Python Bootcamp {2019}

PROJECTS FINAL WEEK

PROJECT 8:

In this project, you will create a Class that is named **Logic** and your **filename** will be name logic_lib.py. Your class must contains all the below methods with the same prototype:

- **rec_sum(self, list):**

⇒ This function take a list of integer in parameter and return the sum. Your must do it using recursivity only (you cannot simply sum all the numbers. If there are invalid elements or the list is empty, your method will **return 0**.

- **rec_fact(self, number):**

⇒ This function take a valid and positive integer in parameter and return the factorial value. You must do it using recursivity only. If the number is not valid you function you will **return 0**.

- **rec_revstr(self, string):**

⇒ This function take a string and return a reversed string. To do it you cannot use any function to help you and you must do it using recursivity only. If the string is not valid or empty you will **return 0**.

- **anagram(self, word, other):**

⇒ This function take two string in parameters, word and other. This function return **True** if the words are valid anagram (example “abc”, “cab” will return **True** / “def” “dee” will return **False**) You cannot use any function for this exercise.

- **rec_anagram(self, word):**

⇒ This function take one string in parameter and return a list of word with all the anagram possible. You must use recursivity to do it and your returned array must be sorted alphabetically. Example: **“abc”** will return **[“abc”,”acb”,”bac”, “bca”, “cab”, “cba”]** If word is not a valid string or the length equal 0, you will **return 0.**

Requirements :

- Program name : **logic_lib.py**
- Directory : **week04/projects**
- Class name : **Logic**

Python Bootcamp {2019}

PROJECT 9:

You will create a class that is named Maze and contains theses methods:

- **check(self, maze):**

⇒ This function take a maze in parameters and return if the maze is valid or not. To be valid, a maze must be a 2D array that contains only wall '#', free space ' ', and one mouse only 'm'.

Also, the 2D Array must be a valid rectangle (each row must be the same size and each column must be the same size, row and columns don't need to be the same size)

If the Maze is valid you will **return True**
Else you will **return False**

- **solve(self, maze):**

⇒ This function take a maze in parameters and return the shortest number of move to escape for the mouse (m) to escape the maze. If it's not possible to escape, you will **return 0**.

Example:

```
Maze().solve([   '# ##',
                 '# m#',
                 '####']
```

In this example the mouse can leave in 3 moves (LEFT,UP,UP) : **Return 3**

```
Maze().solve([   '####',
                 '# m#',
                 '####']
```

In this example the mouse cannot leave. **Return 0**

Python Bootcamp {2019}

- **generate(self, solvable, height, width):**

⇒ This function will generate and return a maze. 'solvable' is a boolean that will represent if the Maze can be solved or not (**True/False**). 'height' will represent the number of row, 'width' the number of columns.

Example:

```
Maze().generate(True, 3, 4)
```

Could return something like that :

```
['# ##',
 '# m#',
 '####']
```

```
Maze().generate(False, 3, 10)
```

Could return something like that :

```
['##########',
 '#       m#',
 '##########']
```

You must use random function for your generate methods so the output will always be different. Also I suggest you to use your three function (generate/check and solve) to create unit_test and print all the output for debugging faster.

Requirements :

- Program name : **maze_solver.py**
- Directory : **week04/projects**
- Class name : **Maze**

Make sure that all the methods are prototyped the same way as expected and make a lot of test before submitting. Good luck!