Lab07

COMP 125 Programming with Python

Recording Disclaimer

The synchronous sessions are recorded (audiovisual recordings). The students are not required to keep their cameras on during class.

The audiovisual recordings, presentations, readings and any other works offered as the course materials aim to support remote and online learning. They are only for the personal use of the students. Further use of course materials other than the personal and educational purposes as defined in this disclaimer, such as making copies, reproductions, replications, submission and sharing on different platforms including the digital ones or commercial usages are strictly prohibited and illegal.

The persons violating the above-mentioned prohibitions can be subject to the administrative, civil, and criminal sanctions under the Law on Higher Education Nr. 2547, the By-Law on Disciplinary Matters of Higher Education Students, the Law on Intellectual Property Nr. 5846, the Criminal Law Nr. 5237, the Law on Obligations Nr. 6098, and any other relevant legislation.

The academic expressions, views, and discussions in the course materials including the audio-visual recordings fall within the scope of the freedom of science and art.

A word guessing game: Hangman

Let’s program the word guessing game Hangman.

- For this exercise, we will input the **word** to be guessed directly into the code and program the rest of the game. The **word** should have at least 5 at most 10 letters, but no need to check for this.
- The player tries to guess this **word** by entering one letter at a time.
- The input has to be a single character and can be lower or upper case.
- If the word contains this letter, then the matching characters are shown to the player.
- If the word does not contain this character or if a previously selected letter is picked again, increment the number of wrong guesses by one.
- The player wins the game if he/she guesses the word with less than **N** wrong guesses (where **N** is the length of the word to be guessed).

Implement Hangman by using the helper functions described below

get_guess()

Implement function get_guess.

This function gets no parameters and returns the single letter input received from the user in lower case format.

The input should be a single character and it can be lower or upper case.

If a wrong input is received, it should ask for a new input, until a valid input is provided.

Sample run:

```
In [63]: letter = get_guess()

Which letter?:
1

Has to be a single letter a-z. Which letter?:
*

Has to be a single letter a-z. Which letter?:
P

In [64]: print(letter)
p
```

match(word, current, letter, wrong_guess)

Implement function **match.**

This function should receive the word to be guessed (**word**), the current state of the player's guess (**current**), the new input from the user (**letter**), and number of wrong guesses (**wrong_guess**).

current variable should be set to a string "_"*N (e.g. if word="hangman", current="_____" (seven "_" characters). **current** should be updated by replacing the "_" characters with the correctly guessed letters.

wrong_guess keeps track of the number of wrong guesses by the user. If the user picks a letter that does not exist in **word** or if the letter was already used and marked as existing in **word**, **wrong_guess** should be increased by one.

The function returns the updated state of the player's guess (**new_current**) and number of wrong guesses (**wrong_guess**)

Sample runs for demonstrating the match function are on the next slide

Demo of **match** function

Case 1: If the **letter** does not exist in word, **current** remains the same, **wrong_guess** is updated

Case 2: If the **letter** exists in word, **current** is updated, **wrong_guess** remains the same

Case 3: If the **letter** was already guessed before, **current** remains the same, **wrong_guess** is updated

```
In [98]: word='hangman'
    ...: current='_______'
    ...: letter='p'
    ...: wrong_guess= 0
    ...: current, wrong_guess = match(word, current, letter, wrong_guess)
    ...: print('current= ',current, 'wrong_guess= ',wrong_guess)
current=  _______ wrong_guess=  1

In [99]: letter='h'
    ...: current, wrong_guess = match(word, current, letter, wrong_guess)
    ...: print('current= ',current, 'wrong_guess= ',wrong_guess)
current=  h______ wrong_guess=  1

In [100]: letter='a'
     ...: current, wrong_guess = match(word, current, letter, wrong_guess)
     ...: print('current= ',current, 'wrong_guess= ',wrong_guess)
current=  ha___a_ wrong_guess=  1

In [101]: letter='h'
     ...: current, wrong_guess = match(word, current, letter, wrong_guess)
     ...: print('current= ',current, 'wrong_guess= ',wrong_guess)
You have already guessed that letter
current=  ha___a_ wrong_guess=  2
```

Main function

In the main part of the program, by using the helper functions **get_input** and **match** implement the Hangman game.

word should be set in the main part. For demo purposes let's set **word='hi'**

Initialize the game by printing a welcome message and information about the length of the word:

Finish the game if the user guesses the word successfully or if wrong_guess = N (length of word).

```
In [108]: main()
Welcome to a game of Hangman
The word contains 2 letters.

Which letter?:
h
So far you have: h_
You made 0 wrong guesses.

Which letter?:
i
So far you have: hi
You made 0 wrong guesses.

Congratulations, you win!!!
```

```
In [111]: main()

Output from spyder call 'get_namespace_view':
Welcome to a game of Hangman
The word contains 2 letters.

Which letter?:
p
So far you have: __
You made 1 wrong guesses.

Which letter?:
o
So far you have: __
You made 2 wrong guesses.

You have made  2 wrong guesses.

The word was:  hi
```