Assignment 05

Due: Monday, Nov 9th, 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 5 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
  - return correct and complete
- main:
  - identification template included and filled in correctly

Assignment 05
Due: Monday, Nov 9th, 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 and for loops
  - 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. In mathematics, a square number or perfect square is an integer that is the square of an integer; in other words, it is the product of some integer with itself. (i.e 9 is a perfect square 9 = 3× 3 while 7 is not).

   Write a function called `perfect_square` that takes an integer and **prints** all the perfect square number between 1 and N (exclusive). If `num` is a negative number, your function should print an error message. Save the function in a PyDev library module named `functions.py`

   Write a main program **t01.py** to test the function by asking the user to enter a number.

   Example for input:
   ```
   Enter a positive number: 10
   ```

   The output will be:
   ```
   Perfect squares below 10 are: 1 4 9
   ```
   Or

   Example for input:
   ```
   Enter a positive number: -1
   ```

   The output will be:
   ```
   Error: you entered a negative number
   ```

   - Test your program with 2 different values for N than the example, one will cause the program to display the error message.
   - Copy the results to `testing.txt`.

2. In mathematics, the notation n! represents the factorial of the non-negative integer n. The factorial of n is the product of all the non-negative integers from 1 to n.

For example,
7! = 1 × 2 × 3 × 4× 5 × 6 × 7 = 5,040
and
4! = 1 × 2× 3 ×4 = 24

Write a function called `factorial` that takes an non-negative integer `num` as a parameter then uses a for loop to calculate and return the factorial of that number. Save the function in a PyDev library module named `functions.py`

Write a main program named **t02.py** that tests the function by asking the user to enter a number and displaying the output. **If the user enters an invalid input (negative number) the testing program should not call the function.**

Example for input:
```
Enter a positive number: 7
```

The output will be:
```
7! = 5040
```

Or

Example for input:
```
Enter a positive number: -7
```

The output will be:
```
Error: you entered a negative number
```

- Test your program with 2 different values for N than the example, one will cause the program to display the error message.
- Copy the results to `testing.txt`.

3. In mathematics, a positive integer greater than 1, which has no other factors except 1 and the number itself, is called a prime number. 2, 3, 5, 7 etc. are prime numbers as they can only be divided by 1 and by themselves to give a whole number. 4, on the other hand is not a prime because, 4 can be divided by 1,2 and 4.

   Write a function called `is_prime` that takes a positive number called `num` as parameter and returns `True` if the number prime and `False` otherwise. Save the function in a PyDev library module named `functions.py`

   Write a main program named **t03.py** that tests the function by asking the user to enter a number and displaying the returned value. **If the user enters an invalid input (negative number) the testing program should not call the function.**

Example for input:
```
Enter a positive integer number: 8
```

The output will be:
```
8 is not a prime number
```

Or

Example for input:
```
Enter a positive integer number: -8
```

The output will be:
```
Error: you entered a negative number
```

- Test your program with3 different numbers:
  - Prime number
  - non-prime number
  - invalid input that will cause an error message to be displayed
- use while loop only. Using `break/continue` is not allowed.
- Copy the results to `testing.txt`

4. Write a function called `print_pattern` that takes a positive integer number `num_rows` as a parameter and **displays** the following pattern where it has `num_rows` rows. Save the function in a PyDev library module named `functions.py`

Example for input:
```
Enter a positive integer number: -1
Enter a positive integer number: 8
```

The output will be:

```
##
# #
#  #
#   #
#    #
#     #
#      #
#       #
```

Assignment 05

Due: Monday, Nov 9th, 2020 at 10:30 am.

Write a main program named **t04.py** that tests the function by asking the user to enter a number and displays the output. The negative numbers are invalid inputs, if the user enters a negative number the program should keep asking for positive number until the user enters one.

Copy the results to `testing.txt`.

5. Write a function called `winner` that takes no parameters and asks the user to enter a series of strings that represent the output of a game with a loop. The user should enter an empty string "" to signal the end of the series.

   After all strings have been entered, the function **returns** two numbers representing how many times the string "red" appeared in the input and how many times the string "green" appeared in the user input. Save the function in a PyDev library module named `functions.py`

   Write a program named **t05.py** that tests the function. If the string "red" appeared more than the "green" string then your program should display "`red wins`", otherwise it will display "`green wins`". If you have the same count for "green" and "red" strings the program should display display "`tie`".

   The output will be:

```
Enter the winning team: red
Enter the winning team: green
Enter the winning team: red
Enter the winning team: red
Enter the winning team: blue
Enter the winning team: green
Enter the winning team:

Number of red entered: 3
Number of green entered: 2
red team wins!!!
```