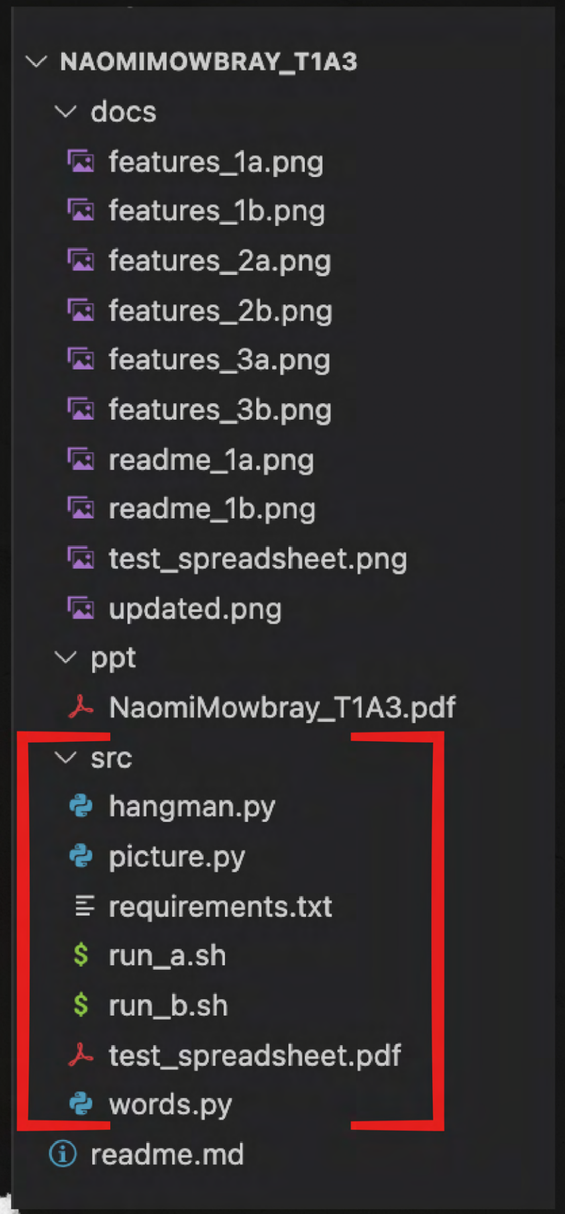
NAOMIMOWBRAY_T1A3'S

PYTHON HANGMAN



FILE ORGANIZATION

Well, the game's hangman, so, naturally, the game file is called **hangman.py.**

The picture that shows our progress (or lack thereof!) is in a dictionary in the **picture.py** file.

words.py is where our nice huge ~2,400 list of potential words is kept, as a library for the mystery word to be randomly selected from.

All requirements are in **requirements.txt**, as per convention ☺

The shell scripts that allow the average Joe to run Python Hangman are named **run_a.sh** and **run_b.sh.**

and **test_spreadsheet.pdf** shows the list of tests and procedures I used to ensure the program runs correctly.

PACKAGES

PACKAGES + SCOREBOARD SET UP

There are four packages that brings all of the gloriousness that is my Python Hangman:

- **csv:** to keep track of everyone's wins/losses
- **random:** to randomly select our mystery word
- **emoji:** for a little creepy pizazz to the game comments
- **colored:** adds a way more interesting experience to our hanging man's fate

We also need to import our **hanging dictionary** from **picture.py** and the list of **potential_words** from **words.py** to show where our man's at when you guess wrong letters and to give the program options to pick a word from.

The **file variable** is because we need to give a destination that our **scores.csv** file will be created in.

The **scorecard variable** is going to be updated every game, so we leave it blank for now!

And the **try/except blocks** are for the creation of that file. It's the very first thing that runs when the program is open, and tests by checking if there's the **file** that we want it. Obviously, the very first time the game's opened it won't be there, so the **except block** creates it for you!

```python
import csv
import random

import emoji
from colored import fg, attr

from picture import hanging
from words import potential_words

file = "src/scores.csv"
scorecard = ""

try:
    with open(file, "r") as scores:
        reader = csv.DictReader(scores)
        scorecard = list(reader)

except FileNotFoundError:
    with open(file, "w", newline="") as scores:
        columns = ["Player Name", "Total Wins", "Total Losses"]
        writer = csv.DictWriter(scores, fieldnames=columns)
        writer.writeheader()

```

FUNCTIONS

THE 'INTRO' FUNCTION

The '**intro**' function is the true beginning of your interactive Python Hangman sesh.

We create a global variable **name** and ask the player for their name, which is applied (and remembered!).

We also start our **wins** and **losses** at zero because that's where we all start for most things in life! 0!

From there, we're going to open our **scores file** and check if this player has played before.

To do this, we create a list of all the information in **scores** and call it **scorecard**. Then, we check every row in that **scorecard**:

- Firstly, we're going to assume that all players are new: that is, 'Are they a **returned_player**'? **False!**
- However, if the given name matches a name already in our **scores.csv** then they *are* a **returned_player**, so that changes to **True**

Now, if **returned_player** = **False**, we need to add them to our **scorecard** by appending them (adding them!).

The appended **scorecard** is saved, and we write it back over our **scores** file to update it correctly and voila!

```python
def intro(file, scorecard):
    global name
    name = input(emoji.emojize(
        "Hellllllo there :clown_face: What's your name? "))
    global wins
    wins = 0
    global losses
    losses = 0
    print("_____________________________________________")

    with open(file, newline="") as scores:
        columns = ["Player Name", "Total Wins", "Total Losses"]
        reader = csv.DictReader(scores, fieldnames=columns)
        scorecard = list(reader)

        return_player = False
        for row in scorecard:
            if row["Player Name"] == name:
                return_player = True
                break

        if not return_player:
            new_player = {"Player Name": name,
                          "Total Wins": wins, "Total Losses": losses}
            scorecard.append(new_player)

        with open(file, "w", newline="") as scores:
            columns = ["Player Name", "Total Wins", "Total Losses"]
            writer = csv.DictWriter(scores, fieldnames=columns)
            writer.writerows(scorecard)
```

FUNCTIONS

THE 'CHOOSE MYSTERY WORD' FUNCTION

The '**choose_mystery_word**' function is pretty self explanatory - it chooses a mystery word for us!

Lowercase and uppercase letters are read as different characters in Python. So, to make sure the program doesn't read 'g' and 'G' for example, and waste one of your chances, I've used the **.upper()** method. This automatically changes all the letters in the **mystery_word** to capitals, as well as your input guesses - which we'll see in a sec.

The letters in the **mystery_word** are then added to a **set** called **word_letters**, which kind of breaks down the word into its individual characters. This allows your guess to be able to be checked against every letter in there in the next function.

```python
def choose_mystery_word():
    # For assignment reference only:
    # These are what I refer to in the test spreadsheet, as used
    # throughout building and testing the application:

    # potential_words = [
    #     "hello",
    #     "bye",
    #     "lamp",
    # ]

    global mystery_word
    mystery_word = random.choice(potential_words).upper()
    global word_letters
    word_letters = set(mystery_word)

```

Note: the commented out part in green was left just to help educators refer to how I did some of the testing.

If they weren't marking and needed to see, I'd delete from the code because it's not necessary.

FUNCTIONS

THE 'HANGMAN_GAME' FUNCTION

```python
def hangman_game(mystery_word, word_letters):
    global chances
    chances = 9
    guessed = set()
    letters = set("ABCDEFGHIJKLMNOPQRSTUVWXYZ")

    while len(word_letters) > 0 and chances > 0:
        player_word = ""

        for letter in mystery_word:
            if letter in guessed:
                player_word = player_word + letter
            else:
                player_word = player_word + "_ "

        print(f"\nThis is our word so far: {fg(117)}{player_word}{attr(0)}")
        print("When you make a guess, it'll show here: ", " ".join(guessed))
        guess = input("What letter do you want to try? ").upper()

        # The following if statement checks if our guess is still in the
        # alphabet, minus the letters we've already guessed (i.e. what
        # letters haven't we guessed yet?)
        if guess in letters - guessed:
            guessed.add(guess)
            if guess in word_letters:
                word_letters.remove(guess)
            else:
                chances = chances - 1
                print(hanging[chances])

        elif guess in guessed:
            print(
                f"\nYou already tried {guess}. Maybe try a different letter?")
        else:
            print(emoji.emojize(
                f"""\nCome on {name}, {guess} isn't a letter :woozy_face:
Let's give that another go..."""))
```

Alrighty, now that the program knows who we are and has chosen a word for you, we can play!

Your **chances** are set to 9, because I am the boss and I said so.

guessed starts off empty - you haven't guessed anything yet, but as soon as you do, those **guess**es will now have somewhere to go.

letters is essentially our alphabet - it's a set of the characters that can be used in the game, which quickly filters out the fact that you can't try a number or a symbol. They're in capitals to match our **mystery_word** of course, and they're in a set so that when you make a **guess**, it will be removed from there to essentially give the 'left over letters' for your future guesses. More on that in next slide.

The **while loop**:
While there's still letters in the **word_letters** that haven't yet been guessed and you've got at least one chance, you're 'allowed to play'.

The **player_word** is an empty variable, because it's going to change throughout the game (from _'s to letters).

CONTINUED

```python
def hangman_game(mystery_word, word_letters):
    global chances
    chances = 9
    guessed = set()
    letters = set("ABCDEFGHIJKLMNOPQRSTUVWXYZ")

    while len(word_letters) > 0 and chances > 0:
        player_word = ""

        for letter in mystery_word:
            if letter in guessed:
                player_word = player_word + letter
            else:
                player_word = player_word + "_ "

        print(f"\nThis is our word so far: {fg(117)}{player_word}{attr(0)}")
        print("When you make a guess, it'll show here: ", " ".join(guessed))
        guess = input("What letter do you want to try? ").upper()

        # The following if statement checks if our guess is still in the
        # alphabet, minus the letters we've already guessed (i.e. what
        # letters haven't we guessed yet?)
        if guess in letters - guessed:
            guessed.add(guess)
            if guess in word_letters:
                word_letters.remove(guess)
            else:
                chances = chances - 1
                print(hanging[chances])

        elif guess in guessed:
            print(
                f"\nYou already tried {guess}. Maybe try a different letter?")
        else:
            print(emoji.emojize(
                f"""\nCome on {name}, {guess} isn't a letter :woozy_face:
Let's give that another go..."""))
```

The **for loop** basically just determines if the letters in the **mystery_word** should show as an underscore (because they're not yet guessed), or a letter (because you guessed correctly), and THAT'S where the **player_word** variable is used!

You get told the current progress of the word at the beginning of each guess with the first **print** statement there, and I've set the colour to blue to give a little hope (blue's typically calming ☺).

Then, your set of **guessed** letters are printed - obviously, on your first go there's nothing there!

Then it's time for your **guess!** I've used **.upper()** again to change whatever you type into a capital, so that it matches the word correctly, like I said earlier.

The next comments help explain this as well, but basically if your **guess** is in the alphabet *minus* the letters you've already guessed, that means you haven't guessed it yet! So we add it to your set of **guessed** letters.

CONTINUED

```python
def hangman_game(mystery_word, word_letters):
    global chances
    chances = 9
    guessed = set()
    letters = set("ABCDEFGHIJKLMNOPQRSTUVWXYZ")

    while len(word_letters) > 0 and chances > 0:
        player_word = ""

        for letter in mystery_word:
            if letter in guessed:
                player_word = player_word + letter
            else:
                player_word = player_word + "_ "

        print(f"\nThis is our word so far: {fg(117)}{player_word}{attr(0)}")
        print("When you make a guess, it'll show here: ", " ".join(guessed))
        guess = input("What letter do you want to try? ").upper()

        # The following if statement checks if our guess is still in the
        # alphabet, minus the letters we've already guessed (i.e. what
        # letters haven't we guessed yet?)
        if guess in letters - guessed:
            guessed.add(guess)
            if guess in word_letters:
                word_letters.remove(guess)
            else:
                chances = chances - 1
                print(hanging[chances])

        elif guess in guessed:
            print(
                f"\nYou already tried {guess}. Maybe try a different letter?")
        else:
            print(emoji.emojize(
                f"""\nCome on {name}, {guess} isn't a letter :woozy_face:
Let's give that another go..."""))
```

If your **guess** is in the **word_letters**, then it's removed from there - almost like a countdown of what's left.

Else if it's not in the word, you lose a chance and we print out the current state our hanging man is in!

Now, if your **guess** is already in **guessed**, we know that you've tried it before, so you can just go right ahead and try something else!

I decided not to take a chance off, even though it's a bit silly of you/the player.

And if the letter doesn't match any of the above, that means it must not be a valid character (or even multiple letters), so you're told to try again.

FUNCTIONS

THE 'DID_HE_DIE' FUNCTION

A nice and easy function/question: **did_he_die**?

Remember that **while loop** that was in the **hangman_game** function?

That's kinda where we are now.

```
while len(word_letters) > 0 and chances > 0:
```

If your **chances** get down to zero, then yep, that little man's goooooooone. You're given the total **mystery_word** so that you can see what you did wrong and get some closure (in red, because you lost), and then wished luck for next time.

If there are no more letters in **word_letters**, then you've guessed them all correctly AND have lives left - you won!
We print that out too, with a little party face for you

```
def did_he_die(chances):
    if chances == 0:
        print(emoji.emojize(
            f"""\nDang {name}, the word was {fg(196)}{mystery_word}{attr(0)}.
Better luck next time! :crossed_fingers:"""))
    else:
        print(emoji.emojize(
            f"""\nNice job {name}, you legend! :partying_face:
The word was {fg(117)}{mystery_word}{attr(0)} - you saved the man!"""))
```

FUNCTIONS

THE 'OUTCOME' FUNCTION

Not gunna lie, this one was a REAL pain in the butt to figure out.

It comes back to the **scorecard** - which gave me nightmares, but I conquered eventually with a STACK of nudging and explanation and patience from Alex!

It's kind of the same concept of adding a new player - we want to read the **scorecard,** find the right player **name**, update their Total Wins/Total Losses, depending on the outcome, and then rewrite that over the **scorecard** to correctly show the new tally.

Much easier said than done!

To update their scores, I had to convert the number of Wins or Losses into an **integer** so I could add 1 for that win/loss, then convert it back to a **string** so it all works nicely.

Side note: I knoooooooow the inline comment '*You won!*' on next to **else:** isn't strictly kosher with Pep 8, but I was SO FREAKING happy for both myself for figuring it out and you for winning the game that I kept that in there anyway. (I have just updated it so that there are two spaces, so at least that helps with the style code ☺)

```python
def outcome(chances, losses, wins):
    if chances == 0:
        with open(file, "r") as scores:
            reader = csv.DictReader(scores)
            scorecard = list(reader)

        for row in scorecard:
            if row["Player Name"] == name:
                row["Total Losses"] = str(int(row["Total Losses"]) + 1)

        with open(file, "w", newline="") as scores:
            columns = ["Player Name", "Total Wins", "Total Losses"]
            writer = csv.DictWriter(scores, fieldnames=columns)
            writer.writeheader()
            writer.writerows(scorecard)

    else: # You won!
        with open(file, "r") as scores:
            reader = csv.DictReader(scores)
            scorecard = list(reader)

        for row in scorecard:
            if row["Player Name"] == name:
                row["Total Wins"] = str(int(row["Total Wins"]) + 1)

        with open(file, "w", newline="") as scores:
            columns = ["Player Name", "Total Wins", "Total Losses"]
            writer = csv.DictWriter(scores, fieldnames=columns)
            writer.writeheader()
            writer.writerows(scorecard)
```

FUNCTIONS

THE 'MAIN' FUNCTION

Then, we put that all together in the **main** function!

We assume that the player is here to play, so set **while loop** to **True**, then ask the player just to be sure!

- If whatever they type in starts with an '**N**', the loop will **break**, and the program ends.
- If whatever they type in starts with a '**Y**', they play!
- If they're a little cheeky and type anything else, we're not letting them get away with it and they'll have to try again ☺

```python
def main():
    while True:
        play = input(
            f"Do you want to play Python Hangman and save this fella? (y/n) "
            ).upper()[0]
        if play == "N":
            print(emoji.emojize(
                f"No worries {name}, let's play later instead :waving_hand:"))
            break
        elif play == "Y":
            print(f"Righto {name}, let's do this!")
            choose_mystery_word()
            hangman_game(mystery_word, word_letters)
            did_he_die(chances)
            outcome(chances, losses, wins)
        else:
            print(emoji.emojize(
                "Innnnteresting, that's not a yes OR a no... :thinking_face:"))


intro(file, scorecard)
main()
```

The program will run through our functions:

- **choose_mystery_word**
- **hangman_game**
- **did_he_die**
- **outcome**

And do all the things we want it to 😎

Riiiiiiight down the bottom, where we see:

```python
intro(file, scorecard)
main()
```

That's how it all happens!
The **intro** is ran through once (so that you don't keep getting asked your name), and then the magic in **main**.

WHAT IT ALL LOOKS LIKE - WINNING

Let's just say that you have a perfect run - it'll look something like this:

```
Hellllllo there 🤡 What's your name? Naomi
_____________________________________________
Do you want to play Python Hangman and save this fella? (y/n) yeahhhhhhhh I do!
Righto Naomi, let's do this!

This is our word so far: _ _ _
When you make a guess, it'll show here:
What letter do you want to try? b

This is our word so far: B_ _
When you make a guess, it'll show here:  B
What letter do you want to try? Y

This is our word so far: BY_
When you make a guess, it'll show here:  B Y
What letter do you want to try? e

Nice job Naomi, you legend! 🥳
The word was BYE — you saved the man!
Do you want to play Python Hangman and save this fella? (y/n)
```

WHAT IT ALL LOOKS LIKE - LOSING

And if you don't guess correctly?

You'll end up going through your chances like this:

(Obviously, with the questions and guesses in between)

```
 _______
|       |
|
|
|
|_______
```

8 turns left, just breathe, you've got this.

```
 _______
|       |
|       |
|
|
|
|_______
```

Okay well 7 turns remaining isn't terrible?

```
 _______
|       |
|       o
|
|
|
|_______
```

Wait, there's his head ... With 6 guesses to go!

```
 _______
|       |
|       o
|       |
|
|
|_______
```

Yep, 5 guesses left now...

```
 _______
|   |
|   \o
|   |
|
|
|_______
```

I think he's waving for help – 4 more chances

```
 _______
|   |
|  \o/
|   |
|
|
|_______
```

EEEP, DANGER: DOWN TO 3 TURNS

```
 _______
|   |
|  \o/
|   |
|  /
|
|_______
```

Come on, there's still hope for your last 2 guesses!

```
 _______
|   |
|  \o/
|   |
|  / \
|
|_______
```

Ohhhhh gosh, this is your (and his!) last chance...

YOU LOST THE BATTLE? DAMN.

```
 _______
|   |
|   o
|  /|\
|  / \
|
|_______
```

Ba-bow, you lose (and so does our unfortunate man).

Dang Naomi, the word was LAMP.
Better luck next time! ☝️

OTHER BITS

```
This is our word so far: _ A_ P
When you make a guess, it'll show here:  P A
What letter do you want to try? 2

Come on Naomi, 2 isn't a letter 😔
Let's give that another go...

This is our word so far: _ A_ P
When you make a guess, it'll show here:  P A
What letter do you want to try? ▌
```

When you make a guess, typing anything that's not in our set of **letters**:

```
letters = set("ABCDEFGHIJKLMNOPQRSTUVWXYZ")
```

will look like this

And any letters that you've already guessed will look like this

```
This is our word so far: LA_ P
When you make a guess, it'll show here:  P A L
What letter do you want to try? l

You already tried L. Maybe try a different letter?

This is our word so far: LA_ P
When you make a guess, it'll show here:  P A L
What letter do you want to try? ▌
```

OTHER BITS

When you're asked if you want to play again, you've gotta give a 'y' or 'n' - anything else that starts differently will just keep asking you till you comply:

```
Do you want to play Python Hangman and save this fella? (y/n) ahhhhhh I like llamas
Innnnteresting, that's not a yes OR a no... 🤔
Do you want to play Python Hangman and save this fella? (y/n)
```

```
Do you want to play Python Hangman and save this fella? (y/n) n
No worries Naomi, let's play later instead 👋
```

And laaaaaaast but not least, all the scores are over here in the **scores.csv** file, and they look like this:

Note: if you add your name and then straight up change your mind and say '*no thank you, I don't actually want to play*', it'll still save your name for next time!

```
Player Name,Total Wins,Total Losses
Naomi,4,1
Mickey Mouse,1,3
Anthony Albanese,0,0
Rupert,2,0
```

PICTURE.PY FILE

```python
from colored import bg, attr


# Note: This does look quite repetitive, but that's on purpose!
# If I don't reset the color attribute every line, it will apply
# that color across the ENTIRE line, and ends up looking silly.
# By setting the color at the beginning and then using {attr("reset")}
# at the end of every line, we get a nice even block every time!

hanging = {
    8: f"""
        {bg(15)} ________ {attr("reset")}
        {bg(15)} |        {attr("reset")}
        {bg(15)} |        {attr("reset")}
        {bg(15)} |        {attr("reset")}
        {bg(15)} |        {attr("reset")}
        {bg(15)} |_______ {attr("reset")}
        {bg(15)}          {attr("reset")}
8 turns left, just breathe, you've got this.""",
```

This is how the man comes to life (until he loses it!).

The code here seemed quite repetivite, but I needed it this way to look right, and I loved how he turned out!

WORDS.PY FILE

```python
# For assignment reference only:
# These are what I refer to in the test spreadsheet, as used
# throughout building and testing the application:

# potential_words = [
#     "hello",
#     "bye",
#     "lamp",
#   ]

potential_words = [
    "aback",
    "abaft",
    "abandoned",
    "abashed",
    "aberrant",
```

And over in **words.py** is the nice big list of **potential_words** for the game to choose from each time!

As with the comments back in the **hangman.py** file, I've kept the first mini **potential_words** list there for reference as I used it in the trial and error a whole heap!

FAVOURITE PARTS

Honestly, choosing the red gradient for the hanging man picture was super satisfying!

As were the semi-creepy clown emojis that gave me a bit of an *It* or *Saw* vibe... because really, the whole backstory of hangman is pretty grim.

The ABSOLUTE hardest thing for me to get past was the csv file updating scores and players correctly - massive shout out to Alex who helped get me on track with that!!!

Which should technically go in the challenges part, but I can't not say that I literally yelled with joy when I first saw those numbers going up how I wanted them to on the file.

Having it finally click *almost* made all the pain worth it.

CHALLENGING PARTS

Obviously, like I said, the **scores.csv** file gave me both nightmares and tears and headaches and was altogether horrific.

I got there in the end, but I feel like it took a heck of a lot of brain space and time to try and make it click in my head, and then convert that into code that worked.

Overall though, my biggest challenge through this was a mental and emotional one.

My partner and I have had to come across to Perth (we're based in Brisbane) to help my parents in law as they've moved from Africa to Australia, and we all thought that supporting them in their transition was a huge priority.

I asked for an extension for this right at the start of the assignment period and was told that because it was the last assignment of term, the educator couldn't simply give a five day extension like every other assignment.

I gave flight details, explained the situation, and was told that I need to manage my schedule better. This felt both highly insensitive and unfair that other students have had extensions granted due 'stress', but my family migrating continents was bad 'time management' on my part.

The emotional toll of juggling trying to do my best for this, while needing to be there for my family has left a huge sour taste in my mouth, especially as it seems for any other assignment the extension would have been fine.

HAVE FUN!

PYTHON HANGMAN

https://github.com/captainaomi/NaomiMowbray_T1A3