**Assignment 07**
**Due: Monday, Nov 23rd, 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 7 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
    - parameters correct and complete
    - returns correct and complete
- main:
    - identification template included and filled in correctly

## Assignment 07

**Due: Monday, Nov 23rd, 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 that does not include one.**

1. The local driver's license office has asked you to create an application that grades the written portion of the driver's license exam. The exam has 20 multiple-choice questions. Here are the correct answers:

| | | | |
|---|---|---|---|
| 1. A | 6. B | 11. A | 16. C |
| 2. C | 7. C | 12. D | 17. B |
| 3. A | 8. A | 13. C | 18. B |
| 4. A | 9. C | 14. A | 19. D |
| 5. D | 10. B | 15. D | 20. A. |

Your program should store these correct answers in a list.

Write a function `driver_license` that takes two lists of strings as parameters. A list with the correct answers and a list of student answers in that order. The function returns 3 values:

- the total number of correctly answered questions
- the total number of incorrectly answered questions
- a list showing the question numbers of the incorrectly answered questions

Save the functions in a PyDev library module named `functions.py`

Write a program **t01.py** that reads a list of strings, represents the students answers from the user. Your program should display a message indicating whether the student passed or failed the exam. (A student must correctly answer 15 of the 20 questions to pass the exam). You may assume that the student will answer 20 questions, no questions will have blank answers.

A sample run:

```
Enter a sequence of 20 choices separated by a space, each A-D in
Uppercase: A A A A B B B B C C C C D D D D E E E E
```

**Assignment 07**
**Due: Monday, Nov 23rd, 2020 at 10:30 am.**

```
#Correct answers = 6 ,#incorrect answers = 14
A list of incorrectly answered questions = [1, 4, 6, 7, 9, 10,
11, 12, 13, 15, 16, 17, 18, 19]
To pass the exam you need to score 15 out of 20

Or
Enter a sequence of 20 choices separated by a space, each A-D in
Uppercase: A C A A D B C A C B C D C A D C B B A A
You passed your driver license exam
#Correct answers = 18, #incorrect answers = 2
A list of the incorrectly answered questions = [10, 18]
To pass the exam you need to score 15 out of 20
```

- Test your program with 2 different inputs one will pass the test and one that will not pass.
- Copy the results to testing.txt.

2. Write a function called `sum_digit` that takes a string that has single-digit characters as a parameter and **returns** a number that represents the sum of all the single digit numbers in that string as in the provided docstring below. Save the functions in a PyDev library module named `functions.py`

```
def sum_digit(my_str):
    """
    ------------------------------------------------------
    sums all the single digit numbers in my_str
    use: total = sum_digit(my_str)
    ------------------------------------------------------

    Parameters:
        my_str: string that has single-digit numbers (str)

    Returns:
        total: sum of all the single digit number (integer >= 0)
    ------------------------------------------------------

    Sample run
    >>>total = sum_digit('2501')
    >>>print("{}".format(total))
    8
    """
```

Write a program **t02.py** that asks the user to enter a series of single-digit numbers with nothing separating them and pass it to the function `sum_digit`. The program should display the sum.

- Test your program with 2 different inputs.
- Copy the results to testing.txt.

## Assignment 07
**Due: Monday, Nov 23rd, 2020 at 10:30 am.**

3. Write a function called `find_frequent` that finds the character that appears most frequently in the input parameter string and returns it. For Example, if the string is "Canada day" the function returns string “a”, if there are two characters or more with the same frequency the function should return the first of many values with the same frequency. Save the functions in a PyDev library module named `functions.py`

   A sample run:
   ```
   Enter a string: Canada
   ```

   output should be
   ```
   Most frequent characters: a
   ```

   Or
   ```
   Enter a string: Hello Hi
   ```

   output should be
   ```
   Most frequent characters: H
   ```

   Write a program named **t03.py** that tests the function by asking the user to enter a string, and displays the character that appears most frequently in the string.
   - test your function with two different strings
   - Copy the results to testing.txt.

4. Write a function called `add_spaces` that takes a string as a parameter, the string represents a sentence in which all of the words are run together but the first character of each word is uppercase. The function returns a new string in which the words are separated by spaces and only the first word starts with an uppercase letter.

   For example, the string “StopAndSmellTheRoses.” would be converted to “Stop and smell the roses.”. Save the function in a PyDev library module named `functions.py`

   ```
   def add_spaces(my_str):
       """
       ------------------------------------------------------
       create a new string with added space between words
       use: new_str = add_spaces (my_str)
       ------------------------------------------------------

       Parameters:
           my_str: string that represents a sentence in which all the
                   words are run together (no spaces), but the first
                   character of each word is uppercase.
   ```

## Assignment 07

**Due: Monday, Nov 23rd, 2020 at 10:30 am.**

```
            my_str should has at least one character   (str)
Returns:
      new_str: new string in which the words are separated
               by spaces and only the first word starts with
               an uppercase character.
-----------------------------------------------------
 """
```

Write a program **t04.py** to test your function. Your program accepts as input a sentence from the user with no spaces in it, calls your function and displays the converted string with the added spaces.

- test your program with a string other than the one given in the examples.
- Copy the results to testing.txt.

5. A word chain is a list of words such that for each consecutive pair of words, the last letter in the first word is the same as the first letter in the second word. For example, the following list of animals forms a word chain: ['camel', 'leopard', 'dog', 'giraffe', 'elephant']. Write a function `is_chain` that takes a list of words(list of strings) as parameter and returns `True` if the list contains a word chain and `False` otherwise. Your list should have at least 2 words. Save the function in a PyDev library module named `functions.py`

   Write a program **t05.py** to test your function.

   - test your program with two lists of string, where one list represents a word chain and the other is not.
   - Copy the results to testing.txt.