

## Problem Two

I think you're really going to like this one! Although it may prove to be trickier than the previous. I'm not sure if you've ever played any of the Fallout games (or if you've even heard of them), but if you have, this exercise is based on one of the minigames from Fallout!

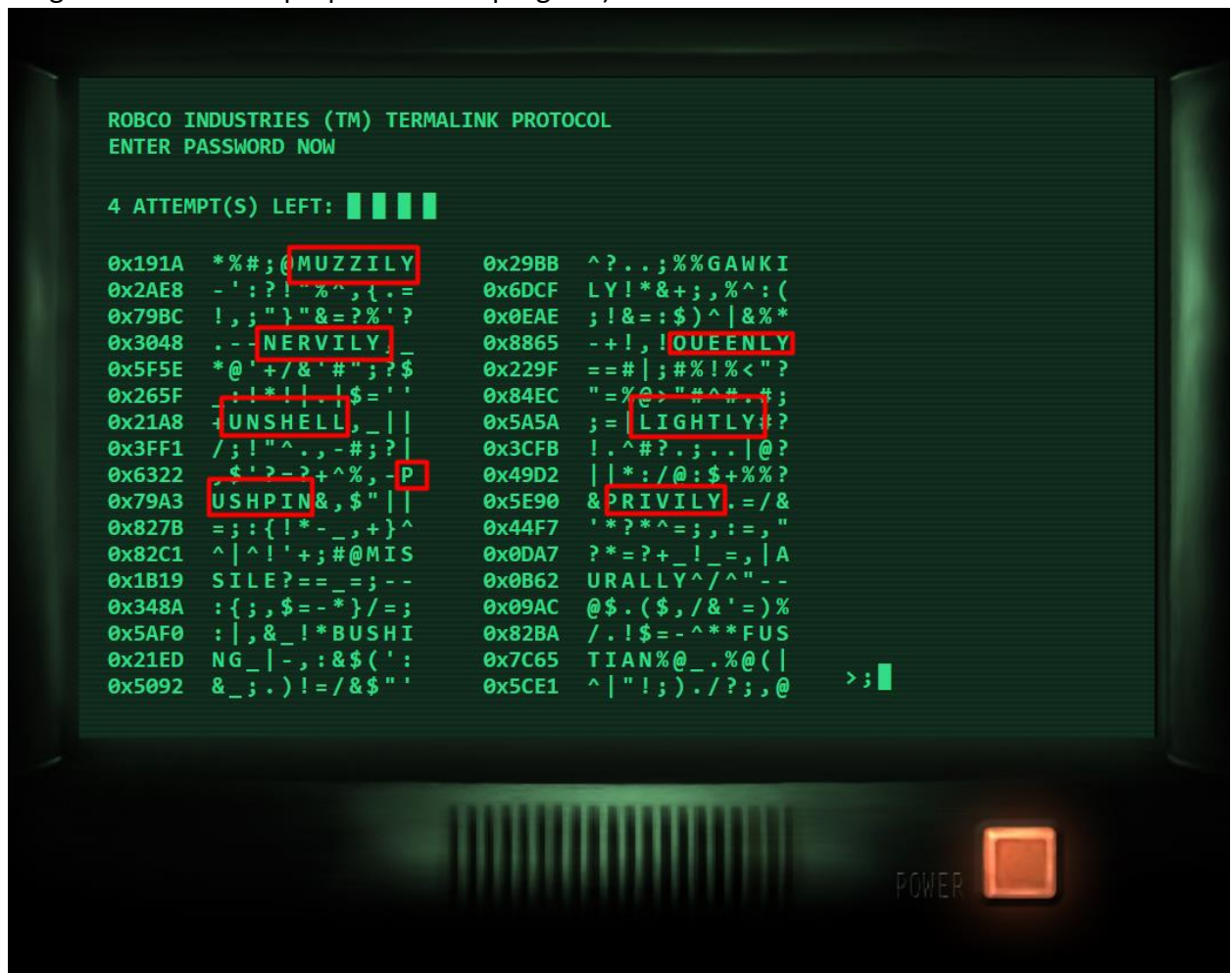
I looked around a little bit online and I found the following:

<http://mitchellthompson.net/demos/terminal/>

Go ahead, try it out. I'll wait.

Alright, so now hopefully you'll have a basic grasp of how the game works. Just as a recap, here's basically the rundown on what happens when you play.

1. Words are displayed to the screen, along with a bunch of garbled text (but we're going to ignore that for the purposes of our program):



2. When you correctly select the secret word from the words displayed, you win and access is granted!



3. Otherwise, if you guess incorrectly, the program tells you how many characters from the word you selected match with that of the password (or secret word). For example, selecting 'LATCHES' displays 2/7 correct, since the two characters 'ES' match with two characters from the secret word 'TOUZLES'. They both end in 'ES'.



Now let's try to model this game in Python! By the end of this challenge your result should look something like this:

```
condchallenge_solution
C:\Users\chris\AppData\Local\Programs\Python\Python38-32\python.exe
-----
Loading words from falloutdict.txt
Words loaded.
-----
Words:
    CRUMBLING
    SCATTERED
    STRUCTURE
    STUMBLING
    CONCERNED
    FOLLOWERS
    BEAUTIFUL
    ENDORPHIN
    OFFICIALS
-----
Guess?
CRUMBLING
0/9 Correct.
4 guesses left.
BEAUTIFUL
1/9 Correct.
3 guesses left.
OFFICIALS
You won!
9/9 Correct.
The secret word was: OFFICIALS

Process finished with exit code 1
```

Now let's get started! (By the way, remember to use the test file to check your work as you go!)

1. Picking the secret word
2. Getting the additional words
3. Displaying the words
4. Checking if the user's guess is valid
5. Comparing the user's guess with the secret word

### 1. Picking the secret word ( `pick_secret_word()` )

If you read through the code already written, you'll notice that the line:

```
WORDS = get_words() # The shuffled list of words from falloutdict.txt
```

Gets a list of words from the text file 'falloutdict.txt' which is stored in the same folder as this Python script. The words are stored in the list like so:

```
['MATTERS', 'CONDESCENDINGLY', 'HINTS', 'COW', 'PAYING', 'CLUES', 'BONE',  
'JONATHAN', 'DISAPPEARED', ..... ]
```

So basically, in each value of the list, a single word of some length exists.

Knowing this, we need to select a single word from this list randomly, in order for the rest of the game to work. That's your job!

### 2. Getting the additional words ( `get_additional_words()` )

Alright, so now that we have the secret word to be displayed, we need to pick the other words to be displayed along with it. But here's the catch: the length of each of the additional words must be equal to the length of the secret word! If you're not sure why, try playing the minigame again, and imagine if the words had different lengths. The reason should become quickly apparent.

### 3. Displaying the words ( `word_display()` )


This part is really up to you (which is why there is no test for this function). Basically all you have to do here is print out to the user the secret word and the additional words (making sure that the secret word is mixed in randomly with the others!) I suggest in the notes to do this line by line, to make your output more readable, but feel free to format your display however you'd like!

Also, make sure to use the variable `ADDITIONAL_WORDS_NUM` (defined at the top of file) in your code. This way, if you want to change the amount of words displayed later, you can do so by modifying that single line at the top of the file. Neat!


#### Hint:

Make sure that your secret word exists at a different position in the display every time your program is run, for example:

```
C:\Users\chris\AppData\Local\Programs\Python\Python
-----
Loading words from falloutdict.txt
Words loaded.
-----
Words:
    SKIES
    FIELD
    TAINT
    YOUNG
    PULLS
    DRANK
    STUFF
    STEAL
    CHECK
-----
Guess?
```



```
condchallenge_solution
C:\Users\chris\AppData\Local\Programs\Python\Python3
-----
Loading words from falloutdict.txt
Words loaded.
-----
Words:
    ASSAULTED
    OCCASIONS
    COMMITTEE
    SPONSORED
    SCRUBBERS
    STOREROOM
    GENERATOR
    CLUSTERED
    WORSHIPER
-----
Guess?
```



#### 4. Checking if the user's guess is valid ( `is_valid_guess()` )

Alright, so now this is the last thing we have to set up before working on the actual logic of the game. We just need to make sure that the entry from the user is actually one of the additional words or the secret word.

**Hint:**

Watch out for list mutation! If you aren't sure what that is, look it up online or ask me!

**5. Comparing the user's guess with the secret word ( `compare_user_entry()` )**

Now we finally get to work on the actual logic and rules that the game uses! In fact, there's only really one function of the game that needs to be handled; the amount of matching characters between the user's entry and the secret word must be counted and displayed to the user, in order to give him a hint as to what word to choose next. You must return the amount of matched characters.

**When you finish this file, give yourself a pat on the back! This is a much trickier challenge than the last, and I know I may be lacking in some of my descriptions. Feel free to contact me if you have any questions about any of the code you've seen or written!**