CISC106 Summer 2011 Lab03

- This lab an all subsequent labs will be due Sunday at 11:55 PM EDT on Sakai.
- You should create a file called *lab03.py* and implement all the functions described here in it. Remember to put a header in this file containing at least your name, the course name and number and a brief description of what the file contains.
- A lab03_tests.py skeleton is provided for you on the course website.

Preparation (do not submit for grading)

1. The *modulus* operator (% in Python) takes two integer operands and returns the remainder of the result of dividing the first by the second. For example, $1\%4$ would return 1, and $6\%4$ would return 2. Write a function `is_divisible` which takes two parameters and returns True if the first divides evenly by the second and returns False if not.

Programs (to be graded)

1. Mathematical equations in one variable are claims about an unknown number. For example, the equation $x^2 + 2 \cdot x + 1 = 0$ is a claim concerning some unknown number x. For $x = -1$ the claim holds: $x^2 + 2 \cdot x + 1 = -1^2 + 2 \cdot -1 + 1 = 1 - 2 + 1 = 0$. For $x = 1$ it doesn't, because $x^2 + 2 \cdot x + 1 = 1^2 + 2 \cdot 1 + 1 = 1 + 2 + 1 = 4$ and 4 is *not* equal to 0. A number for which the claim holds is called a *solution* to the equation. A function that represents the above claim looks like:[1]

```
def equation0(x):
    """
    returns True if x is a solution for x**2 + 2*x + 1 = 0,
    False otherwise.
    """
    return x ** 2 + 2 * x + 1 == 0
```

   Translate the following equations into Python functions:

   (a) $4 \cdot n + 2 = 62$
   (b) $2 \cdot n^2 = 102$
   (c) $4 \cdot n^2 + 6 \cdot n + 2 = 462$

   Write tests to determine whether or not 10, 12, or 14 are solutions to these equations. (Each function should have one test defined with three asserts. All nine asserts should hold so that all three tests pass.)

2. In software engineering, a *loop* is a construct which causes the body of code contained within it to be executed repeatedly based on some conditions. A *for loop* in Python executes its body once for each value over some range. For example:

[1] NB: There is a docstring between the def statement and the first line of code for this function. This is a common form of documentation in source code. From now on, you should put a docstring in all of your functions which explain what each parameter is and what the return value of the function will be. Use the docstring in the example as a guideline.

```
for i in range(10):
    print i
```

Will produce the following output:

```
0
1
2
3
4
5
6
7
8
9
```

The *range* function takes as an argument a number n and then produces the first n natural numbers (starting at 0.) On each pass - or *iteration* - of the loop, the variable i is assigned the next value in the range. Using a for loop and the range function, implement the following functions:

(a) A function `sum_to` which takes a number n and returns the sum of 1 thru n.

(b) A function `sum_odd_to` which takes a number n and returns the sum of all odd numbers between 1 and n (inclusive.)

(c) A function `sum_even_to` which takes a number n and returns the sum of all even numbers between 1 and n (inclusive.)

For each function, you should write a test which tests three different cases (*i.e.* three different kinds of input.)

3. Write a function `risk_metric` which will calculate how risky a prospective borrower would be for a bank. The function will take into consideration the following values:

   - The borrower's *credit ratio*, which is his/her credit score minus 550 and then divided by 250
   - An *adjusted balance* which is the amount of the loan minus the borrower's liquid (cash) assets.
   - An *expected income* which is the borrower's salary times ten, plus the value of the borrower's non-liquid assets.

   The risk metric itself involves subtracting the adjusted balance from the expected income multiplied by the credit ratio. The result of this is then divided by the adjusted balance. You should write a test for this function that tests the following three cases:[2]

[2] Comparing floating point numbers isn't exact like it is with integers, due to the way floating point numbers are stored in the computer. For this reason use `assertAlmostEqual` rather than `assertEqual` in your tests for this function.

(a) A borrower taking out a loan of $\$100,000$ who makes $\$40,000$ a year with $\$20,000$ in cash saved up, no non-liquid assets and a credit score of 700 will have a risk factor of 2.0

(b) A borrower taking out a loan of $\$250,000$ who makes $\$80,000$ a year with $\$25,000$ in cash saved up, $\$110,000$ worth of non-liquid assets and a credit score of 725 will have a risk factor of 1.8311111

(c) A borrower taking out a loan of $\$400,000$ who makes $\$120,000$ a year with $\$100,000$ in cash saved up, no non-liquid assets and a credit score of 710 will have a risk factor of 1.56

You should submit lab03.py and lab03_tests.py along with any other docs required by your TA on Sakai.