ASSIGNMENT 1

Read the instructions below carefully. The instructions must be followed. This assignment is worth 4% of your grade. The assignment is due on Wednesday 6th of Feb 8AM. No late assignments will be accepted. This is an individual assignment. Please review the Plagiarism and Academic Integrity policy presented in the first class, i.e. read in detail pages 12-18 of course outline (ITI1120-course- outline-syllabus- on BRS). You can find that file on Brightspace under Course Info. While at it, also review Course Policies on pages 14 and 15.

The goal of this assignment is to learn and practice (via programming) the concepts that we have learned so far: numbers, algebraic expressions, boolean expressions, strings, operations on strings, type conversion, variables, use of Python's builtin functions including input and output functions, designing your own functions, documenting your own functions via docstrings, and testing your functions. Before you can start this assignment, you need to know how to use a (IDLE's) text editor to edit python modules (i.e. files) as well as how to use python shell interactively to test your code. If you have no idea how to do these things watching video of the 3rd lecture, for example, will help. Submit your assignment by the deadline via Brightspace (as instructed and practiced in the first lab.) You can make multiple submissions, but only **the last submission before the deadline** will be graded. What needs to be submitted is explained next.

The assignment has 14 programming questions (in Section 1 below). Each question asks you to design one function. Put all these functions (for all the questions below) in ONE file only, called a1_xxxxxx.py (where xxxxxx is replaced with your student number). Within this file, a1_xxxxxx.py, separate your answers (i.e. code) to each question with a comment that looks like this:

```
########################################################################
 # Question X
 ########################################################################
```

To have an idea on what this file a1_xxxxxx.py should look like, I included with this assignment a solution to an nonexistent assignment. You can view this mockup solution by opening the included file called a1_mockup_assignment_solution.py

In addition to a1_xxxxxx.py you must submit a separate file called a1_xxxxxx.txt. What should be in that file is explained below. Thus for assignment 1 you have to **SUBMIT TWO FILES**:

- a1_xxxxxx.py and
- a1_xxxxxx.txt.

as instructed in lab 1. Submit your assignment by the deadline via Brightspace (as instructed in the first lab.)

Your program must run without syntax errors. In particular, when grading your assignment, TAs will first open your file a1_xxxxxx.py with IDLE and press Run Module. If pressing Run Module causes any syntax error, the grade for the whole assignment will be zero.

Furthermore, for each of the functions below, I have provided one or two tests to test your functions with. For example, you should test question 2 by making function call `is_prime(5)` in the Python shell. To obtain a partial mark your function may not necessarily give the correct answer on these tests. But if your function gives any kind of python error when run on the tests provided below, that question will be marked with zero points.

To determine your grade, your functions will be tested both with examples provided in Section 2 Note (I have included some samples within the questions): " consider Testing your code" to cover all the 14 functions. Thus you too should test your functions with more example than what I provided in Section 2.

Each function has to be documented with docstrings (as will be explained in the Lectures. In particular, each function has to have docstrings that specify:
- type contract
- description about what the function does (while mentioning parameter names)
- preconditions, if any

Section 1: Assignment 1 questions

1. Write a function **`repeat(string, n, delim)`** that returns the string string repeated n times, separated by the string delim. For example,
```
>>>repeat("ho", 3, ", ")
"ho, ho, ho".
```
2. Write a function **`is_prime(n)`** that returns a boolean value indicating whether an integer is prime or not.
3. Implement function **`points()`** that takes as input four numbers *x*1, *y*1, *x*2, *y*2 that are the coordinates of two points (*x*1;*y*1) and (*x*2*; y*2) in the plane. Your function should compute:
    a. The slope of the line going through the points, unless the line is vertical
    b. The distance between the two points.

   Your function should *print* the computed slope and distance in the following format. If the line is vertical, the value of the slope should be string 'infinity'. *Note:* Make sure you convert the slope and distance values to a string before printing them.
```
>>> points(0, 0, 1, 1)
The slope is 1.0 and the distance is 1.41421356237
>>> points(0, 0, 0, 1)
The slope is infinity and the distance is 1.0
```

4. Write a function named **`month_apart()`** that accepts four integer parameters representing two calendar dates. Each date consists of a month (1 through 12) and a day (1 through the number of days in that month [28-31]). Assume that all dates occur during the same year. The method returns whether the dates are at least a month apart. For example, the following dates are all considered to be at least a month apart from 9/19 (September 19): 2/14, 7/25, 8/2, 8/19, 10/19, 10/20, and 11/5. The following dates are NOT at least a month apart from 9/19: 9/20, 9/28, 10/1, 10/15, and 10/18. Note that the first date could come before or after (or be the same as) the second date. Assume that all parameter values passed are valid.

   Sample calls:

```
month_apart( 6, 14,  9, 21) should return True,       because June 14 is at least a month
before September 21
month_apart( 4,  5,  5, 15) should return True,       because April 5 is at least a month
before May 15
month_apart( 4, 15,  5, 15) should return True,       because April 15 is at least a month
before May 15
month_apart( 4, 16,  5, 15) should return False,      because April 16 isn't at least a month
apart from May 15
month_apart( 6, 14,  6,  8) should return False,      because June 14 isn't at least a month
apart from June 8
month_apart( 7,  7,  6,  8) should return False,      because July 7 isn't at least a month
apart from June 8
month_apart( 7,  8,  6,  8) should return True,       because July 8 is at least a month
after June 8
month_apart(10, 14,  7, 15) should return True,       because October 14 is at least a month
after July 15
```

5. Implement function **`reverse_int()`** that takes a three-digit integer as input and returns the integer obtained by reversing its digits. For example, if the input is *123*, your function should return *321*. You are not allowed to use the string data type operations to do this task. Your program should simply read the input as an integer and process it as an integer using operators such as // and %. You may assume that the input integer does not end with the *0* digit.

```
>>> reverse_int(123)
321
>>> reverse_int(908)
809
```

6. Write function **`vowelCount()`** that takes a string as input and counts and prints the number of occurrences of vowels in the string.

```
>>> vowelCount('Le Tour de France')

a, e, i, o, and u appear, respectively, 1, 3, 0, 1, 1
times.
```

7. Write the following functions.
   a. **`allTheSame(x, y, z)`** (returning true if the arguments are all the same)
   b. **`allDifferent(x, y, z)`** (returning true if the arguments are all different)
   c. **`sorted(x, y, z)`** (returning true if the arguments are sorted, with the smallest one coming first)
8. A year with 366 days is called a leap year. Leap years are necessary to keep the calendar synchronized with the sun because the earth revolves around the sun once every 365.25 days. Actually, that figure is not entirely precise, and for all dates after 1582 the *Gregorian correction* applies. Usually years that are divisible by 4 are leap years (for example, 1996). However, years that are divisible by 100 (for example, 1900) are not leap years, but years that are divisible by 400 are leap years (for example, 2000). Implement function **`leap()`** that takes one input argument—a year—and returns True if the year is a leap year and False otherwise.

```
>>> leap(2008)
True
>>> leap(1900)
False
>>> leap(2000)
True
```

9. Write function **`letter2number()`** that takes as input a letter grade (A, B, C, D, F, possibly with a - or +) and returns the corresponding number grade. The numeric values for A, B, C, D, and F are 4, 3, 2, 1, 0. A + increases the number grade value by 0.3 and a - decreases it by 0.3.

```
>>> letter2number('A-')
3.7
>>> letter2number('B+')
3.3
>>> letter2number('D')
1.0
```

10. Write a function **`is_palindrome()`** that accepts a string as argument and returns True or False indicating whether the string is a palindrome or not. A palindrome is a string that can be read the same way forward and backward. Your function does not need to convert between lower and upper case characters and needs to check only for an exact match including case. For example, the string 'madam' is a palindrome but the string 'Madam' is not. You are not allowed to generate a new string in your implementation of this function. Rather, you should walk through the string to determine whether it is a palindrome or not.

11. Write a function **`is_nneg_float()`** that checks if string s denotes a non-negative floating point value (not in scientific notation). This function should return True if s contains (at most) one decimal point and one or more digits (and nothing else); and False otherwise. According to this definition an integer should return True.

```
>>> is_nneg_float("2.15")
True
>>> is_nneg_float("3.")
True
>>> is_nneg_float(".5")
True
>>> is_nneg_float("123")
True
>>> is_nneg_float("-12")
False
```

```
>>> is_nneg_float("1e10")
False
```

12. Rock, Paper, Scissors is a two-player game in which each player chooses one of three items. If both players choose the same item, the game is tied. Otherwise, the rules that determine the winner are:
    a. Rock always beats Scissors (Rock crushes Scissors)
    b. Scissors always beats Paper (Scissors cut Paper)
    c. Paper always beats Rock (Paper covers Rock)

    Implement function **`rps()`** that takes the choice ('R', 'P', or 'S') of player 1 and the choice of player 2, and returns -1 if player 1 wins, 1 if player 2 wins, or 0 if there is a tie.

```
>>> rps('R', 'P')
1
>>> rps('R', 'S')
-1
>>> rps('S', 'S')
0
```

13. Write a function **`alogical(n)`** , that takes as input a number, n, where n is bigger or equal to 1, and returns the minimum number of times that n needs to be divided by 2 in order to get a number equal or smaller than 1. For example 5.4/2=2.7. Since 2.7 is bigger than 1, dividing 5.4 once by 2 is not enough, so we continue. 2.7/2=1.35 thus dividing 5.4 twice by 2 is not enough since 1.35 is bigger than 1. So we continue. 1.35/2=0.675. Since 0.675 is less than 1, the answer is 3. In particular, these calculations determine that 5.4 needs to be divided by 2 three times minimum in order to get a number that is less than or equal to 1.

14. Write a function named **`count_even_digits()`** that accepts two integers as parameters and returns the number of even-valued digits in the first number. An even-valued digit is either 0, 2, 4, 6, or 8. The second value represents how many digits the number has. The second value is guaranteed to match the number of digits in the first number.
    For example, the number 8546587 has four even digits (the two 8s, the 4, and the 6), so the call count_even_digits(8346387, 7) should return 4.
    You may assume that the values passed to your function are non-negative.

Section 2: Testing your code

We would like to see evidence that you have tested your functions in python shell, like we did in class. Do this by opening your assignment solution, i.e. opening a1_xxxxxx.py, with IDLE and then test each of

your functions individually. Then copy and paste the python shell output into a text file called a1_xxxxxx.txt The contents of a1_xxxxxx.txt must have something like this in it:

```
>>> # testing Question 1
>>>repeat("ho", 3, ", ")

"ho, ho, ho"

>>> # testing Question 2
>>>is_prime(3)

True

>>>is_prime(4)

False

.

.

.

>>> # testing Question 14
>>> count_even_digits(8346387, 7)

4
```