Programming Assignment 07
-
Lists

Instructions

This programming assignment consists of **2 programming exercises**.
You have to:

1. **download** the empty Python files on NYU Classes
2. **edit** them according to the assignment
3. **verify** on your computer that it works
4. **upload** them back on NYU Classes (**do not change the filenames**)

Exercise 1 - Remove Special Usernames

Write a function called `clean_users` (in the file `exercise1.py`) that does the following:

Function `clean_users(L)`

- takes 1 parameter `L` → type `list` (of strings): each element in the list is a username
- returns a new cleaned list (see the rules below) → type `list` (of strings)

The logic for cleaning the list of usernames is:

- if the username contains `c`, `g` or `z`, it should be removed (the same with `C`, `G` or `Z`)
- otherwise, keep it in the list

Note: your function should not modify the original list!

Then, **write a program** (in the `main()` in the file `exercise1.py`) that does the following:

Program `main()`

1. asks the user to **input 5 usernames** (only alphanumeric characters will be input, no space, no underscore, ...) and **stores them into a list**
2. **prints out** the list of usernames
3. **calls** `clean_users` function to clean the list
4. **prints out** the cleaned list

Note: no need to check if the user inputs incorrect usernames or inputs duplicated usernames, you still have to create an entry in the list for each username input by the user

Sample examples (the user input is in red, the printed output is in blue):

```
username: hexnumber
username: someone
username: waitFor2100
username: zeroBalance
username: TTT
['hexnumber', 'someone', 'waitFor2100', 'zeroBalance', 'TTT']
['hexnumber', 'someone', 'waitFor2100', 'TTT']
```

```
username: qwerty
username: azerty
username: ICPforever
username: user00
username: abcdefghij
['qwerty', 'azerty', 'ICPforever', 'user00', 'abcdefghij']
['qwerty', 'user00']
```

Exercise 2 - Username / Password

The file `exercise2.py` is a module that offers functions for **managing a list of users and their passwords**.

The usernames and passwords are stored in a **list of lists** that we will name `userList`. Each inner list represents a username/password pair in that order `[username, password]`.

The module `exercise2.py` offers the following functions:

Function `valid_username(user, L)`

- takes 1 parameter `user` → type `str` : username
- takes another parameter `L` → type `list` (of lists): `userList`
- **displays** a message (`Valid` , `Invalid` or `User Name Exists`) depending on username rules (see below)
- returns if the username is valid → type `bool`

Valid username rules are:

- The username is made of at least 4 characters
- It only contains alphanumeric characters (no space, no underscore, . . .)
- Case-sensitive
- The first character cannot be a digit
- The username does not exist in the `userList`

The **printed out message/returned value** will be:

- `Invalid` / `False` if the format is not correct
- `User Name Exists` / `False` if the username already exists
- `Valid` / `True` if all the rules are respected

Hint: the methods `.isalpha()` , `.isdigit()` and `.isalnum()` can be useful

Function `valid_password(pwd)`

- takes 1 parameter `pwd` → type `str`: password
- **displays** a message (`Valid` or `Invalid` depending on password rules (see below)
- returns if the password is valid → type `bool`

Valid password rules are:
- The password is made of at least 10 characters
- It only contains alphanumeric characters (no space, no underscore, ...)
- There is at least:
  - 1 uppercase character
  - 1 lowercase character
  - 1 digit

The **printed out message/returned value** will be:
- `Invalid` / `False` if the password is not correct
- `Valid` / `True` if all the rules are respected

Function `add_user(L)`

- takes 1 parameter `L` → type `list` (of lists): `userList`
- adds a username/password pair to the `userList`

The function should:
1. continuously ask the user to input a username, until it is a valid one
2. ask for a password with the following logic:
   (a) continuously ask the user to input a password, until it is a valid one
   (b) ask the user to input the password again:
      - if the 2 passwords match → move to 3.
      - otherwise → display `Passwords do not match` and go back to 2.(a)
3. add the username/password pair to the userList and display `User created`

Write the code of the functions in the file `exercise2.py`:
- `valid_username`
- `valid_password`
- `add_user`

Below is an example when using the functions directly from the shell (the user input is in red, the printed output is in blue and returned values from the function calls or variables are in green):

```
>>> # Initialization of a userList
>>> userList = [['sunny1', 'pwd1DdeEff'], ['superS', 'pwD2Abcdefgh'],
['likeA', 'pwd3AAAAAA'], ['qwerty', 'pwd4QWERTY']]
>>> valid_username('likeA', userList)
User Name Exists
False
>>> valid_username('user', userList)
Valid
True
>>> valid_password('azertyuiop')
Invalid
False
>>> valid_password('12345AZERTYuiop')
Valid
True
>>> add_user(userList)
Enter Username: user_1
Invalid
Enter Username: sunny1
User Name Exists
Enter Username: ICP2018user
Valid
Enter Password: ABcd1
Invalid
Enter Password: STRONGpassword321
Valid
Confirm Password: STR
Passwords do not match
Enter Password: STRONGpassword321
Valid
Confirm Password: STRONGpassword321
User created
>>> userList
[['sunny1', 'pwd1DdeEff'], ['superS', 'pwD2Abcdefgh'], ['likeA', 'pwd3AAAAAA'],
['qwerty', 'pwd4QWERTY'], ['ICP2018user', 'STRONGpassword321']]
```