Assignment 06
Due: Monday, Nov 16th, 2020 at 10:30 am.

General Assignment Notes

Your assignments must follow the following requirements:

- Name your Eclipse project with:
  - your login,
  - an underscore,
  - 'a' (for 'assignment'),
  - the assignment number: `login_a#`.
- For example, if student `barn4520` submits Assignment 1, the name should be: `barn4520_a1` or `barn4520_a01`. Make sure your programs and the testing file are in this Eclipse/PyDev project.

- In your program, use the variable naming style given in Coding and Documentation Style Standards : i.e. lower case variable names, underscores between words.

- Test your programs:
  - Copy the output from your test to a file in your Eclipse/PyDev project named `testing.txt`.
  - Make sure that you have included an identification header for each question; there was a sample identification header for the testing file in lab 1.
  - Make sure you do the correct number of tests as specified in the question.
  - Make sure the tests are well labeled so that the markers know how the results match the questions.
  - The solutions for all programs go into *one* `testing.txt` file.

- Zip the entire project using Eclipse.
  - Give your .zip file the same name as your project when exporting your project, e.g. `barn4520_a1.zip` .
  - Use only Eclipse's built-in archive capability to create these .zip files. No other format will be accepted.

- Use the Validate Assignment link to make sure that your project and .zip file are named correctly and have the proper contents. Improper assignment submissions are given a grade of zero.

- Submit the validated .zip file to the appropriate drop box on MyLearningSpace.
  - You can submit as many times as you like. Only the last submission is kept.
- Unless otherwise indicated by the question you may only use the built-in functions

and special forms introduced in the lecture slides
- The solutions you submit must be entirely your own work. Do not look up either full or partial solutions on the Internet or in printed sources.

General Marking Expectations

- Although the marking scheme is tailored for each question, you can use the following as an indication of what we are looking for when marking your assignments.
    - General:
        - project and zip files named correctly. Using the wrong names is an automatic zero.
        - uses the variable naming style given in Coding and Documentation Style Standards : i.e. lower case variable names, underscores between words
    - main:
        - identification template included and filled in correctly
        - inputs as required for program
        - outputs as required for program
    - testing:
        - identification header included and filled in correctly
        - Tests as required for program, e.g. number of tests, types of tests.

Note: Refer back to Lab 1 for help in the process of submitting an assignment.

Assignment 6 marking Expectations:
- All programs should use the formatted output method taught in lab 2.
- Define constant when appropriate.
- The output for all of the questions in an assignment go into one testing.txt file (as the template shown in lab 1 task 3). Make sure the tests are labeled so that the markers can easily find the answers to a particular question.
- function(s):
    - **FUNCTIONS MUST BE IN A SEPARATE FILE FROM THE MAIN PROGRAM.**
    - for questions that specify the objectives of the function(s), did you do what was requested?
    - correct parameters and return values
    - calculation(s) correct
    - function documentation has correct format
    - preconditions correct and complete
    - postconditions correct and complete
- main:
    - identification template included and filled in correctly

Assignment 06
Due: Monday, Nov 16th, 2020 at 10:30 am.

  - proper use of constants if appropriate
  - inputs as required for program
  - outputs as required for program
  - outputs should use the formatted output method
  - implementation of decision structures, for loops and while loops, as appropriate
  - functions calls have appropriate arguments
  - return values from functions are appropriately handled
- testing:
  - identification header included and filled in correctly
  - tests as required for program, e.g. number of tests, types of tests.
- **Provide the docstring for all questions**

1. Write a function called `read_positive` that takes no parameters and asks the user for a series of numbers and returns a list of numbers where only the positive numbers are kept. The negative numbers are invalid inputs, if the user enters a negative number the function should keep asking for positive number until the user enters one. The user indicates that they are finished entering numbers by entering number 0. You may assume that the user will only enter numbers. Save the functions in a PyDev library module named `functions.py`

   A sample run,

```
Enter a positive number: 1
Enter a positive number: 30
Enter a positive number: 40
Enter a positive number: 50
Enter a positive number: -5
Enter a positive number: -1
Enter a positive number: 5
Enter a positive number: -5
Enter a positive number: 0

List entered: [1, 30, 40, 50,5]
```

   Write a program named **t01.py** to test the function.
   - Copy the results to testing.txt.

2. Write a function called `find_target` that finds the locations of a target value in a list. Your function should have two parameters a list of numbers and a target number to search. The function returns the list of the locations of targeted number. If the target value does not exist in the list, the function should return an empty list.
   Write a program named **t02.py** that tests the function. In your program call the function `read_positive` from t01 to get a list of positive numbers from the user and pass this list along

with a target value to `find_target` function. Your program should display the returned list on the screen. Save the functions in a PyDev library module named `functions.py`

A sample run,

```
Enter a positive number: 9
Enter a positive number: 11
Enter a positive number: 32
Enter a positive number: 17
Enter a positive number: -5
Enter a positive number: 9
Enter a positive number: 33
Enter a positive number: -5
Enter a positive number: 8
Enter a positive number: 0

List entered: [9, 11, 32, 17, 9, 33, 8]
target = 9
target exists at location [0,4]
```

- Test your function with a list that contains the target and a list that does not contain the target
- Copy the results to testing.txt.

3. Write a function called `largest_odd` that takes a list of integers as a parameter. If the list has no odd integers, your function should return `-1`. Otherwise, it should return the largest positive odd integer in the list. Note: 0 is an even number. Save the function in a PyDev library module named `functions.py`

   Write a program named **t03.py** that tests the function. In your program call the function `read_positive` `from` **t01** to get a list of positive numbers from the user and pass this list to the `largest_odd` function.
   A sample run,

```
Enter a positive number: 9
Enter a positive number: 11
Enter a positive number: 32
Enter a positive number: 17
Enter a positive number: -5
Enter a positive number: 9
Enter a positive number: 33
Enter a positive number: -5
Enter a positive number: 8
Enter a positive number: 0
```

```
List entered: [9, 11, 32, 17, 9, 33, 8]
Largest odd: 33
```

- test your function with a list that have even and odd numbers and a list that has no odd numbers
- Copy the results to testing.txt.

4. Write a function called `reverse_list` that takes a list as a parameter and mutate it such that all the elements in the list are reversed. The function does not return any value. Save the function in a PyDev library module named `functions.py`.

   Write a program named **t04.py** that tests the function. In your program call the function `read_positive` `from` **t01** to get a list of positive numbers from the user and pass this list to the `reverse_list` function.
   A sample run,

```
Enter a positive number: 9
Enter a positive number: 11
Enter a positive number: 32
Enter a positive number: 17
Enter a positive number: -5
Enter a positive number: 9
Enter a positive number: 33
Enter a positive number: -5
Enter a positive number: 8
Enter a positive number: 0

List entered: [9, 11, 32, 17, 9, 33, 8]
List reversed: [8, 33, 9, 17, 32, 11, 9]
```

   Note, you need to use loops (you may not use the built-in Python reverse function or the slicing operator **:**)

   - Hint: you need to use a third variable to do the swap
   - Copy the results to testing.txt.