# T1A3- Terminal Application

HANGMAN

Mitchell Fernandez 25/10/2023

# Introduction

## My Hangman game application

My Terminal application assignment I decided to make a simple game of hangman.

Coded in python and run in Terminal my application uses logic and algorithmic thinking along with ASCII Art to make a fun but simple game of hangman.

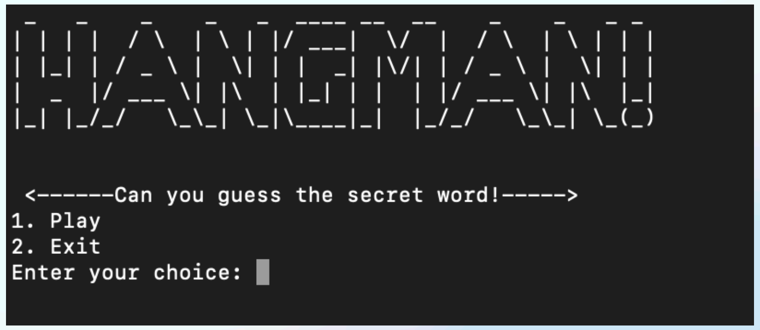
This application was a lot of fun to make and I believe I learnt a lot along the way.

Let's take a look!!

## Features

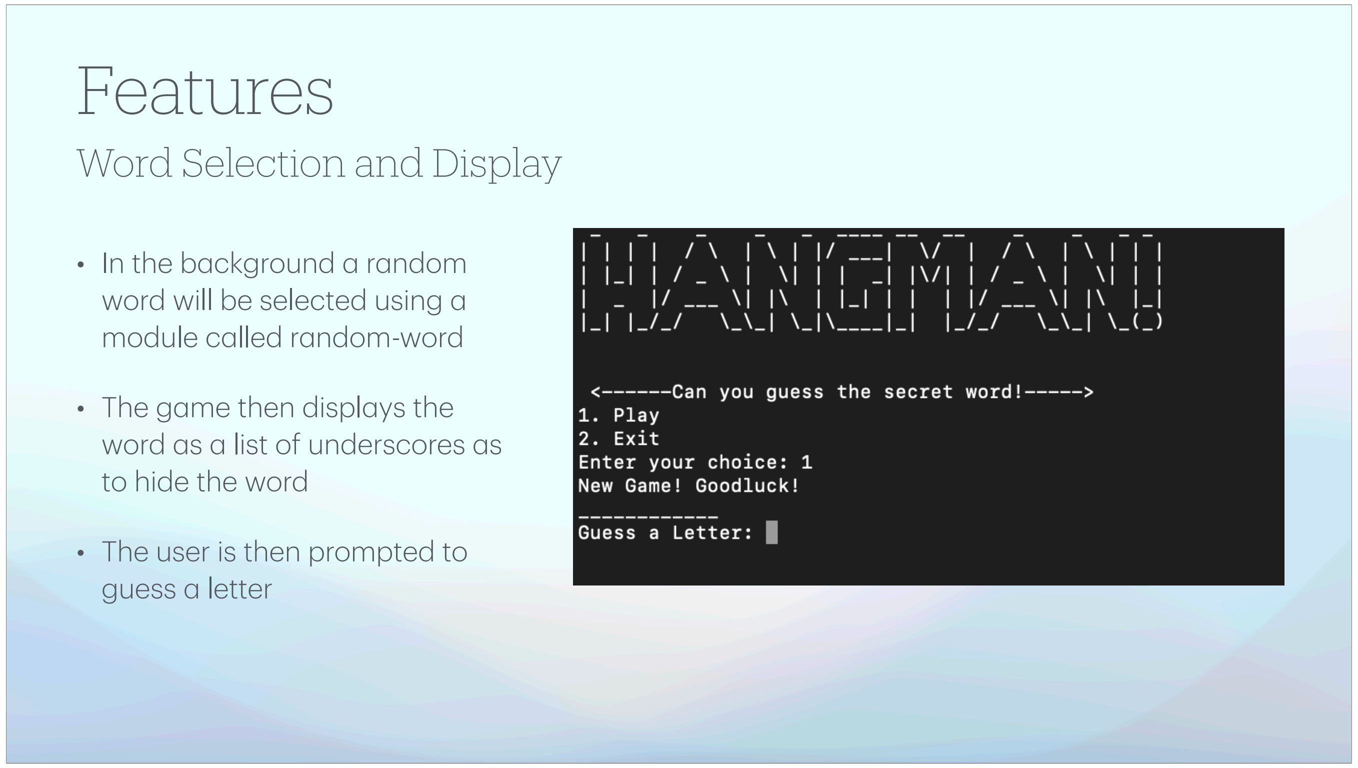
### Main Menu

- A simple Main Menu that displays when game is launched
- Gives two options, Play game or Exit
- User can select 1 or 2 and
- 1 will initiate the game
- 2 will exit the program



```
 _   _    _    _   _  ____ __  __    _    _   _ _ 
| | | |  / \  | \ | |/ ___|  \/  |  / \  | \ | | |
| |_| | / _ \ |  \| | |  _| |\/| | / _ \ |  \| | |
|  _  |/ ___ \| |\  | |_| | |  | |/ ___ \| |\  |_|
|_| |_/_/   \_\_| \_|\____|_|  |_/_/   \_\_| \_(_)


 <------Can you guess the secret word!----->
1. Play
2. Exit
Enter your choice: 
```

Initially the main menu was not in my design plan and the thought came to me as I was progressing through my implementation plan. I am glad the thought came to me as it is a nice feature for user experience. The process is simple select 1 to play or 2 to exit and it will initiate the appropriate screens.



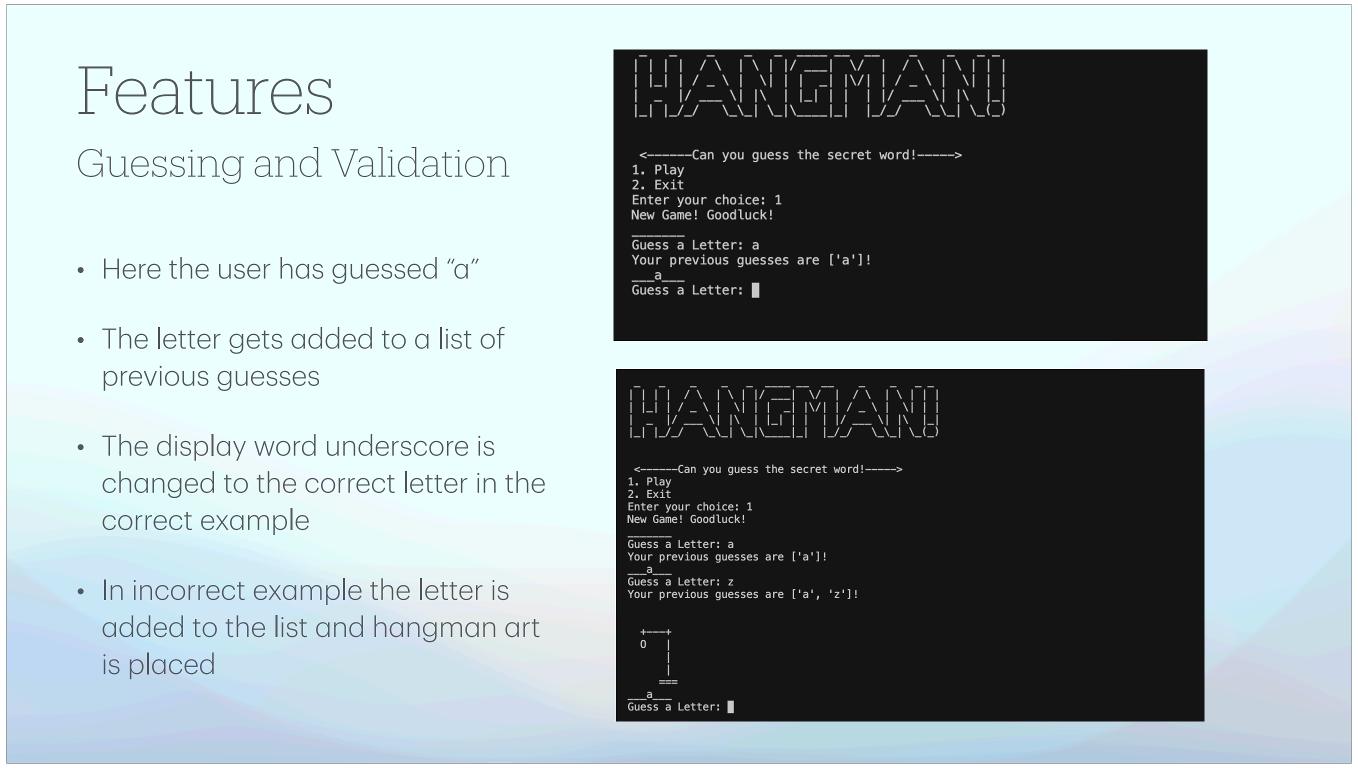
## Features

### Word Selection and Display

- In the background a random word will be selected using a module called random-word
- The game then displays the word as a list of underscores as to hide the word
- The user is then prompted to guess a letter

```
 _   _    _    _   _  ____ __  __    _    _   _ _ 
| | | |  / \  | \ | |/ ___|  \/  |  / \  | \ | | |
| |_| | / _ \ |  \| | |  _| |\/| | / _ \ |  \| | |
|  _  |/ ___ \| |\  | |_| | |  | |/ ___ \| |\  |_|
|_| |_/_/   \_\_| \_|\____|_|  |_/_/   \_\_| \_(_)


 <------Can you guess the secret word!----->
1. Play
2. Exit
Enter your choice: 1
New Game! Goodluck!
____________
Guess a Letter: 
```

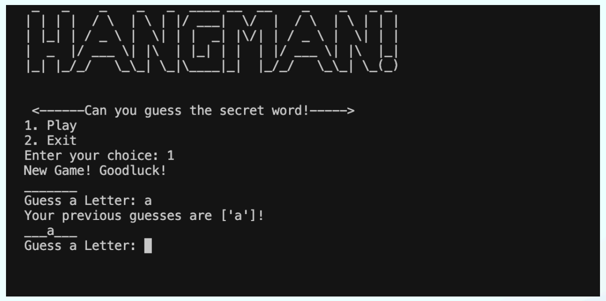
In the background a lot is happening here, first the game is initiating the game then it is selecting a random word from an imported module called random-word, it then manipulates the string to display as underscores. It then asks the user to guess a letter. Only letters can be input and it does not matter if the letters are lowercase or upper case both will be considered.



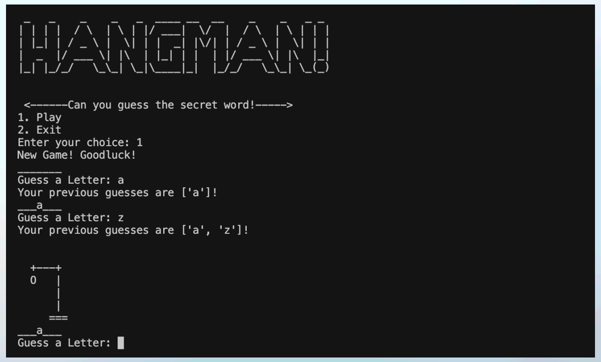
## Features

### Guessing and Validation

- Here the user has guessed “a”
- The letter gets added to a list of previous guesses
- The display word underscore is changed to the correct letter in the correct example
- In incorrect example the letter is added to the list and hangman art is placed



```
 _   _    _    _   _  ____ __  __    _    _   _ _ 
| | | |  / \  | \ | |/ ___|  \/  |  / \  | \ | | |
| |_| | / _ \ |  \| | |  _| |\/| | / _ \ |  \| | |
|  _  |/ ___ \| |\  | |_| | |  | |/ ___ \| |\  |_|
|_| |_/_/   \_\_| \_|\____|_|  |_/_/   \_\_| \_(_)


 <------Can you guess the secret word!----->
1. Play
2. Exit
Enter your choice: 1
New Game! Goodluck!
_______
Guess a Letter: a
Your previous guesses are ['a']!
___a___
Guess a Letter: 
```



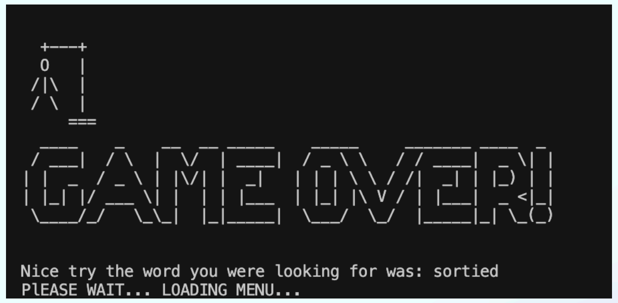
```
 _   _    _    _   _  ____ __  __    _    _   _ _ 
| | | |  / \  | \ | |/ ___|  \/  |  / \  | \ | | |
| |_| | / _ \ |  \| | |  _| |\/| | / _ \ |  \| | |
|  _  |/ ___ \| |\  | |_| | |  | |/ ___ \| |\  |_|
|_| |_/_/   \_\_| \_|\____|_|  |_/_/   \_\_| \_(_)


 <------Can you guess the secret word!----->
1. Play
2. Exit
Enter your choice: 1
New Game! Goodluck!
_______
Guess a Letter: a
Your previous guesses are ['a']!
___a___
Guess a Letter: z
Your previous guesses are ['a', 'z']!


  +---+
  O   |
      |
      |
     ===
___a___
Guess a Letter: 
```

This is the main feature of the game checking against functions if the guesses are valid or invalid and displaying the correct outputs. We will learn more about how it works in our code walkthrough.

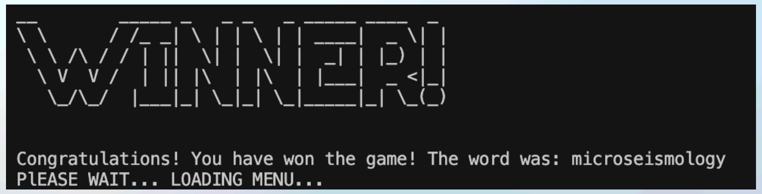
## Features

### Winning and Losing conditions

- The game ends when Hangman is displayed in full
- The player has reached maximum guesses and game over message is displayed
- When the user can guess all the letters in a word the Winner message is displayed.
- As I said the words are completely random!!

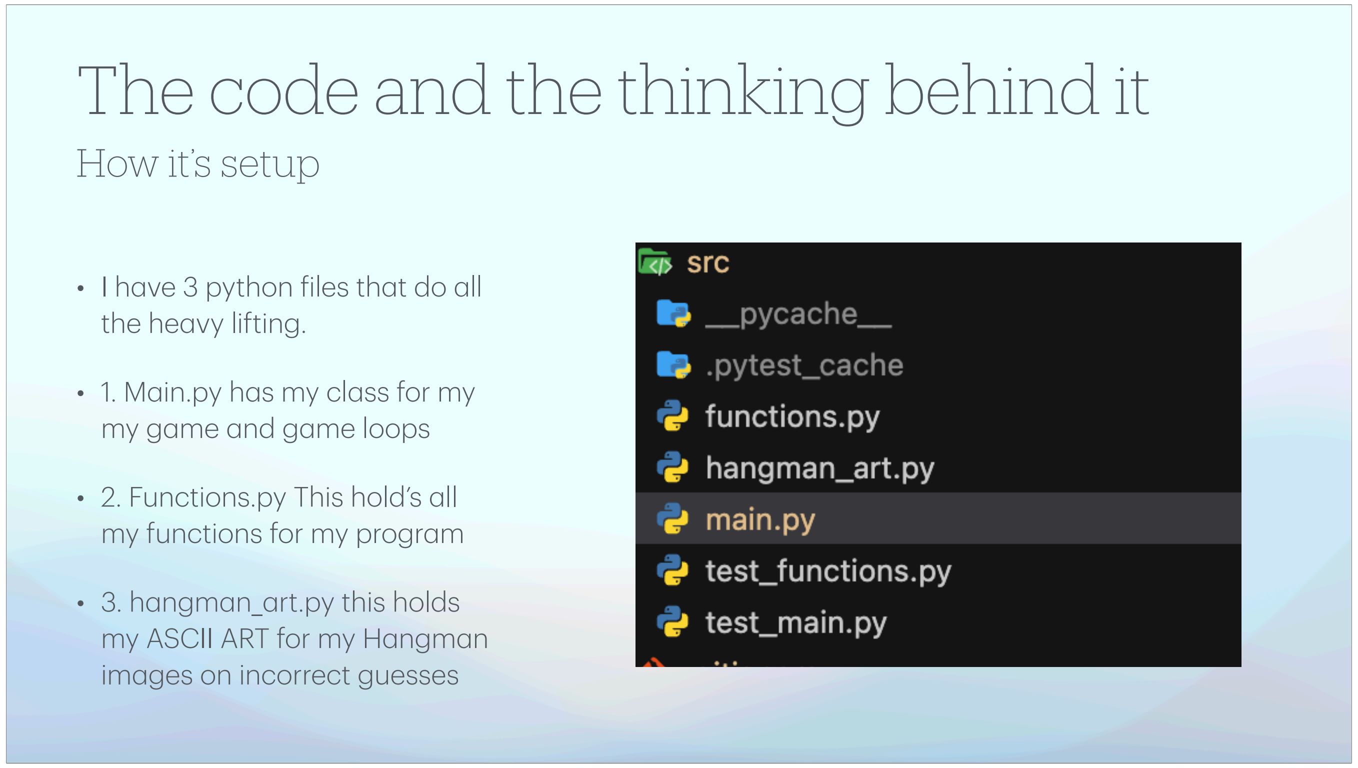


```
  +---+
  O   |
 /|\  |
 / \  |
     ===

  ____    _    __  __ _____    _____     _______ ____  _
 / ___|  / \  |  \/  | ____|  / _ \ \   / / ____|  _ \| |
| |  _  / _ \ | |\/| |  _|   | | | \ \ / /|  _| | |_) | |
| |_| |/ ___ \| |  | | |___  | |_| |\ V / | |___|  _ <|_|
 \____/_/   \_\_|  |_|_____|  \___/  \_/  |_____|_| \_(_)


Nice try the word you were looking for was: sortied
PlEASE WAIT... LOADING MENU...
```



```
__        _____ _   _ _   _ _____ ____  _
\ \      / /_ _| \ | | \ | | ____|  _ \| |
 \ \ /\ / / | ||  \| |  \| |  _| | |_) | |
  \ V  V /  | || |\  | |\  | |___|  _ <|_|
   \_/\_/  |___|_| \_|_| \_|_____|_| \_(_)


Congratulations! You have won the game! The word was: microseismology
PlEASE WAIT... LOADING MENU...
```

As explained the goal of hangman is to guess the word, every guess will display in the list, only correct guesses will alter the underscores. Once you display hangman its game over! If you guess all the correct letters its a win! Both outcomes will result in a clear screen and reload the main menu where you can select to play again or to exit. Now lets take a look at some logic and …the fun stuff… THE CODE!



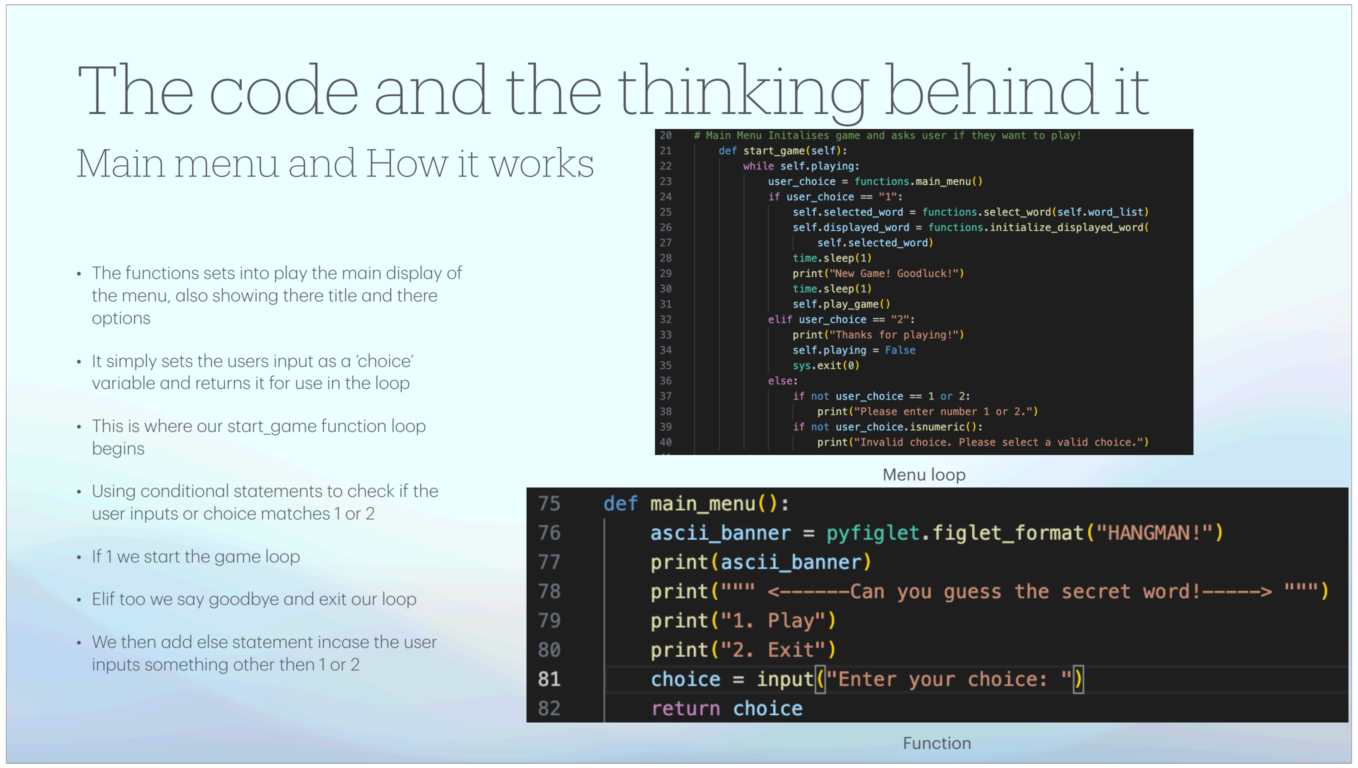
## The code and the thinking behind it

How it's setup

- I have 3 python files that do all the heavy lifting.
- 1. Main.py has my class for my my game and game loops
- 2. Functions.py This hold's all my functions for my program
- 3. hangman_art.py this holds my ASCII ART for my Hangman images on incorrect guesses

```
src
  __pycache__
  .pytest_cache
  functions.py
  hangman_art.py
  main.py
  test_functions.py
  test_main.py
```

At first I started my program with one main file. This quickly became cluttered and when running tests I ran into issues. This taught me the great benefits of separating and organizing your code more appropriately. A file for functions all in one place easy to read and easy to import into main. A file for ART as this gets long and really takes up many lines of code so nice to keep it tucked away. And finally the main file my heavy lifter running my loops and initializing my game. I think this is a great separation for the size of my project. There are also separate test files etc but we will get to that later.



# The code and the thinking behind it

## Main menu and How it works

- The functions sets into play the main display of the menu, also showing there title and there options
- It simply sets the users input as a 'choice' variable and returns it for use in the loop
- This is where our start_game function loop begins
- Using conditional statements to check if the user inputs or choice matches 1 or 2
- If 1 we start the game loop
- Elif too we say goodbye and exit our loop
- We then add else statement incase the user inputs something other then 1 or 2

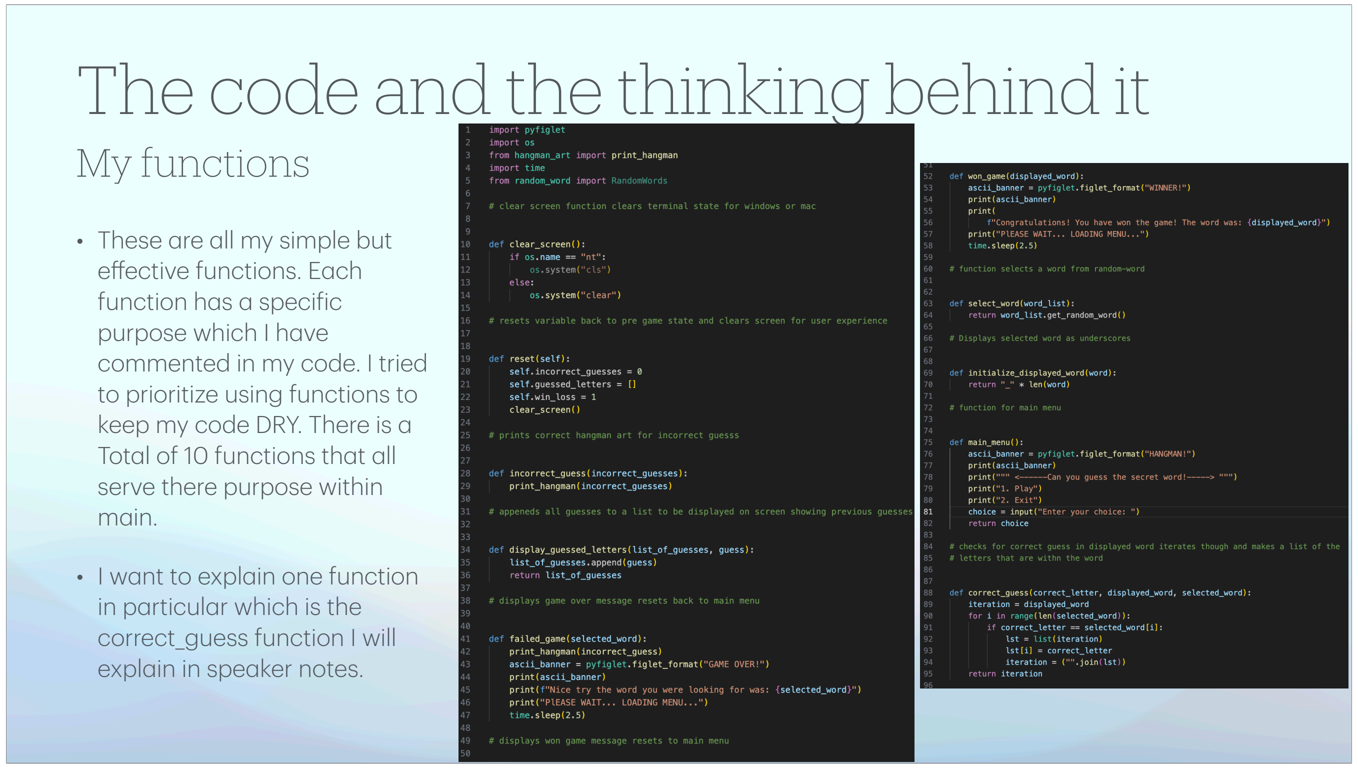
```python
# Main Menu Initalises game and asks user if they want to play!
    def start_game(self):
        while self.playing:
            user_choice = functions.main_menu()
            if user_choice == "1":
                self.selected_word = functions.select_word(self.word_list)
                self.displayed_word = functions.initialize_displayed_word(
                    self.selected_word)
                time.sleep(1)
                print("New Game! Goodluck!")
                time.sleep(1)
                self.play_game()
            elif user_choice == "2":
                print("Thanks for playing!")
                self.playing = False
                sys.exit(0)
            else:
                if not user_choice == 1 or 2:
                    print("Please enter number 1 or 2.")
                if not user_choice.isnumeric():
                    print("Invalid choice. Please select a valid choice.")
```

Menu loop

```python
def main_menu():
    ascii_banner = pyfiglet.figlet_format("HANGMAN!")
    print(ascii_banner)
    print(""" <------Can you guess the secret word!-----> """)
    print("1. Play")
    print("2. Exit")
    choice = input("Enter your choice: ")
    return choice
```

Function

Simply using a function and a while loop with some conditional statements I was able to create the main menu of my game. A while loop was the obvious choice as we wanted to continue only if a condition was met this also let us error handle user input nicely. The main menu loop calls upon many functions including getting the selected word and displaying it after stating new game! I used functions for everything which helps me define what attributes do what things.



## The code and the thinking behind it

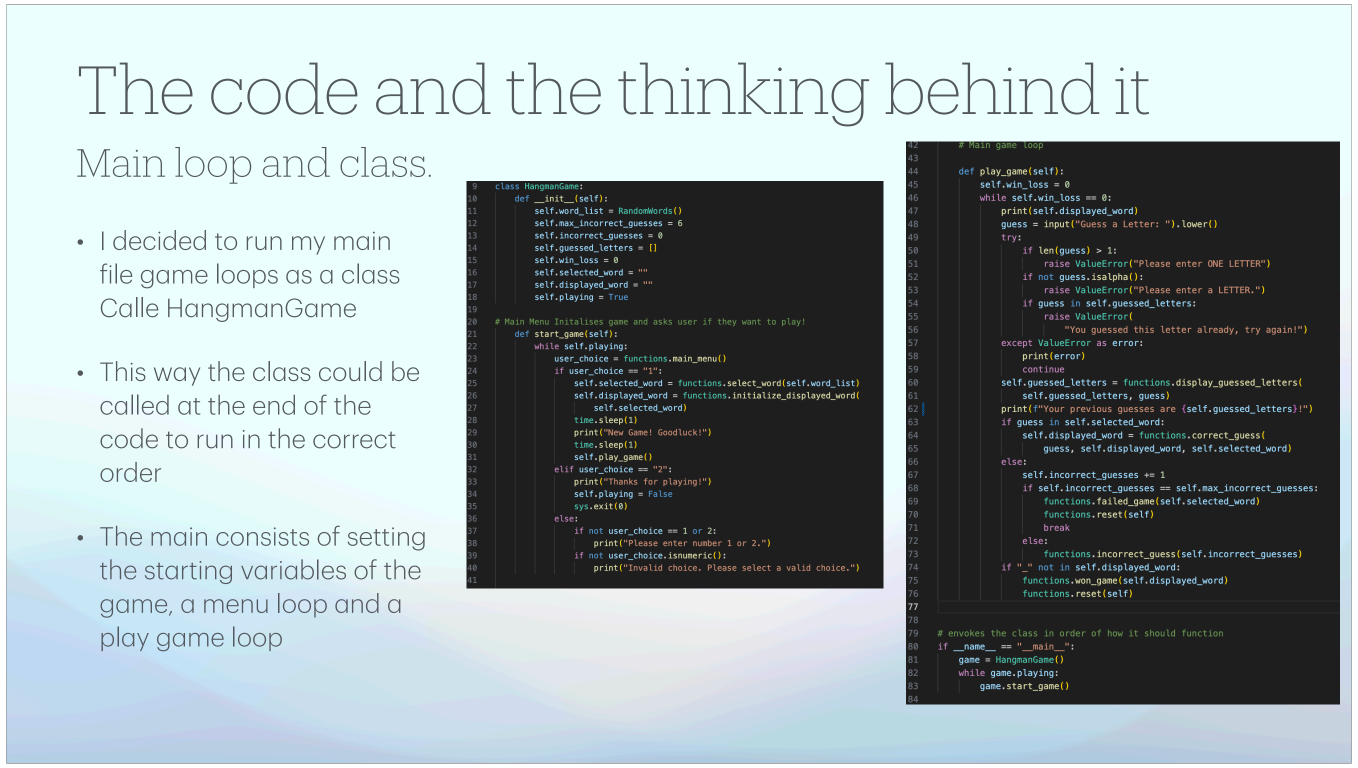
### My functions

- These are all my simple but effective functions. Each function has a specific purpose which I have commented in my code. I tried to prioritize using functions to keep my code DRY. There is a Total of 10 functions that all serve there purpose within main.
- I want to explain one function in particular which is the correct_guess function I will explain in speaker notes.

```python
import pyfiglet
import os
from hangman_art import print_hangman
import time
from random_word import RandomWords

# clear screen function clears terminal state for windows or mac


def clear_screen():
    if os.name == "nt":
        os.system("cls")
    else:
        os.system("clear")

# resets variable back to pre game state and clears screen for user experience


def reset(self):
    self.incorrect_guesses = 0
    self.guessed_letters = []
    self.win_loss = 1
    clear_screen()

# prints correct hangman art for incorrect guesss


def incorrect_guess(incorrect_guesses):
    print_hangman(incorrect_guesses)

# appeneds all guesses to a list to be displayed on screen showing previous guesses


def display_guessed_letters(list_of_guesses, guess):
    list_of_guesses.append(guess)
    return list_of_guesses

# displays game over message resets back to main menu


def failed_game(selected_word):
    print_hangman(incorrect_guess)
    ascii_banner = pyfiglet.figlet_format("GAME OVER!")
    print(ascii_banner)
    print(f"Nice try the word you were looking for was: {selected_word}")
    print("PlEASE WAIT... LOADING MENU...")
    time.sleep(2.5)

# displays won game message resets to main menu

```

```python

def won_game(displayed_word):
    ascii_banner = pyfiglet.figlet_format("WINNER!")
    print(ascii_banner)
    print(
        f"Congratulations! You have won the game! The word was: {displayed_word}")
    print("PlEASE WAIT... LOADING MENU...")
    time.sleep(2.5)

# function selects a word from random-word


def select_word(word_list):
    return word_list.get_random_word()

# Displays selected word as underscores


def initialize_displayed_word(word):
    return "_" * len(word)

# function for main menu


def main_menu():
    ascii_banner = pyfiglet.figlet_format("HANGMAN!")
    print(ascii_banner)
    print(""" <------Can you guess the secret word!-----> """)
    print("1. Play")
    print("2. Exit")
    choice = input("Enter your choice: ")
    return choice

# checks for correct guess in displayed word iterates though and makes a list of the
# letters that are withn the word


def correct_guess(correct_letter, displayed_word, selected_word):
    iteration = displayed_word
    for i in range(len(selected_word)):
        if correct_letter == selected_word[i]:
            lst = list(iteration)
            lst[i] = correct_letter
            iteration = ("".join(lst))
    return iteration

```

Most of my functions are simple but affective. My correct guesses function was the trickiest function as this is where string manipulation came into play so let me explain what is happening. When a guessed letter is in the displayed underscores the function indexes through the word and adds the letters to a variable iteration this is then converted to a list as to continue iterating through the length so that if there is more then 1 of the same letter it won't stop once finding the first one but add it to the list. Finally the list is then changes to a join of the string so that the letters take place of the underscores. This was by far the trickiest function for me to make but now it works very well.



## The code and the thinking behind it

### Main loop and class.

- I decided to run my main file game loops as a class Calle HangmanGame
- This way the class could be called at the end of the code to run in the correct order
- The main consists of setting the starting variables of the game, a menu loop and a play game loop

```python
class HangmanGame:
    def __init__(self):
        self.word_list = RandomWords()
        self.max_incorrect_guesses = 6
        self.incorrect_guesses = 0
        self.guessed_letters = []
        self.win_loss = 0
        self.selected_word = ""
        self.displayed_word = ""
        self.playing = True

# Main Menu Initalises game and asks user if they want to play!
    def start_game(self):
        while self.playing:
            user_choice = functions.main_menu()
            if user_choice == "1":
                self.selected_word = functions.select_word(self.word_list)
                self.displayed_word = functions.initialize_displayed_word(
                    self.selected_word)
                time.sleep(1)
                print("New Game! Goodluck!")
                time.sleep(1)
                self.play_game()
            elif user_choice == "2":
                print("Thanks for playing!")
                self.playing = False
                sys.exit(0)
            else:
                if not user_choice == 1 or 2:
                    print("Please enter number 1 or 2.")
                if not user_choice.isnumeric():
                    print("Invalid choice. Please select a valid choice.")

```

```python
    # Main game loop

    def play_game(self):
        self.win_loss = 0
        while self.win_loss == 0:
            print(self.displayed_word)
            guess = input("Guess a Letter: ").lower()
            try:
                if len(guess) > 1:
                    raise ValueError("Please enter ONE LETTER")
                if not guess.isalpha():
                    raise ValueError("Please enter a LETTER.")
                if guess in self.guessed_letters:
                    raise ValueError(
                        "You guessed this letter already, try again!")
            except ValueError as error:
                print(error)
                continue
            self.guessed_letters = functions.display_guessed_letters(
                self.guessed_letters, guess)
            print(f"Your previous guesses are {self.guessed_letters}!")
            if guess in self.selected_word:
                self.displayed_word = functions.correct_guess(
                    guess, self.displayed_word, self.selected_word)
            else:
                self.incorrect_guesses += 1
                if self.incorrect_guesses == self.max_incorrect_guesses:
                    functions.failed_game(self.selected_word)
                    functions.reset(self)
                    break
                else:
                    functions.incorrect_guess(self.incorrect_guesses)
            if "_" not in self.displayed_word:
                functions.won_game(self.displayed_word)
                functions.reset(self)


# envokes the class in order of how it should function
if __name__ == "__main__":
    game = HangmanGame()
    while game.playing:
        game.start_game()

```

Once again I decided to use while loops for my start game and main game loops my main file relies on importing the functions file and adds conditions to the functions depending on how the game is played. Within the main I used try except statements to catch Value errors from the user as there is many conditions that must be met such ask making sure numbers, symbols don't effect the incorrect guesses counter. I used conditional statements to control the flow of the game. The main needed to use functions and error handling and control flow in order for the game to run smoothly. Also you will see the win_loss is hardest this allows separation of the loops for a new game to start.

# The code and the thinking behind it

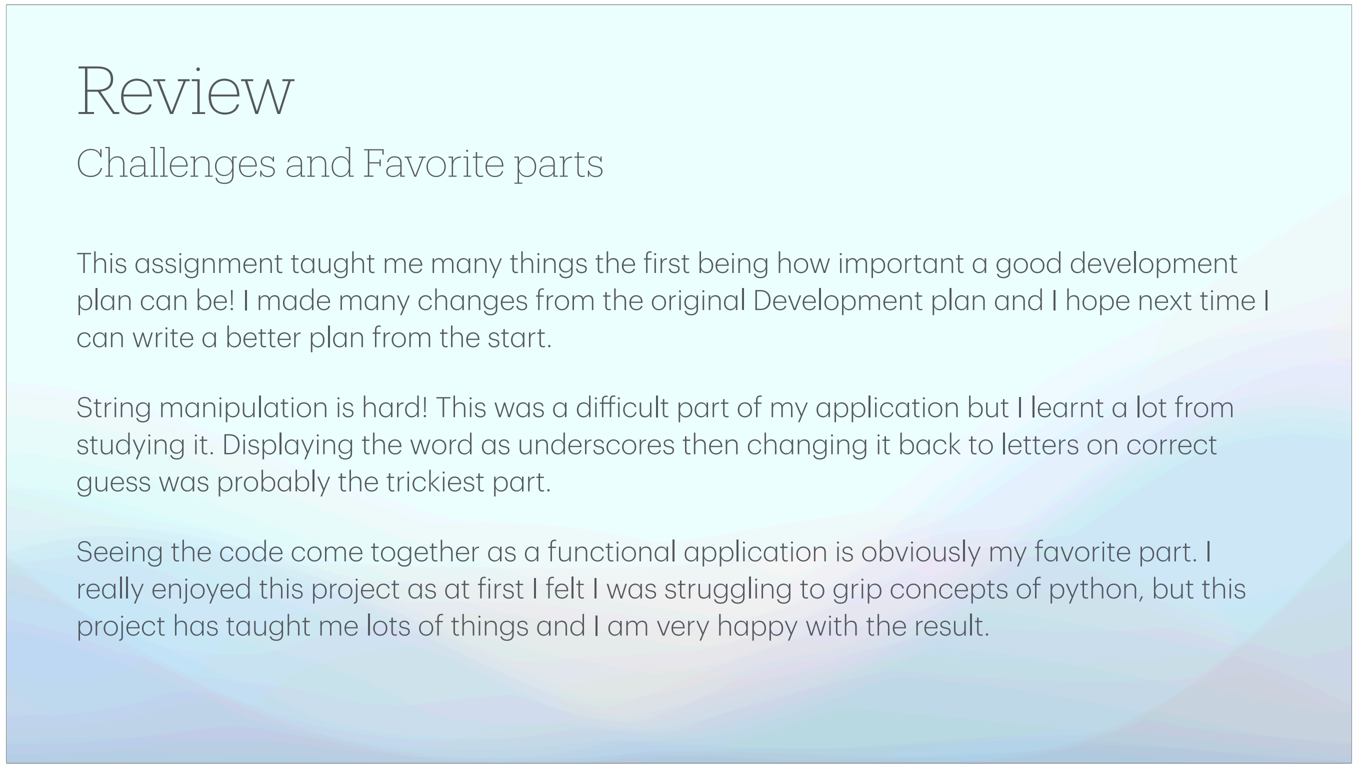
## Summary

As seen I did a walkthrough of the main features of my code and functions and how it works.

The code is more in-depth but I wanted to explain my logic as best as I could.

Using all part of python loops, variables, conditional statements, classes and error handling to make my program run as smoothly as possible.

I think this code is easy to understand its functionality if another coder was to look at it they would be aware of how the system works.



## Review

Challenges and Favorite parts

This assignment taught me many things the first being how important a good development plan can be! I made many changes from the original Development plan and I hope next time I can write a better plan from the start.

String manipulation is hard! This was a difficult part of my application but I learnt a lot from studying it. Displaying the word as underscores then changing it back to letters on correct guess was probably the trickiest part.

Seeing the code come together as a functional application is obviously my favorite part. I really enjoyed this project as at first I felt I was struggling to grip concepts of python, but this project has taught me lots of things and I am very happy with the result.

Thank you very much for taking the time to read me code and slide deck. I hope you enjoy my application.