Assignment 09

Due: Monday Dec 7th, 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 9 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

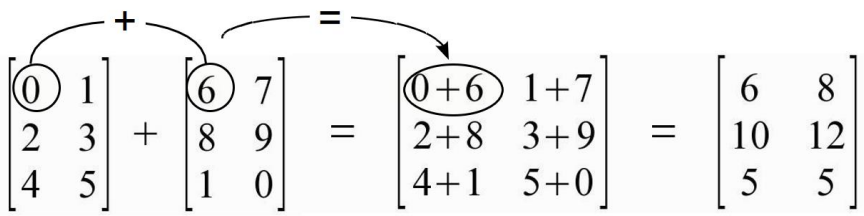
Assignment 09

Due: Monday Dec 7th, 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. Write a function called `add_2d_list` that take two 2d-lists of numbers as parameters and returns their sum(a matrix/2d-list). Here is an example of how to add two 2d-lists/matrices:



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

   Write a testing program **t01.py** that asks the user for 12 numbers on one line. Split them and put them into two 2d-lists matrix1 and matrix2 each is 3 rows by 2 columns, then call `add_2d_list` function and prints their sum.
   - Do not use python's built-in function sum().
   - Test your program with one example.
   - Copy the results to `testing.txt`.

2. Write a function called `largest_even` that takes a 2d-list of integers as a parameter and returns the maximum even value for the entire list. Your function should return a list with that number or an empty list if all the numbers in the 2d-list are odd. Save the function in a PyDev library module named `functions.py`

   Write a program **t02.py** that tests `largest_even` function
   - Do NOT use python's list method max().
   - Test your function with 3 different 2d-list, hardcoded in **t02.py**

- Copy the results to `testing.txt`.

3. Write a function `flatten_2d` that takes a 2D list and returns all the items of each list concatenated together into one new 1D list.

   For example:
   ```
   Flatten_2d ([["a","b"],["c","d"],["e","f"]])
   ```

- would return
  ```
  ["a","b","c","d","e","f"]
  ```

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

  Write a program **t03.py** that tests `flatten_2d` function and prints the returned flat list to the screen.

  - Test your program with a different list, hardcoded in **t03.py**
  - Copy the results to `testing.txt`.