TU856/858-2 Object Oriented Programming with Python
Semester 1, Lab 2, 02.10.2020
Dr. Bianca Schoen-Phelan

Lab solutions will be discussed in the Monday lecture. Labs are not marked.
Link to the lab groups (click).

Objectives:

- Learn how to sync your forked repo
- Get to know Python data types String and List
- Use type casting

ONLINE guideline:

- We meet in the virtual classroom in brightspace first. Just like the last lab.
- You will be able to stay within your own lab group throughout this online lab. Each tutor offers a teams link. You will be required to sign a sign-in sheet for each lab group. You will work within this lab group today and in the following weeks.

Note: all `git` commands used here are further explained in the walkthrough slide set from the tutorial in week 1! All terms used, such as "fork" and "branch" are explained in the lecture slide set from week 1. New commands are explained under (1).

All string or list related functions have been introduced in the recent lecture. Open the slide set to help solve the lab.

Link to Python documentation (click).

Further tutorial on Python Data Structures (click).

1. **Synch your forked repo with the original `BiancaSP` repo to get updates**

   For this lab there is a new file in the `BiancaSP` repository on GitHub called `lab2.py`. You will need to update your local copy in order to sync with the original repository `BiancaSP` to get the new file. Follow these instructions in order to do this:

   - Open a terminal window (mac/linux)/ command prompt (windows) (or the git bash on windows) and navigate to your `OOP2020-21` directory.
   - Firstly, you will need to add the original repo `BiancaSP` as an `upstream` (see terminology in the lecture slides week 1!)
     Check what your current remote repository is:
     ```
     $ git remote -v
     ```
     Your output should look like the following, just using **your own** GitHub username instead of *`MyUserName`*:
     ```
     originhttps://github.com/MyUserName/OOP2020-21.git (fetch)
     originhttps://github.com/MyUserName/OOP2020-21.git (push)
     ```
     Then add the upstream location of the original `BiancaSP` repository as follows:
     ```
     $ git remote add upstream https://github.com/BiancaSP/OOP2020-21.git
     ```

And check that you also get the last two lines now as follows:

```
originhttps://github.com/MyUserName/OOP2020-21.git (fetch)
originhttps://github.com/MyUserName/OOP2020-21.git (push)
upstream    https://github.com/BiancaSP/OOP2020-21.git (fetch)
upstream    https://github.com/BiancaSP/OOP2020-21.git (push)
```

Notice the additional 2 lines. Observe that your own online GitHub location is called “origin” and my GitHub repository for the class module is now called “upstream”. These two names are aliases for the online locations on the right. So, upstream is an alias for `https://github.com/BiancaSP/OOP2020-21.git` and origin is an alias to your own GitHub location online, which can be found under `https://github.com/MyUserName/OOP2020-21.git`, where `MyUserName` is replaced by your own user name.

Now you can just get everything that might have changed on the original class repository `BiancaSP` over to your **local** *`MyUserName`* repository whenever there is new content.

So, let’s get the file for lab2 as follows:

```
$ git fetch upstream
```

You should see output similar to the following appear:

```
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/BiancaSP/OOP2020-21
 * [new branch]      master     -> upstream/master
```

This tells you that there is a new file. Use the following command to integrate the file with your local repository on your computer:

```
$ git merge upstream/master
```

You should see output similar to the following appear:

```
Updating 4ec5365..59b0a7d
Fast-forward
 Labs/lab2.py | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 Labs/lab2.py
```

Now make sure that you are working on the branch `master` using the following command.:

```
$ git branch
```

The current branch will have a * by its side. If you are not on master, switch to master now using `git checkout master`.

Now your **local** repository is all synched to the original remote repository `BiancaSP` and all set up to fetch future updates. However, what is missing is that your **online** GitHub repository receives the change as well. Go to your

GitHub account in the browser and see that the new file does not yet appear there. You now push your **local** change to **our** remote repository:

```
$ git push origin master
```

And you should see output similar to the following:

```
Total 0 (delta 0), reused 0 (delta 0)
To https://github.com/TheBianca/OOP2020-21.git
   197a014..2c32dab  master -> master
```

Reload your GitHub repo in the browser and see that the new file is there now too.

- Note that the terms “origin” and “upstream” are conventions that are commonly used in the git / version control world.

2. **Run the program**
   - Open Pycharm
   - Click on open project, then navigate to the **root folder** holding your project, not any of the separate files! Click on the folder and choose open.
   - Open the file for lab 2 and read through it
   - Run the file
   - You will find that the programme asks you to enter a name, which you should do in the bottom half of Pycharm
   - After pressing enter the program will print what you originally entered and terminate after
   - Check how this is done in the code. You will fine that `input()` asks the user for keyboard input

3. **Extend the program working with strings. The file for lab 2 contains Python comments for the location where you should add your code. The code should be written underneath the comment.** The file contains one class and two methods within the class. At the bottom we create an instance of the class and call a method with the dot notation. We will discuss classes very soon. Observe how you are intuitively able to read the code. Check the name of the variable that holds the input (called message).
   - Using the index notation for strings, print the first and the last character from your input word. Hint indexes are referred to with square brackets. In Python index locations start with 0 (not with 1) for the first position. The last position is indexed with -1.
   - Using the slice notation, print only a section of your string. Also experiment with shorthands, such as `[:4]` or `[2:]` or `[:]`
   - Aim to print the following sentence `He said “that’s fantastic”!` You will need to **escape** certain elements of this sentence in order to print. Escaping a character means that we remove its special meaning. For example, `\n` means new line and `\t` means tab, and `\’` escapes the apostrophe.
   - Use the `find()` method to find the position of the first character `a` in your input word and `count()` how many a’s you find in the input word. For this first make sure that all characters in the string are lower case characters. In

order to generate a print out of your result you will find that you will have to cast the integer results of `find()` and `count()` via the `str()` function. Also print how many characters there are in total.

- Replace all characters a with the - sign. Try this first by assigning the new value via the index location notation such as `my_input[4]='-'`. What happens and why? Then use the `replace()` method.

4. **We are now going to work with lists. The file for lab 2 contains Python comments for the location where you should add your code. The code should be written underneath the comment.**

   - Comment out the line that says `tas.play_with_strings()` and uncomment the line that says `tas.play_with_lists()`
     i. Get used to the key shortcuts to toggle comments in Python. The keyboard shortcut is always shown on the right in the menu item "Code" > "Comment…"
   - Run the program now and enter a sentence and see how the program currently just returns the sentence you inputted before. Now it's time to work with lists:
   - `split()` up your string entry from `input()` into its individual words and assign it to the `list` type. You will need to cast to the list type using list(....) You have used type casting earlier in the section on strings. You can cast a string to a list using list(...) as follows:

     ```
     my_list = (list(message.split(" "))
     print(my_list)
     ```

     Print this new `my_list` variable out. Every word has an index location in the list. Try with "Hello World" as message. Then Hello is index 0 in the list, and World is index 1 in the list.
   - `append()` a new element to your list
   - Take away the last element in the list. In class we looked at three different ways of doing this. Experiment with all three. What happens for `remove(...)` if the item doesn't exist in the list?
   - Reverse the items in your list and print. The implementation of Python does reverse the **original** list item. This means the return value is None. You have to print it out separately. This implementation was chosen to save space on very large list items. Check what happens if you do everything in one line vs printing in the next line after reversing. You could achieve the same result with the `[::-1]` slicing notation. Try it out!
   - Print the list 3 times by using multiplication with `*3`

5. **Commit the changes you have made to your lab file and then push it to origin.**

   ```
   $ git add Labs/lab2.py
   $ git commit -m "my lab2 solution" Labs/lab2.py
   ```

Check your work with the following command, which should now show your new solution as a commit history item, similar to the following output (your hash codes on the left will be specific to your own computer):

```
$ git log --oneline
```

Output should be similar to the following:

```
f08c565 (HEAD -> master) my lab2 solution
1cd04c3 original file lab 2
6ff1dbf File Lab 1
```

Push to your own GitHub repository as follows:

```
$ git push origin
```

Check in your browser that your own lab solution is now visible in your own GitHub account online.

Well done!