CSE101 – Spring 2020
Programming Assignment #2
(12 points, Submission due date: 2 April 2020)

Instructions

For each of the following problems, create an error-free Python program.

- Each program should be submitted in a separate Python file that follows a particular naming convention: Submit the answer for problem 1 as "Assign2Answer1.py" and for problem 2 as "Assign2Answer2.py" and so on.
- These programs should execute properly in PyCharm using the setup we created in lab.
- At the top of every file add your name and Stony Brook email address in a comment.
- Write 2 test cases to test your code for all problems that do not take user input.

Problems

Problem 1: (2 points)

An equation for creating some chaotic output is given by:

X = 3. 9 * X * (1 - X).

Write a Python program that allows a user to input two initial values and the number of iterations, and then prints a nicely formatted table showing how each value change over time following the equation.

For example, if the starting values were 0.25 and 0.26 and they chose 10 iterations, the table might look like this:

```
index 0.25       0.26
-------------------------
1     0.731250   0.750360
2     0.766441   0.730547
3     0.698135   0.767707
4     0.821896   0.695499
5     0.570894   0.825942
6     0.955399   0.560671
7     0.166187   0.960644
8     0.540418   0.147447
9     0.968629   0.490255
10    0.118509   0.974630
```

Problem 2: (2 points)

Write a Python program that defines and uses a function named `printAsteriskPattern(numRows)` to construct the pattern described below using a `for` loop. The function receives as input the number of rows in the pattern (can be even or odd).

The number of asterisks in the first row and last row are: `math.ceil(numRows/2)`.

In the case where the number of rows is odd, then there should be one single asterisk in the middle row, while in the case where the number of rows is even, there should be two middle rows with one asterisk each as shown in the following:

```
>>>printAsteriskPattern(9)

* * * * *
* * * *
* * *
* *
*
* *
* * *
* * * *
* * * * *

>>>printAsteriskPattern(10)

* * * * *
* * * *
* * *
* *
*
*
* *
* * *
* * * *
* * * * *
```

Problem 3: (3 points)

Write a program that has a function which takes a string as a parameter in the following date format: `month/day/year` and prints whether or not the date is valid. For example, 05/04/1962 is valid, but 09/31/2000 is not (since September has only 30 days).

You can assume the date is always in the format MM/DD/YYYY.

Be sure to consider leap year. Leap year occurs in every year that is a multiple of 4, except for years that are multiples of 100 and **not also** multiples of 400. For example, 2000, 2020, and 2104 are leap years, but 2003, 2021, 1900 are not leap years.

Test your function using input provided by a user. For example:

```
>>>Enter date: 01/22/1996
```

```
'01/22/1996' is a valid date.
>>>Enter date: 01/32/1996
'01/32/1996' is an invalid date.
>>>Enter date: 02/29/2100
'02/29/2100' is an invalid date.
```

Problem 4: (2 points)

Define a new function named `collatz` that takes one positive integer as a parameter and returns a list of numbers, starting with the initial number. Then use a while loop to add to the list using the following process:

- Let `n` be the value currently at the end of the list.
- If `n` is even, add to the list the number: `(n // 2)`
- If `n` is odd, add to the list the number: `(3*n + 1)`
- If the number 1 is added to the list, the function should return the entire list.

This problem is known as the Collatz conjecture and for any value of n, the list will eventually get to a 1 (though this hasn't been formally proven).

Here are some test cases with correct output:

```
>>> collatz(5)
[5, 16, 8, 4, 2, 1]
>>> collatz(6)
[6, 3, 10, 5, 16, 8, 4, 2, 1]
```

Problem 5: (3 points)

A Mersenne prime is a prime number that is one less than a power of 2. For example, 131071 is a Mersenne prime because 131071 + 1 = 2^{17}. Write a function named `mersennes` that returns a list of all Mersenne primes less than a specified value. For example, the following call makes a list of Mersenne primes less than 1000:

```
>>> mersennes(1000)
[3, 7, 31, 127]
```

And this call makes a list of the Mersenne primes less than 1,000,000

```
>>> mersennes(1000000)
[3, 7, 31, 127, 8191, 131071, 524287]
```

Hint: it may help to write a helper function to calculate if a number is a prime number, and use that function to only check the numbers that are one less than a power of 2.