# Word Game

**Goals:**

The goal of this lab assignment is to revise some notions that you have seen up to now through realizing a word game!

If you have any question about the syntax or how to write a part of your code, please use the "memo", the Python documentation or any other resource.

For this course and its practical parts, you will use the PyCharm Edu Integrated Development Environment (IDE). Thus, your first work here is to create a new project named `SIP_WORDGAME` in PyCharm Edu.

## 1 Clean code

### Exercise 1 : Understanding the code

Please tell what the following code does (you can of course execute it).

```python
import random

AA = ("python", "information", "systems", "programming",  "centralesupelec")
a = random.choice(AA)
b = ''
c = 10

while c > 0:
    d = 0
    for e in a:
        if e in b:
            print (e)
        else:
            print ("*")
            d += 1
    if d == 0:
        print ("You won")
        break
    f = input("guess a character:")
    b += f
    if f not in a:
        c -= 1
        print ("Wrong, you have ", + c,  "more guesses")
        if c == 0:
            print ("You Lost, the word was: "+a)
```

### Exercise 2 : Meaningful variable names

In the original code, the variable names are uninformative, which makes the code hard to read. To be more understandable, please modify the names of the variables such that one can easily understand the program. You can also add some comments.

### Exercise 3 : Modular functions

There is no function in the original code. Please make it more modular by defining two or three functions, based on which one can reconstruct exactly the same functionality as the original one. For example, you can define two function as follows:

- one function is to check whether the word is totally guessed or not, and at the same time print the currently guessed word. Thus, this function can take in two parameters: a string which is the secret word and another string representing all guessed characters. It returns true if the secret word is guessed. Otherwise, it returns false.
- One other function is only to update the number of guesses left after each guess. It can take in three parameters: the guessed character, the secret word and the guesses left before this guess. Only when this is not a good guess, the user looses one guess. This function returns the updated number of guesses left.

### Exercise 4 : Unit tests

Please write unit tests for your defined functions. And tell what are the advantages of the modular program with respect to the original one.

## 2 Hangman word game

The above program is just a simplified version of hangman game with very few interactions with the user. Now les's see how to implement a more complicated version of this game step by step, allowing more interactions with the user.

### 2.1 Preliminary functions

### Exercise 5 : User's guessed word

Implement one function *get_guessed_word* that takes in two parameters: a string which is the secret word and another string containing all guessed characters. This function returns a masked guessed word composed of characters in the secret word that are already guessed by the user and of stars representing those that are not yet guessed. One example of the usage (one may assume that all characters in the secret word and all guessed one are lowercase):

```
>>> secret_word='centralesupelec'
>>> letters_guessed='baeltu'
>>> get_guessed_word(secret_word, letters_guessed)
*e*t*ale*u*ele*
```

Please write unit tests for the function *get_guessed_word*.

### Exercise 6 : Check guessed word

Another preliminary function is *check_guessed_word*, which also takes in two parameters: a string which is the secret word and a string containing all guessed characters. This function returns true if all the characters of the secret word are guessed. Otherwise, it returns false. You can reuse the function *get_guessed_word* defined before. One example of the usage:

```
>>> secret_word='centralesupelec'
>>> letters_guessed='baeltu'
>>> check_guessed_word(secret_word, letters_guessed)
False
```

Please write unit tests for the function *check_guessed_word*

## 2.2 User interaction

### Exercise 7 : Simple interaction

Now we consider the first interaction with the user, which is described as follows:

- before each guess, the user should know how many guesses (s)he has left and all the letters (s)he has not yet guessed (available letters). In this version, the number of guesses is decreased by one after each guess, whether it is a good one or a bad one.
- the user is asked to supply one guess (one letter per guess) at a time
- after each guess, the user is told whether the guessed letter is in the secret word or not, with the masked guessed word displayed (display correctly guessed letters and replace non-guessed ones by stars)

You can define one function *word_game* that takes in two parameters: a string representing the secret word and a positive number that is the maximum number of guesses allowed. This function terminates when the secret word is totally guessed or there is no guess left. And it returns the number of guesses effectuated by the player if the secret word is guessed. Otherwise, the negative value -1 is returned.

Suppose that the secret word is "are" and maximum number of guesses is 6, after executing the instruction " > > > *word_game*("are", 6)", the following shows the beginning part of the interaction with the player:

```
you have 6 guesses left
current available letters: abc...yz
please guess a letter: a
Excellent: a**
you have 5 guesses left
current available letters: bc...yz
please guess a letter: b
Sorry, the guessed letter is not in the secret word: a**
you have 4 guesses left
current available letters: c...yz
please guess a letter: ...
```

### Exercise 8 : Interaction with scores

To be able to calculate the score, which is based on the number of guesses realized, the value of each different letter in the secret word (higher value with less frequency), two basic functions are required.

The first function is to automatically calculate the maximum number of guesses initially allowed for a given secret word. This function, called *calculate_max_guesses*, takes in only one parameter that is the secret word, outputs the maximum number of guesses allowing each different letter in the secret word being guessed twice. One example of the usage:

```
>>> secret_word='centralesupelec'
>>> calculate_max_guesses(secret_word)
20
```

The second function *calculate_score* is to calculate the score for the user who succeeds to guess the secret word. This function takes in three parameters: the secret word, the number of guesses effectuated by the user, and the value of each letter in the alphabet (here we use the same value for the Scrabble, stored in a dictionary whose key is the letter). Besides the value of letters, an additional bonus will be added to the final score, which is the number of guesses still available when the word is guessed. One example of the usage:

```
>>> secret_word='centralesupelec'
>>> num_guesses=15
>>> value_letter={'a': 1, 'b': 3,'c': 3, 'd': 2, 'e': 1,'f': 4,'g': 2,'h': 4,'i': 1,'
    j': 8,'k': 5,'l': 1,'m': 3,'n': 1,'o': 1,'p': 3,'q': 10,'r': 1,'s': 1,'t': 1,'u':
    1,'v': 4,'w': 4,'x': 8,'y': 4, 'z': 10}
>>> calculate_score(secret_word, num_guesses, value_letter)
19
```

Please write unit tests for the function *calculate_score*

With all the functions that you have defined up to now, one can easily realize the word game that interacts with the user by giving the corresponding score when the user succeeds to guess the secret word.

## Exercise 9 : Input validation

Please modify the currently implemented version of the hangman game to be able to validate the user's input, making sure that each time the guessed letter is in the alphabet (you can use isalpha()). Otherwise, the program should display exception message but let the user continue to guess. To do this, one simple way is to define one function *validate_input* that takes in one parameter: the guessed character. This character is returned if it is in the alphabet. Otherwise, wrong information is displayed and the user is asked again to enter another character. Once this function is defined, then it is enough to replace "input()" in the *word_game* function by "*validate_input*(*input*())".

## Exercise 10 : More choices

Please further modify the hangman game such that before each guess, the user has two choices:

- (s)he chooses to guess only one letter each time.
- (s)he decides to guess the whole secret word at one time: if the user succeeds to guess the whole secret word at one time, then (s)he can get the bonus as the maximum number of guesses initially allowed for the user; otherwise, the user lost the game immediately.

## 3 Hangman AI game

If you want to implement this game in a more intelligent way, now let us consider automated versions of the Hangman game.

## Exercise 11 : Random automated game

The first version of the automated game is about randomly choosing a letter from the set of available letters (letters that are still not guessed) for each time. You should change you previous version by modifying the part of input, i.e., a guessed letter is randomly selected by the machine instead by the user.

## Exercise 12 : Intelligent automated game

The random strategy is of course not a good one to guess the secret word. One better and intelligent strategy is described as follows:

1. Each time before guessing, calculate the probability of each letter (the number of occurrences of a letter with respect to the number of occurrences of all letters in the dictionary) that is not yet guessed based on a dictionary (download dic.txt from edunao page). You can implement an additional function for this.
2. In the function to calculate the probabilities, to have a better performance, each time one can refine the dictionary by for example only keeping the words compatible with the currently guessed word: such as with the same length, with exactly the same letter in the same position for each already correctly guessed letter.
3. Modify the input by intelligently choosing the letter with the highest probability instead of by randomly selecting it from the available letters.

To use this intelligent strategy, the secret word must be included in the dic.txt.