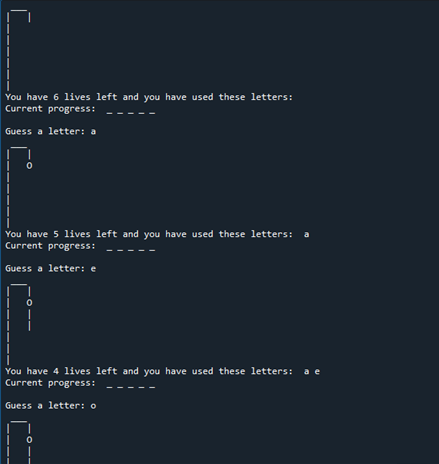
# **Hangman Game in Python**

This tutorial is about creating the Hangman game in Python. The game of Hangman is a word-guessing game played between the computer and a human player. The game play goes as follows:

* The computer generates a secret word
* There is a limited number of failed attempts that the computer needs to keep track of
* The human player tries to guess a letter in the secret word
* If the guess is wrong, the number of available tries is reduced, and more pieces of the *hanged man* picture is revealed
* The player continues to try until either the number of available tries is reduced to zero, or the letters in the word are all correctly guessed.
* Once the number of tries is zero, the complete picture of the hangman is displayed, and the human player loses.

The screenshots below show a sample gameplay:



## **Implementing Hangman in Python**

This tutorial assumes that you have Python 3 and a Python development environment installed in your computer. I am making use of the Spyder Development environment on the Windows 10. I also assume you have knowledge programming in Python.

The hangman game is divided into three modules to simplify the program:

1. **words.py**: Defines class **WordCache** that helps to load the words from a text file, and then allows the game to request a random word.
2. **game.py**: Defines the **Game** class. The **Game** class controls the actual game play.
3. **main.py**: The main program controlling the game object. It restarts the game when the player decides to play the game again.

Each of these modules’ functionality is discussed in detail below, along with screenshots of the hangman game code in python.

### **The Words module: words.py**

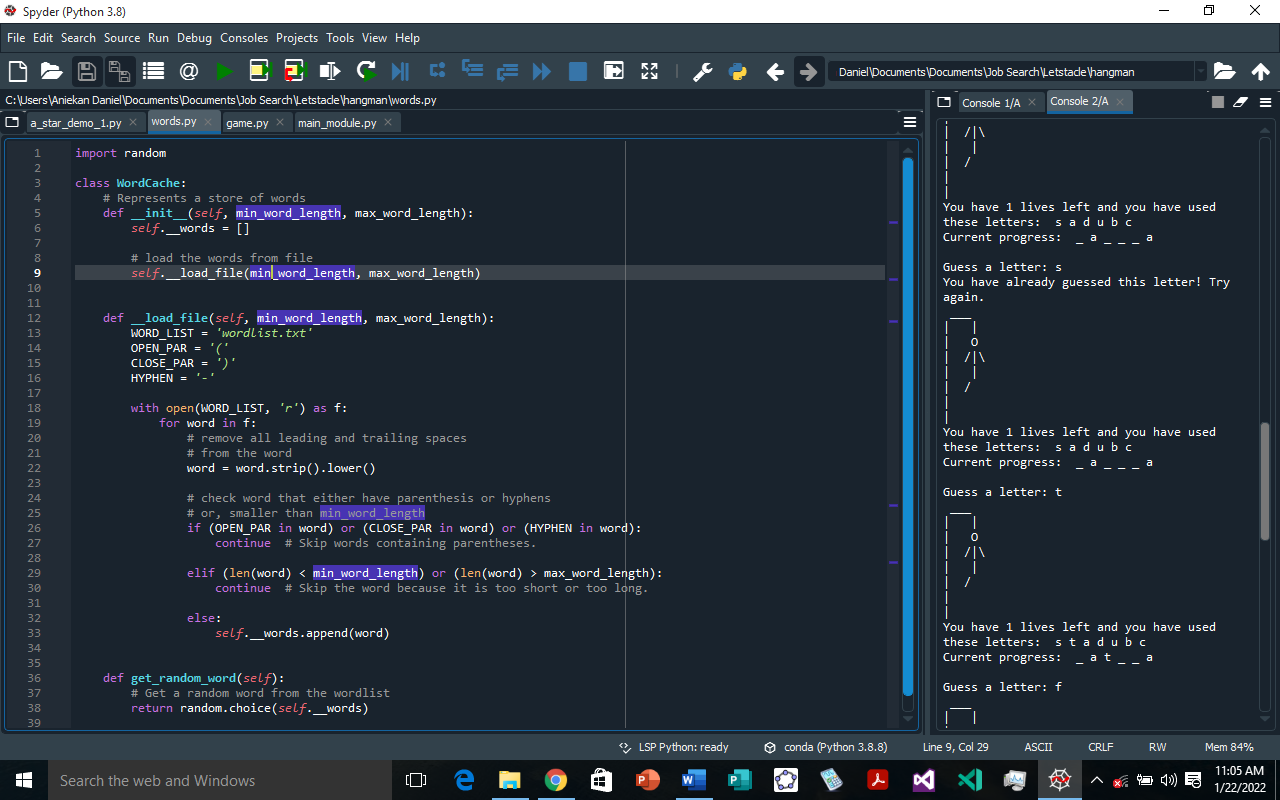
The **WordCache** class defined in this module helps to load words from the “*words.txt*” file, store it in memory, and provide it on request.

The **\_\_load\_file(min\_word\_length, max\_word\_length)** method loads the “*words.txt*” file into the **self.\_\_words** list attribute. However:

* If the word contains an ***open parenthesis***, a ***closed parenthesis***, or a ***hyphen***, it is filtered out
* If the number of letters in the word is less than **min\_word\_length**, or greater than **max\_word\_length**, it is also filtered out.

The **get\_random\_word()** method provides words from the store whenever it is requested. The method simply calls the **random.choice()** method from the **random** module to randomly select words from the **self.\_\_words** attribute.

The screenshot below shows the contents of the **words.py** module.



### **The Game module: game.py**

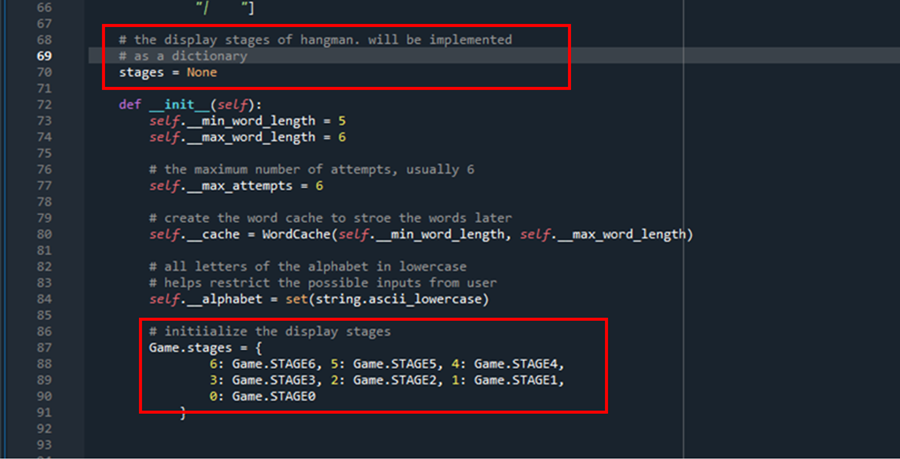
This **Game** class defined here contains the main functionality of the hangman game. However, it needs to import **words.py** so that it can fetch the secret words needed for each game play.

The **Game** class consists of the following definitions:

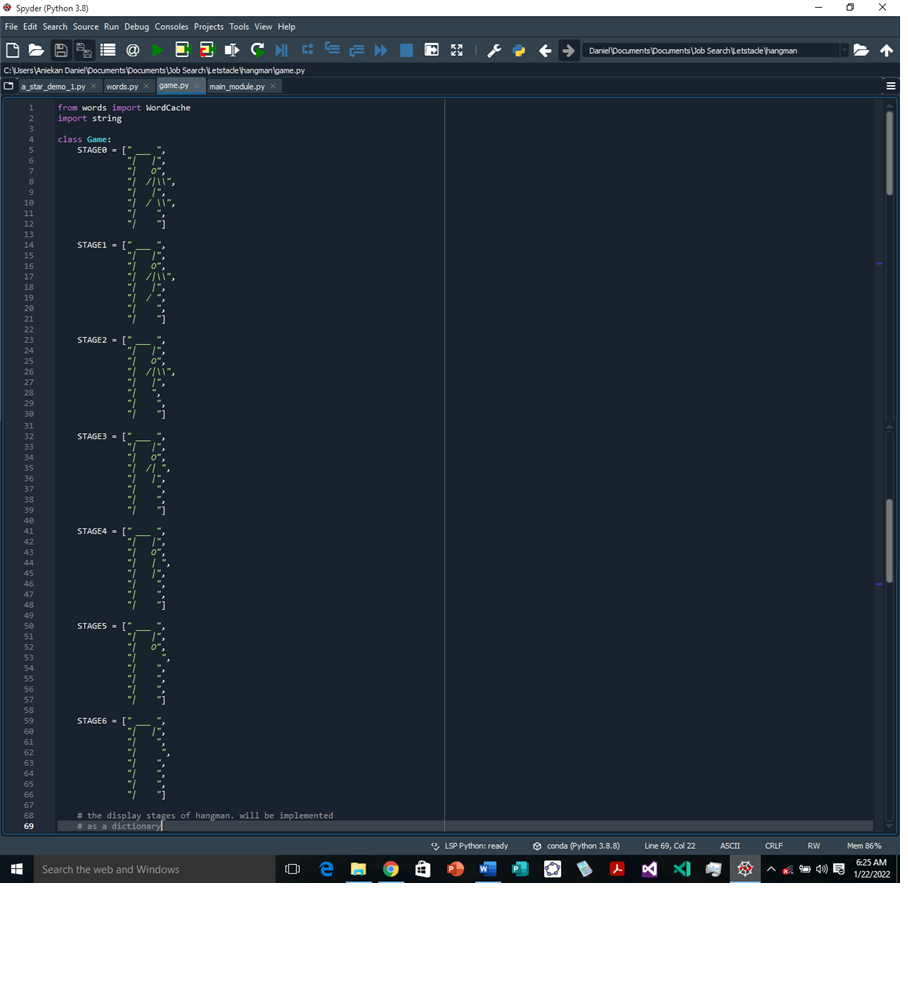
#### **Attributes**

1. **The hangman images**: The class attribute **stages** is a dictionary that stores each stage of the hangman image as a different image. The keys of the dictionary represent the lives/attempts remaining, while the values of the dictionary are the text-based images that will be displayed for each life.

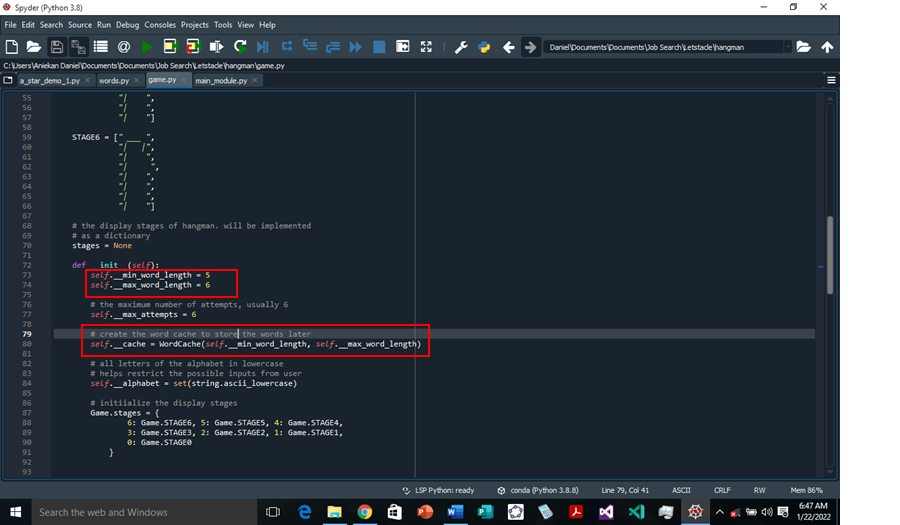
The screenshot below shows the definition and initialization of the **stages** attribute



1. **The hangman text-images**: The class attributes **STAGE0**, **STAGE1**, **STAGE2**, **STAGE3**, **STAGE4**, **STAGE5**, and **STAGE6**, store the images for the different values of lives, or attempts left.



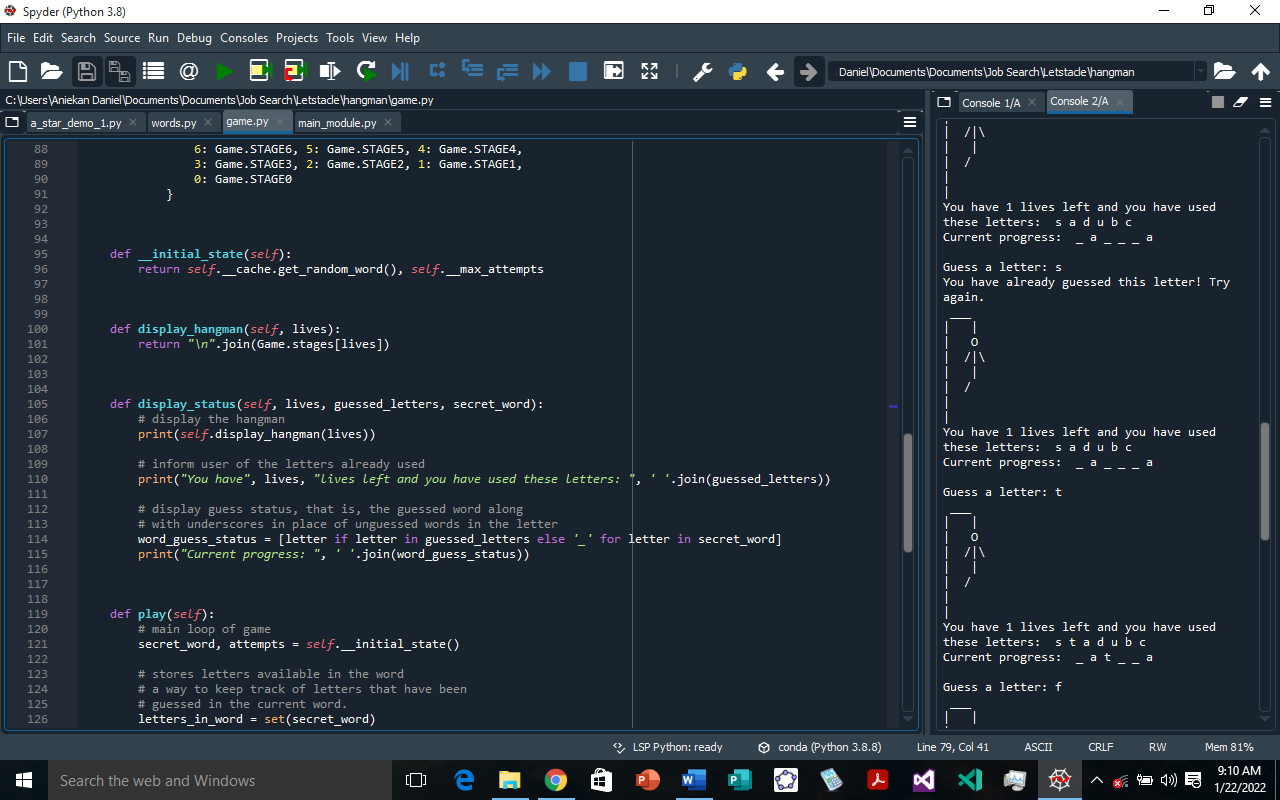
1. **The word length range and the word cache**: Each instance of the hangman **Game** class stores default values for the **minimum** and **maximum** length of words (***self***.**\_\_min\_word\_length**, and ***self***.**\_\_max\_word\_length**, respectively) that will be loaded from the text file by an instance of the **WordCache** class stored in the ***self***.**\_\_cache** attribute. For now, the value for **minimum** is set to 5, and the **maximum** word length is set to 6.



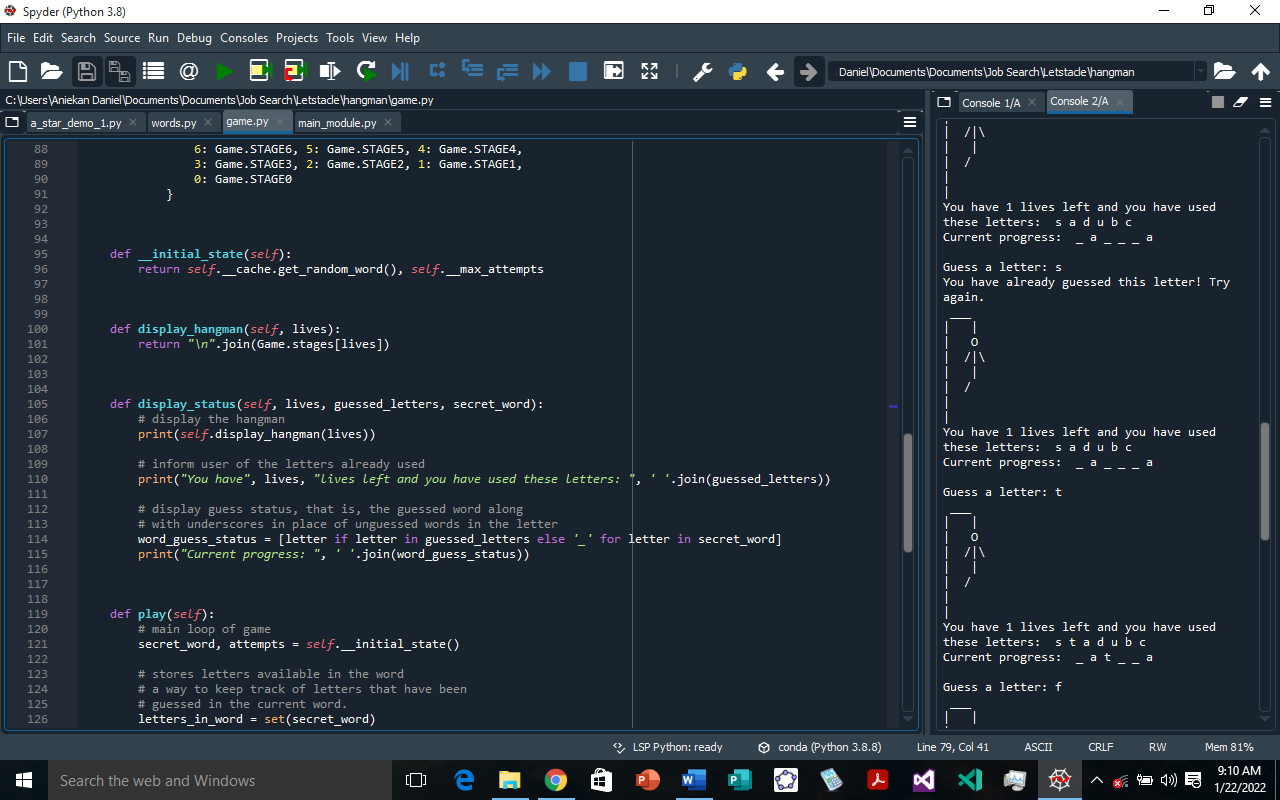
1. **The maximum number of attempts allowed**: The hangman **Game** class also defines the ***self***.**\_\_max\_attempts** instance attribute. This attribute stores the maximum number of attempts allowed before the game is lost. This attribute is set to a value of 6, and does not change throughout the game play.
2. **The alphabet**: The purpose of the attribute **self.\_\_alphabet** is to provide a way to restrict the kind of characters the player can enter while guessing the letters of the secret word. The attribute stores the 26 letters of the alphabet in lowercase as a set of characters.

#### **Methods**

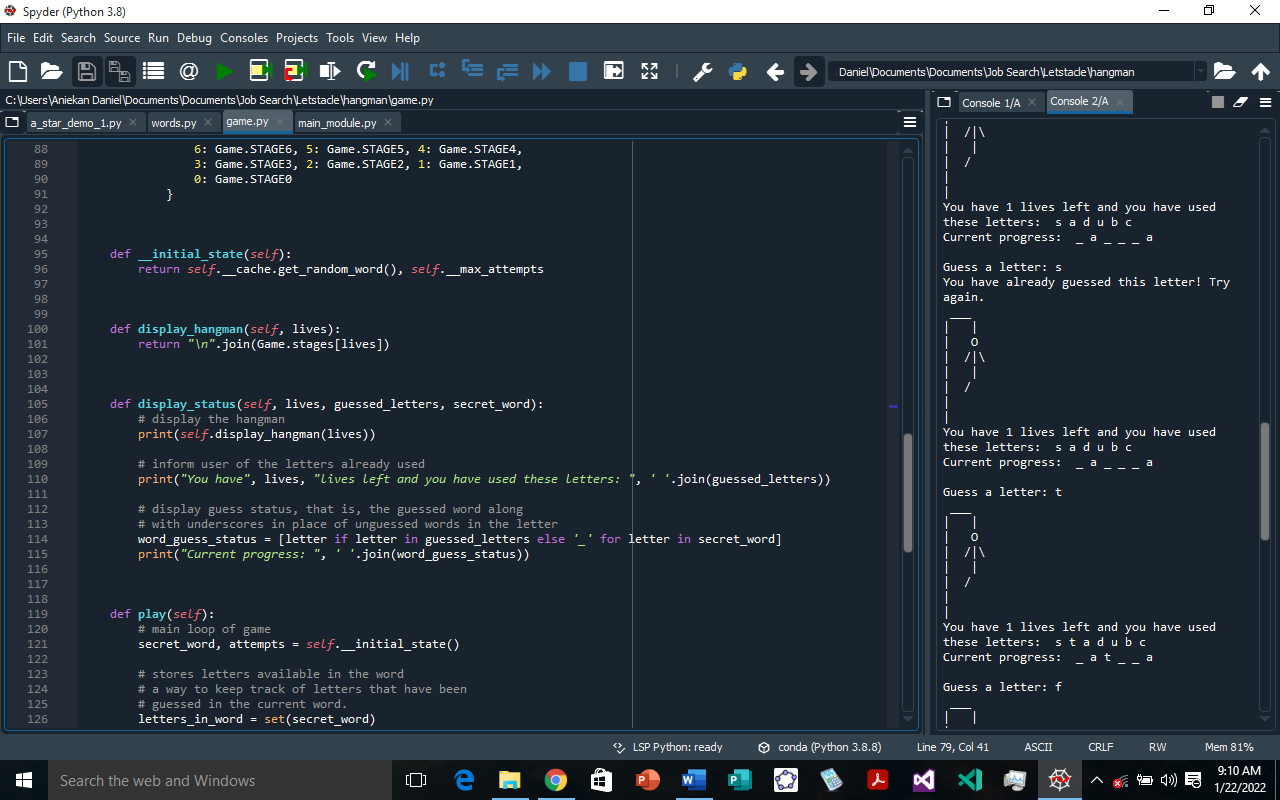
1. ***self***.**\_\_initial\_state()**: It returns a tuple, (***random word***, ***max attempts***), to the ***self***.**play()** method to help initialize the hangman game play.



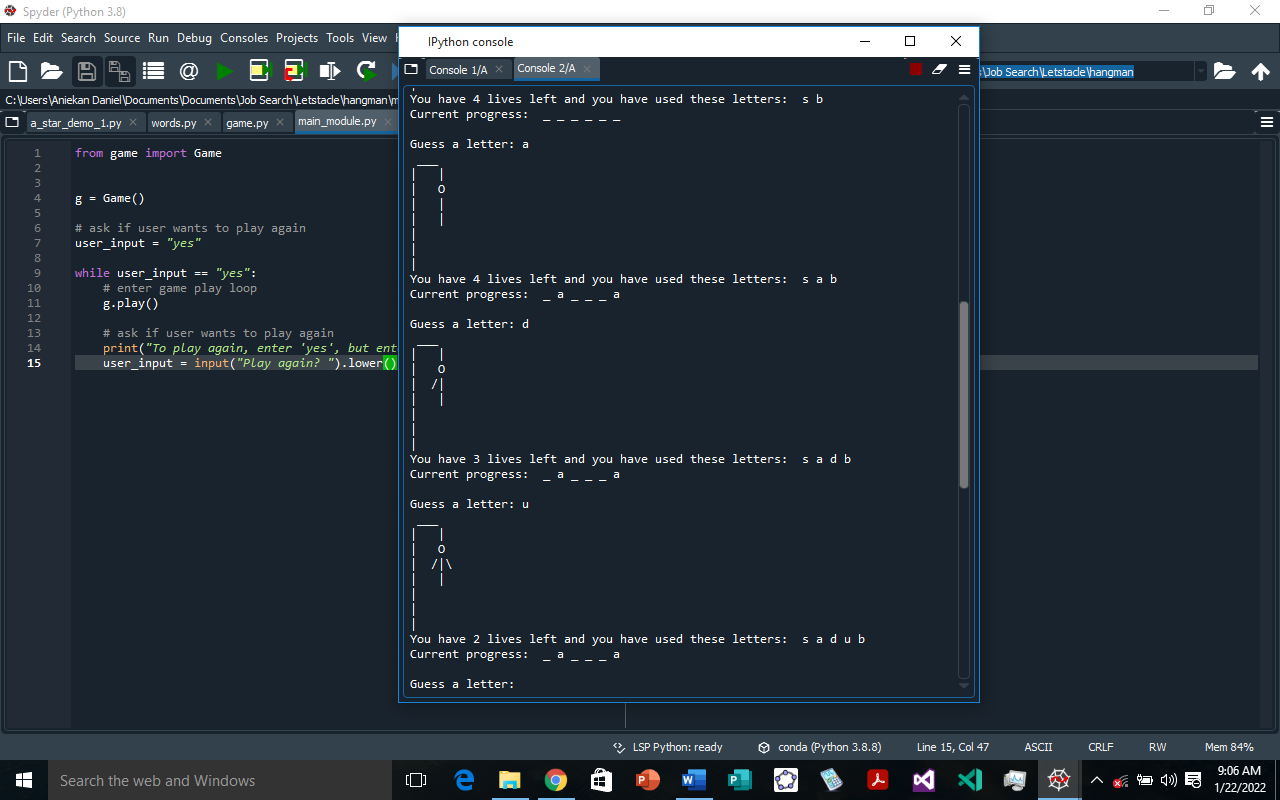
1. ***self*.display\_hangman(*self*, lives)**: This method simply accesses the **Game.status** dictionary using the current lives (or, remaining attempts) as the key. It then returns the corresponding hangman text-image.



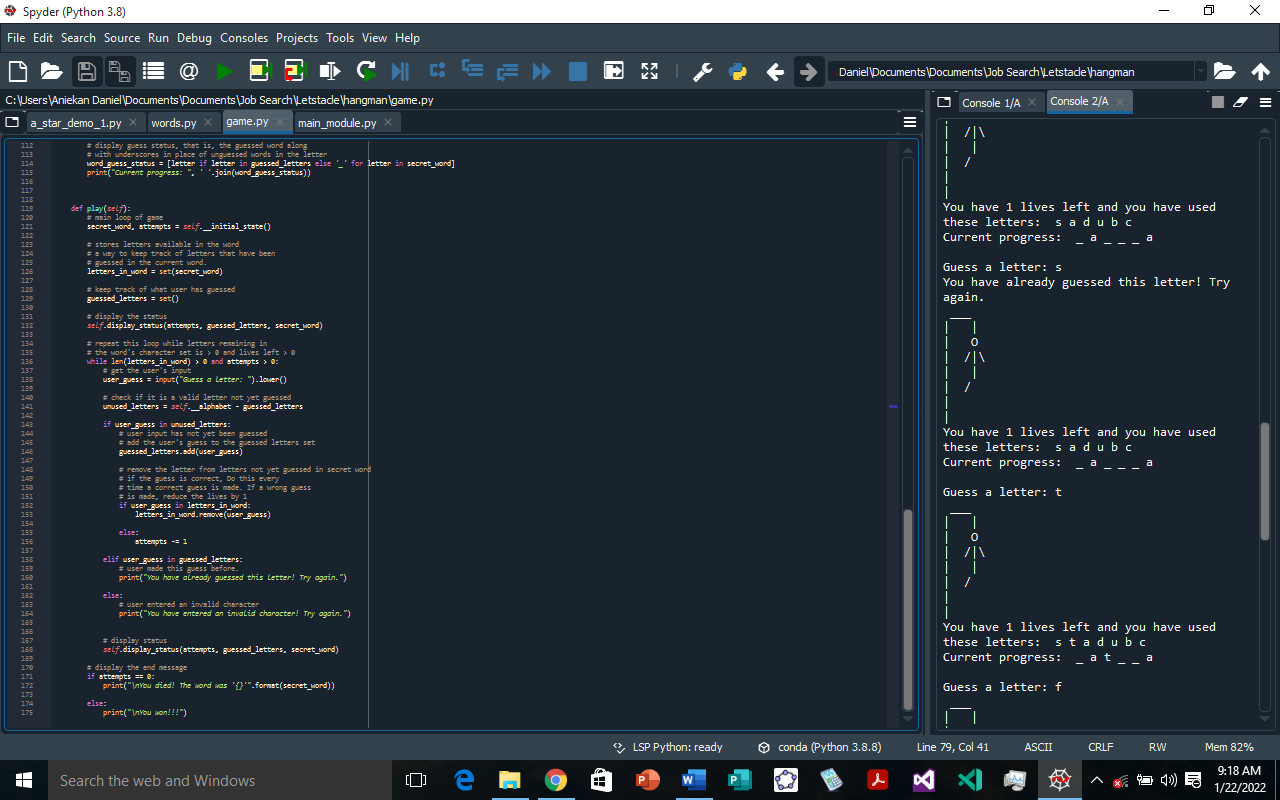
1. ***self***.**display\_status(*self*, lives, guessed\_letters, secret\_word)**: This updates the display with the hangman text-image, the lives/attempts left, the already guessed letters in the secret word.



An example of the display will look as follows:



1. ***self***.**play(*self*)**: This method is where the hangman game loop is controlled.



The previous screenshot shows the code for the **play()** method. The main part of this method is the **while** loop, which keeps running if there are still some letters in the word that have not yet been guessed and the remaining attempts is more than zero.

The **while** loop first gets the player’s guess, then it checks for one of the following situations:

* + ***player’s guess*** is part of the letters of the alphabet ***not yet guessed*** by the player. Then, the ***player’s guess*** is added to the set of guessed letters. If the ***player’s guess*** is part of the ***unguessed letters*** in the ***secret word***, it is removed from the ***unguessed letters*** in the ***secret word***, otherwise the ***remaining attempts*** is reduced by one.
  + ***player’s guess*** is part of the letters of the alphabet ***already guessed*** by the player. Then, it informs the player that his guess is already made.
  + ***player’s guess*** is ***not part*** of the ***letters of the alphabet***. This situation is the **else** part of the **if** statement, and implies that the player either entered a ***whitespace***, or an ***uppercase letter***, or a ***special character***. The player is informed that his guess is an invalid character.

After the **while** loop is exited, a message is displayed to the player based on whether all the letters of the secret message were correctly guessed, or all the attempts were exhausted.

## **Conclusion**

That’s it for our game of hangman. You could think of other ways this our simple hangman game could be extended. For example

* The **WordCache** **\_\_load\_file()** method could be modified to allow it to load larger text files
* Allow the player to also ***enter the full word***, not just a letter in the word
* Allow the player to specify difficulty. For example, the ***lesser the allowed attempts***, the more the difficulty. Also, a ***bigger value of minimum word length*** means a more difficult game of hangman
* Print a ***welcome message***, maybe also asking the player for a ***name***, then using that name throughout the game
* Create a ***GUI version*** of the game

I hope you had fun while learning how to create hangman in python. Thank you for reading to the end, and don’t forget to continue building your skills by creating more fun projects!