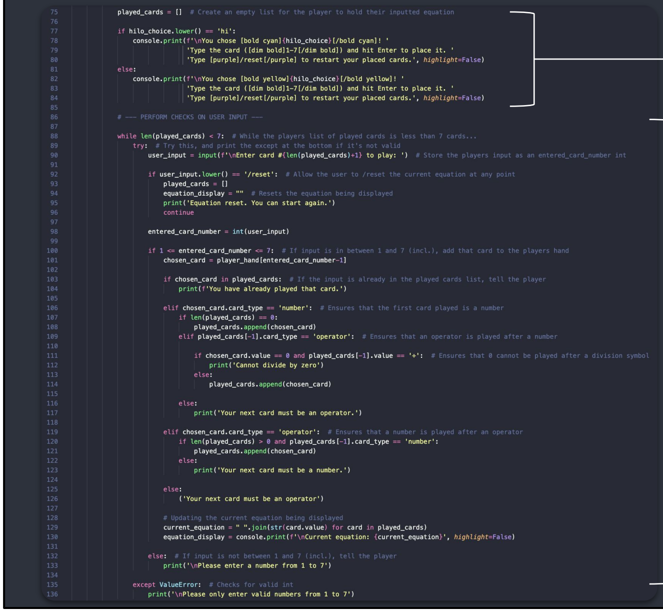
Following the welcome and introduction of the game, the main game loop is initiated through the code.

- At line 30, a **main game loop** was implemented to ensure that the player can continue playing multiple rounds of the game without completely exiting out of the app
    - This becomes more apparent at the end of each round when the player is given the choice to type /quit or try another round
- At line 35, for the initial launch of the game and for when the player decides to play another round, the **single round loop** of code is initiated
- Now nested within the game round loop, from lines 37 to 44 is when the **deck of cards is created and shuffled** to be randomised
    - The deck_of_cards variable ensures that it consists of the right number_cards and operator_cards
    - The code here utilises functions that are further defined in the functions.py file, such as the shuffled_deck() function used with Python's built in 'random' module
- Lines 46 to 56 were written to begin **dealing the cards** out to the player
    - The player hand is initiated with an empty list at first, and then deals the cards to the player hand from the shuffled deck
    - A for loop is used with enumerate() to iterate through each card in the player's hand and displays it to the player

- Once the player’s hand is displayed lines 58 to 71 are used to **prompt the user for the choice of Hi or Lo**
    - A short while loop is used to display the right type of prompt as I’m utilising different colours for the word ‘Hi’ and ‘Lo’ respectively
    - Variables for ‘hi’ and ‘lo’ are initiated with their values, and an if-else statement is used to ensure that the equation result target is changed to hi or lo respectively based on the player’s choice

App and code logic

> main.py (cont'd)



```
75                  played_cards = []  # Create an empty list for the player to hold their inputted equation
76  
77                  if hilo_choice.lower() == 'hi':
78                      console.print(f'\nYou chose [bold cyan]{hilo_choice}[/bold cyan]! '
79                                    'Type the card ([dim bold]1-7[/dim bold]) and hit Enter to place it. '
80                                    'Type [purple]/reset[/purple] to restart your placed cards.', highlight=False)
81                  else:
82                      console.print(f'\nYou chose [bold yellow]{hilo_choice}[/bold yellow]! '
83                                    'Type the card ([dim bold]1-7[/dim bold]) and hit Enter to place it. '
84                                    'Type [purple]/reset[/purple] to restart your placed cards.', highlight=False)
85  
86                  # --- PERFORM CHECKS ON USER INPUT ---
87  
88                  while len(played_cards) < 7:  # While the players list of played cards is less than 7 cards...
89                      try:  # Try this, and print the except at the bottom if it's not valid
90                          user_input = input(f'\nEnter card #{len(played_cards)+1} to play: ')  # Store the players input as an entered_card_number int
91  
92                          if user_input.lower() == '/reset':  # Allow the user to /reset the current equation at any point
93                              played_cards = []
94                              equation_display = ""  # Resets the equation being displayed
95                              print('Equation reset. You can start again.')
96                              continue
97  
98                          entered_card_number = int(user_input)
99  
100                         if 1 <= entered_card_number <= 7:  # If input is in between 1 and 7 (incl.), add that card to the players hand
101                             chosen_card = player_hand[entered_card_number-1]
102 
103                             if chosen_card in played_cards:  # If the input is already in the played cards list, tell the player
104                                 print(f'You have already played that card.')
105 
106                             elif chosen_card.card_type == 'number':  # Ensures that the first card played is a number
107                                 if len(played_cards) == 0:
108                                     played_cards.append(chosen_card)
109                                 elif played_cards[-1].card_type == 'operator':  # Ensures that an operator is played after a number
110 
111                                     if chosen_card.value == 0 and played_cards[-1].value == '÷':  # Ensures that 0 cannot be played after a division symbol
112                                         print('Cannot divide by zero')
113                                     else:
114                                         played_cards.append(chosen_card)
115 
116                                 else:
117                                     print('Your next card must be an operator.')
118 
119                             elif chosen_card.card_type == 'operator':  # Ensures that a number is played after an operator
120                                 if len(played_cards) > 0 and played_cards[-1].card_type == 'number':
121                                     played_cards.append(chosen_card)
122                                 else:
123                                     print('Your next card must be a number.')
124 
125                             else:
126                                 ('Your next card must be an operator')
127 
128                             # Updating the current equation being displayed
129                             current_equation = " ".join(str(card.value) for card in played_cards)
130                             equation_display = console.print(f'\nCurrent equation: {current_equation}', highlight=False)
131 
132                         else:  # If input is not between 1 and 7 (incl.), tell the player
133                             print('\nPlease enter a number from 1 to 7')
134 
135                     except ValueError:  # Checks for valid int
136                         print('\nPlease only enter valid numbers from 1 to 7')
```

Initiates list of played cards and confirm player Hi or Lo choice

Enables player to place cards and performs various checks on valid input

App and code logic - main.py (cont'd)

Once the player chooses Hi or Lo,

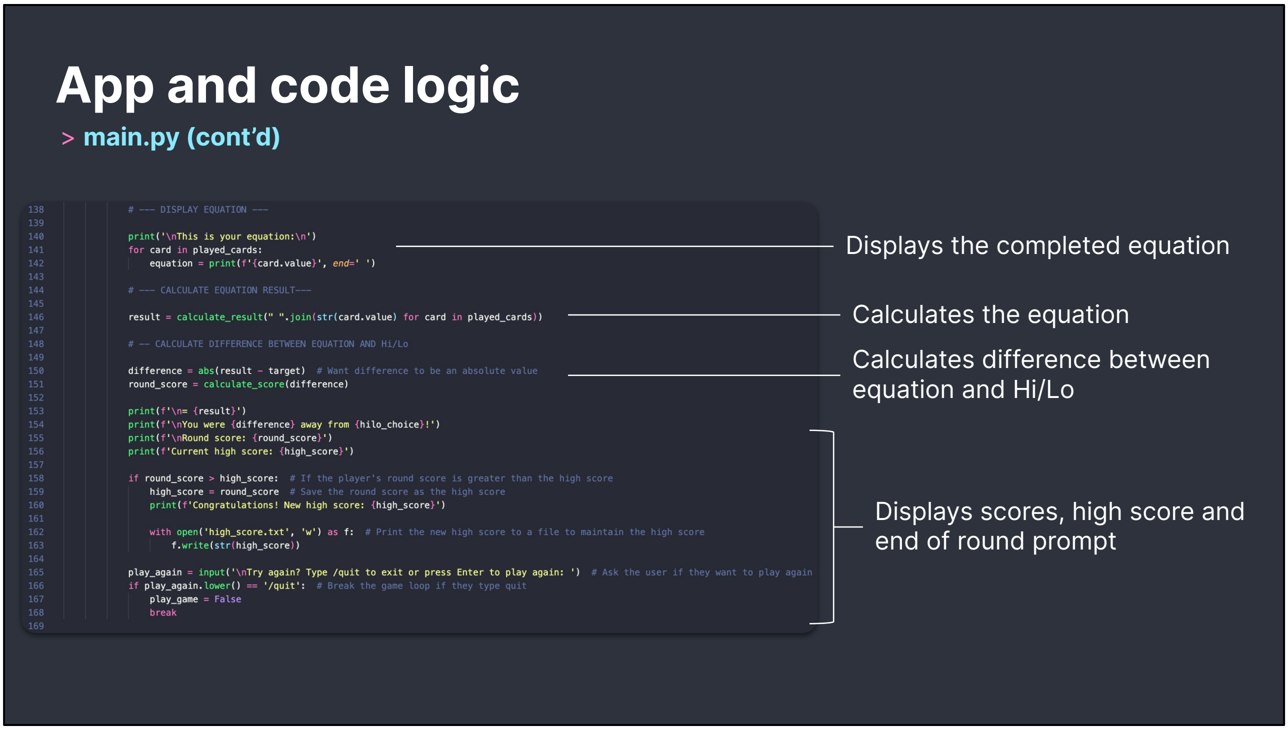
- In lines 75 to 84, the choice of Hi or Lo is confirmed and displayed to the player
- An empty played_cards list is also initiated to hold the cards that the player places down

Starting from line 88 begins the main portion of the game, where the user can begin placing down cards, while multiple checks are put in place to ensure that valid input is made

- The encapsulating while loop ensures that the player only plays 7 cards and begins allowing the user to start inputting the numbers that correspond to their hand of cards
- There are various checks in place throughout the code using if-elif-else statements, including:
    - Checking if the player inputs '/reset', which will allow the player to restart their played cards and start the equation again
    - Checking if the input is within the expected 1 to 7
    - Checking the value of the card played and the order, ensuring that
        - A number card is played first
        - A number is played after an operator and vice versa and
        - A zero is not played after a division operator
    - Checking if a card has already been played or not

- When the player breaks any of these coded rules, an error prompt is displayed

- Throughout this code, as the player plays cards, the evolving equation is displayed to ensure the player always knows what they’ve played so far



App and code logic

> main.py (cont’d)

```python
138             # --- DISPLAY EQUATION ---
139
140             print('\nThis is your equation:\n')
141             for card in played_cards:
142                 equation = print(f'{card.value}', end=' ')
143
144             # --- CALCULATE EQUATION RESULT---
145
146             result = calculate_result(" ".join(str(card.value) for card in played_cards))
147
148             # -- CALCULATE DIFFERENCE BETWEEN EQUATION AND Hi/Lo
149
150             difference = abs(result - target)  # Want difference to be an absolute value
151             round_score = calculate_score(difference)
152
153             print(f'\n= {result}')
154             print(f'\nYou were {difference} away from {hilo_choice}!')
155             print(f'\nRound score: {round_score}')
156             print(f'Current high score: {high_score}')
157
158             if round_score > high_score:  # If the player's round score is greater than the high score
159                 high_score = round_score  # Save the round score as the high score
160                 print(f'Congratulations! New high score: {high_score}')
161
162                 with open('high_score.txt', 'w') as f:  # Print the new high score to a file to maintain the high score
163                     f.write(str(high_score))
164
165             play_again = input('\nTry again? Type /quit to exit or press Enter to play again: ')  # Ask the user if they want to play again
166             if play_again.lower() == '/quit':  # Break the game loop if they type quit
167                 play_game = False
168                 break
169
```

Displays the completed equation

Calculates the equation

Calculates difference between equation and Hi/Lo

Displays scores, high score and end of round prompt

App and code logic - main.py (cont’d)

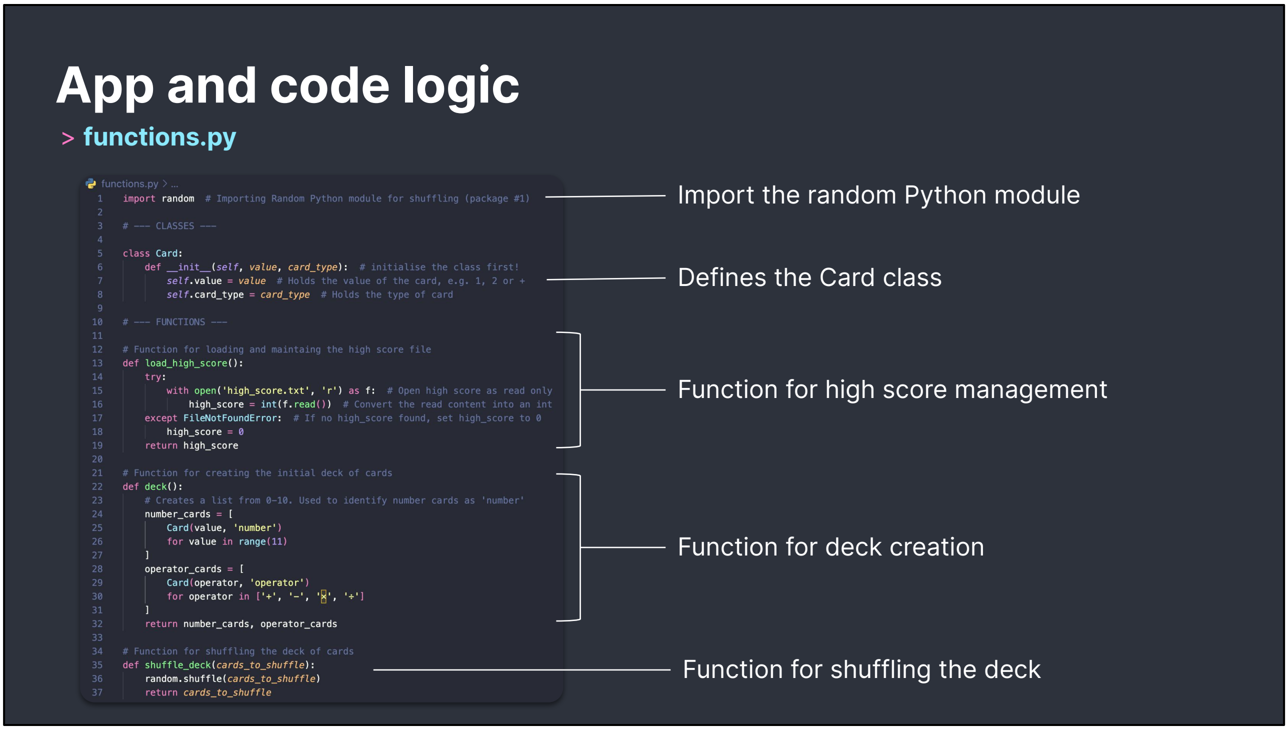
Once the player successfully creates a valid equation,

- Lines 140 to 142 display their created equation by iterating through the played_cards list
- end=’ ‘ is used to print the equation on one line, rather than on separate lines

- The equation result is calculated in line 146, where it calls the calculate_result() function to evaluate the equation
- Within the function, a string representation of the played cards is made

- Lines 150 and 151 compute the absolute difference between the calculated result and the target value, which is either Hi or Lo
- Based on this difference, a round score is given, where the score is defined in functions.py through calculate_score()

- The remaining code in main.py is used to display the player’s results, their score, the current high score and it will display a congratulatory note if a new high score is achieved
- Importantly, a prompt to ask the player if they want to play again or take a break with /quit is used and depending on their choice, the game will loop through once more or will terminate



App and code logic

> functions.py

functions.py > ...

```
1   import random  # Importing Random Python module for shuffling (package #1)
2   
3   # --- CLASSES ---
4   
5   class Card:
6       def __init__(self, value, card_type):  # initialise the class first!
7           self.value = value  # Holds the value of the card, e.g. 1, 2 or +
8           self.card_type = card_type  # Holds the type of card
9   
10  # --- FUNCTIONS ---
11  
12  # Function for loading and maintaing the high score file
13  def load_high_score():
14      try:
15          with open('high_score.txt', 'r') as f:  # Open high score as read only
16              high_score = int(f.read())  # Convert the read content into an int
17      except FileNotFoundError:  # If no high_score found, set high_score to 0
18          high_score = 0
19      return high_score
20  
21  # Function for creating the initial deck of cards
22  def deck():
23      # Creates a list from 0-10. Used to identify number cards as 'number'
24      number_cards = [
25          Card(value, 'number')
26          for value in range(11)
27      ]
28      operator_cards = [
29          Card(operator, 'operator')
30          for operator in ['+', '-', '×', '÷']
31      ]
32      return number_cards, operator_cards
33  
34  # Function for shuffling the deck of cards
35  def shuffle_deck(cards_to_shuffle):
36      random.shuffle(cards_to_shuffle)
37      return cards_to_shuffle
```

Import the random Python module

Defines the Card class

Function for high score management

Function for deck creation

Function for shuffling the deck

App and code logic - functions.py

Separate to the main.py file, a functions.py file was created to separate the created classes and functions to handle much of the background logic of the game and to organise the code cleanly.

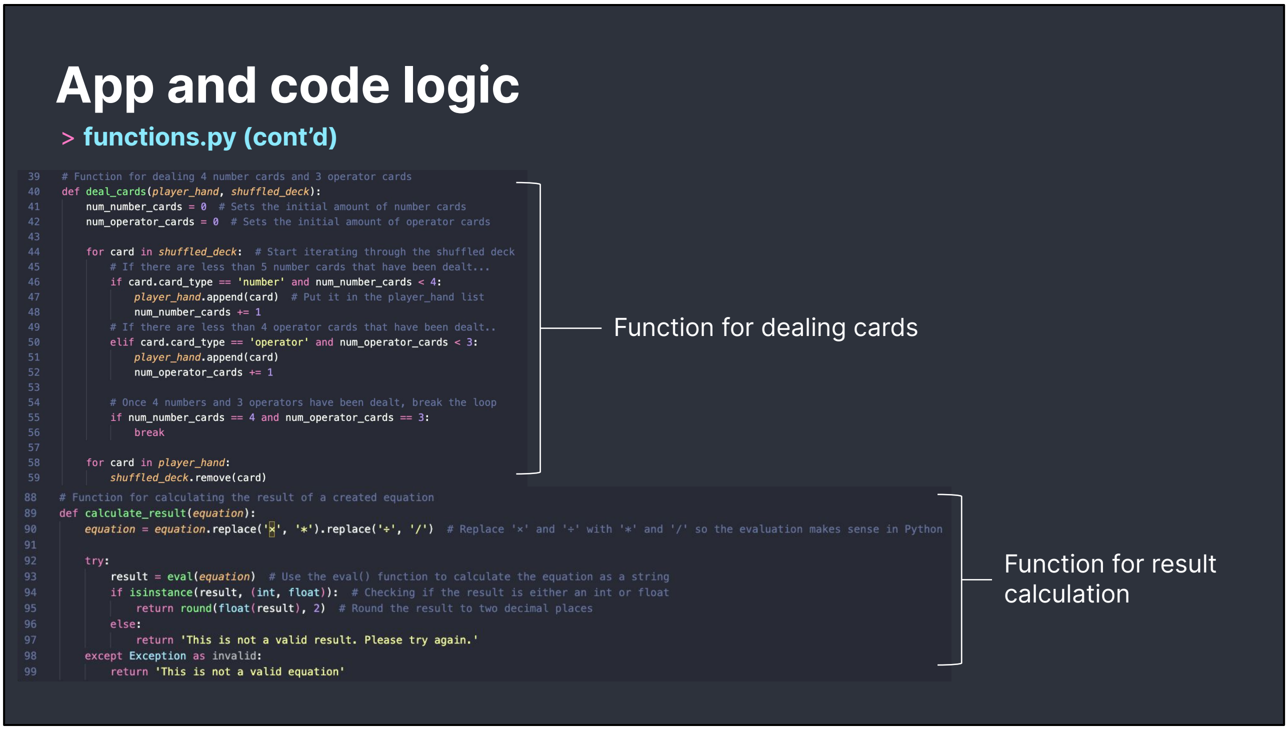
- In line 1, I initially import the random Python module, which is necessary for shuffling the deck and ensuring each round is randomised
- The Card class is then defined at line 5, which represents each individual card with a value and a type

- All of the functions are then defined underneath, starting with the load_high_score() function at line 13, which loads and maintains the game's high score
- This code utilises Python's way of handling files, by opening the high_score.txt file as a read-only file, and stores the content as the high_score
- There is also logic for when a high score is not found

- At line 22, the deck() function creates the deck of cards with a defined set of number cards and operator cards
- The numbered cards include 0 to 10 using range(11) and the operator cards are defined with the addition, subtraction, multiplication and division symbols

- At line 34, shuffle_deck() is defined in order to utilise the random Python

- package that will ensure the cards are dealt in true shuffled fashion



App and code logic

> functions.py (cont'd)

```
39  # Function for dealing 4 number cards and 3 operator cards
40  def deal_cards(player_hand, shuffled_deck):
41      num_number_cards = 0  # Sets the initial amount of number cards
42      num_operator_cards = 0  # Sets the initial amount of operator cards
43
44      for card in shuffled_deck:  # Start iterating through the shuffled deck
45          # If there are less than 5 number cards that have been dealt...
46          if card.card_type == 'number' and num_number_cards < 4:
47              player_hand.append(card)  # Put it in the player_hand list
48              num_number_cards += 1
49          # If there are less than 4 operator cards that have been dealt..
50          elif card.card_type == 'operator' and num_operator_cards < 3:
51              player_hand.append(card)
52              num_operator_cards += 1
53
54          # Once 4 numbers and 3 operators have been dealt, break the loop
55          if num_number_cards == 4 and num_operator_cards == 3:
56              break
57
58      for card in player_hand:
59          shuffled_deck.remove(card)
```

Function for dealing cards

```
88  # Function for calculating the result of a created equation
89  def calculate_result(equation):
90      equation = equation.replace('×', '*').replace('÷', '/')  # Replace '×' and '÷' with '*' and '/' so the evaluation makes sense in Python
91
92      try:
93          result = eval(equation)  # Use the eval() function to calculate the equation as a string
94          if isinstance(result, (int, float)):  # Checking if the result is either an int or float
95              return round(float(result), 2)  # Round the result to two decimal places
96          else:
97              return 'This is not a valid result. Please try again.'
98      except Exception as invalid:
99          return 'This is not a valid equation'
```

Function for result calculation

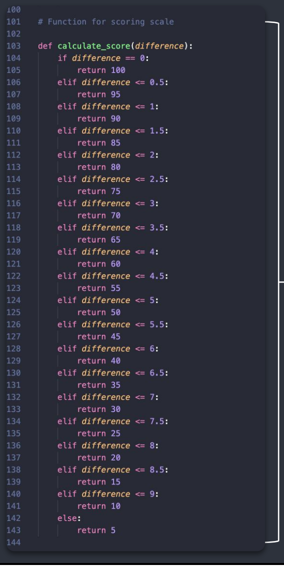
App and code logic - functions.py (cont'd)

- At line 40, the deal_cards() function is responsible for dealing the cards to the player
- Within this function, the number of number and operator cards is set at 0, and tracked in a for loop to ensure that the player only receives the required amount of cards based on their type
- Once the player receives 4 number cards and 3 operator cards, the loop is broken and the cards are also removed from the deck

- At line 89, a function called calculate_result() is defined to assist in calculating the equation created by the player
- Within this function, I built out a way to replace the more custom multiplication and division symbols with the computable * and / symbols respectively so that Python could understand it
- The eval() built in function is used to calculate the created equation as a string, as well as apply BODMAS order of operations to the equation
- isinstance() is used to further verify that the result is either an integer or float data type, and round() is used to avoid results that have too many decimal places for the output
- Finally, an except (along with try:) is used to capture all other cases where the result is not a valid equation

App and code logic

> functions.py (cont'd)



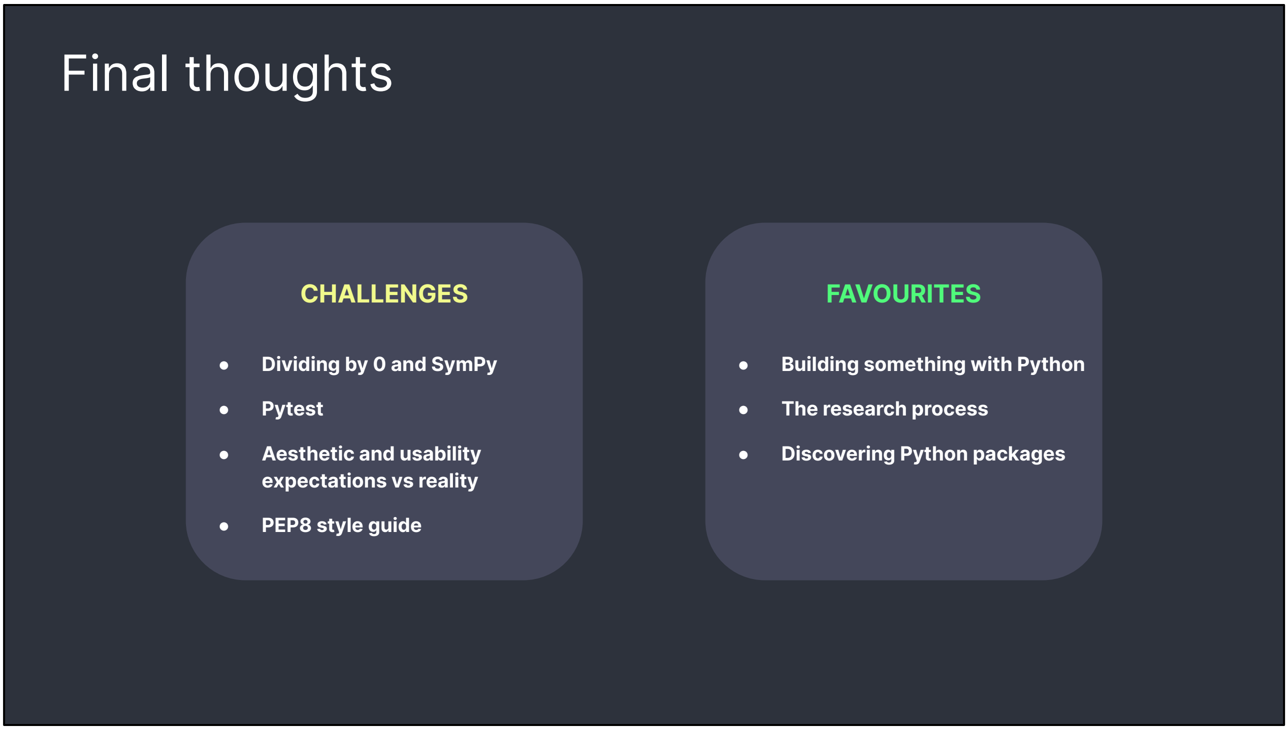
```python
# Function for scoring scale

def calculate_score(difference):
    if difference == 0:
        return 100
    elif difference <= 0.5:
        return 95
    elif difference <= 1:
        return 90
    elif difference <= 1.5:
        return 85
    elif difference <= 2:
        return 80
    elif difference <= 2.5:
        return 75
    elif difference <= 3:
        return 70
    elif difference <= 3.5:
        return 65
    elif difference <= 4:
        return 60
    elif difference <= 4.5:
        return 55
    elif difference <= 5:
        return 50
    elif difference <= 5.5:
        return 45
    elif difference <= 6:
        return 40
    elif difference <= 6.5:
        return 35
    elif difference <= 7:
        return 30
    elif difference <= 7.5:
        return 25
    elif difference <= 8:
        return 20
    elif difference <= 8.5:
        return 15
    elif difference <= 9:
        return 10
    else:
        return 5
```

Function for scoring scale

App and code logic - functions.py (cont'd)

- The last function defined in the functions.py file is the calculate_scor() function, which is used to calculate the player's round score based
- The logic is based on how close the player gets to making their equation equal the chosen target, being Hi or Lo
- There are multiple if-elif statements that return the score based on the difference that the player achieves, with 100 being the highest score when the player makes an equation that is exactly on the target (20 or 1)
- The score gradually decreases as the player gets further away from the target, with a minimum score of 5 being returned if the player gets more than 9 away from the target



Final thoughts

CHALLENGES

- Dividing by 0 and SymPy
- Pytest
- Aesthetic and usability expectations vs reality
- PEP8 style guide

FAVOURITES

- Building something with Python
- The research process
- Discovering Python packages

Finally, to end on some final thoughts and a review of the **Challenges** I faced and my **Favourite parts** of the entire process of building this application.

Starting with the **Challenges**

- I initially set out with the idea that I should let the player **divide by zero**, even though I was aware that this was not mathematically acceptable
- The idea behind this was to give the player more options, but as I began building this feature, I realised this was going to be a very troublesome task as this goes against all Python logic
- I tried multiple methods, including using Python packages such as SymPy. However, even when managing to replace / 0 with 0, the logic would eventually fail when it comes to following BODMAS. As such, after many hours spent trying to create a workaround, I decided to park this feature and not allow division by zero, thus creating another error handling feature to not allow the player to place a zero card after a division card.

- Using **Pytest** initially proved to be a challenge, as I could not get my tests to work, even if I knew that they should logically pass
- When I ran my first two tests, I kept getting the following error:
  “ERROR test_game.py - OSError: pytest: reading from stdin while output is captured! Consider using `-s`.”
- I found it difficult to wrap my head around why this was occurring, even after

- searching online for the reason, and eventually I simplified my tests to adhere to it.
- However, after speaking to a couple of other students in the cohort who were receiving the same error, I realised that it's an issue when Pytest tries to import the module which has input required.
- So instead of using `pytest -s`, we figured out that we could insert "if __name__ == "__main__":" to separate the code when running through Pytest, allowing it to pass the tests as expected

- Compared to building a website in a previous assignment, I found this terminal application assignment challenging from an **aesthetic and usability** point of view
- As terminal applications strip a lot of the ability to drag and drop or tap actions and nice, responsive user interfaces that modern apps are used to, I found it challenging to create an app that met my expectations
- For example, I would've liked to make my cards actually look more like cards, but creating this through ASCII art proved to be difficult.
- While I looked into the possibilities with Python packages, such as 'art', there were complications around creating art for the uncommon division symbol (÷) and lining up the cards side-by-side, instead of print line by line
- Also, the feature that allows players to place their cards isn't the most ideal in my mind, but it worked and I'm certain it would have been a great foundation for the future where players could drag and drop cards down and rearrange them dynamically, just like that would with a real set of cards

- Finally, a minor challenge that I faced was adhering closely to the PEP8 style guide
- While it started out as tedious and proved to be very meticulous and specific, it taught me a lot about how important it is to keep my code in line with style guides, especially as others were going to read my code, and in the future, hopefully would be the standards I would need to follow in the industry

As for some of my **Favourite** parts of the build,

- A real challenge that turned into one of my favourite parts of the assignment and a great learning opportunity, was simply writing in **Python**
- Compared to the previous assignment where I had used some HTML and CSS in the past, Python has been a completely new experience for me, beginning with Coder Academy

- It was daunting to get started and I hit several roadblocks that were gaps in my knowledge along the way, but these turned into excellent opportunities to

- **research and read** — a lot of which was confusing and beyond what was taught so far, but really beneficial in the long run I’m sure.
- Knowing how prominent and sought after Python is in the tech world, it was a great feeling to get my hands dirty with Python and build something of my own from start to finish

- Finally, it was really great to see how supported and loved Python is as a programming language through discovering the endless amounts of packages that are built in and built by the community
- Given the time, it seems it’s possible to really customise terminal applications through really thoughtful and innovative packages, and finding out about just a handful such as ‘art’ and ‘colorama’ and how customisable they are was an experience

Overall and in conclusion, this assignment and build was the toughest project so far. On one hand, I wish I could make something greater, but I’m still proud of what I have built in a week, especially as my first Python experience ever.

Thank you!

> Jonathan Phey
https://github.com/jjjjjpppppp/t1a3-hi-lo

Thank you for reading and please reach out if you have any questions about my application, my slides or what I’ve presented.