Introduction to Scientific Computing, PSCB57, Fall 2016
Assignment 2
Floating Point Numbers

- The deadline for this assignment is Friday, September 23rd, 5pm.
- This assignment comes in multiple parts. Submit all your answers in one file. Your submission should include the following functions: `f(x)`, `g(x)`, `fib1(x)`, `fib2(x)`, `fibd(x)`.
- Only submit the functions you wrote, i.e. no other code such as print statements that you might have used to test your submission.
- Do not use any packages or libraries.
- The entire submission has to be 70 lines of code or less (including comments).
- You must submit the assignment electronically at `http://rein.utsc.utoronto.ca/submit/`. On the same website, you can run an automated test of your program before submitting it. The username is `pscb57`, the password is `2016`.
- Your code needs to pass all tests in 1 second or less.
- You must be present at the tutorial in the following week to take a quiz. If you do not show up or fail to pass the quiz, your assignment will be marked as 0% even if it was correct.
- Plagarism is taken very seriously. However, you are not expected to work in solitude and are encouraged to talk to your classmates. But keep in mind that if you submit an assignment, you have to fully understand it in order to pass the quiz.

Part 1

This part is about the precision of floating point numbers. Have a look at the following function.

```
def f(x):
    return x/x
```

The return value is obviously 1, no matter what the function argument is. Your task is to modify the function to get the following behaviour:

```
>>> f(1e-16)
0.0
>>> f(2.5e-16)
1.0
>>> f(1e-13)
0.9988901220865705
>>> f(1.12e-16)
2.0
>>> f(-1e15)
1.0
>>> f(1e-15)
1.1111111111111112
```

You are not allowed to change the mathematical nature of the function. In other words, you should be able to simplify the expression mathematically to always get exactly 1. To get the desired output in your python function, you need to make use of the finite precision of floating point numbers. To make things a little more interesting, you are only allowed to add the following characters (multiple times if you want) to the body of the function `f`:

```
(     )     +     -     1
```

Part 2

This part is about the finite range of floating point numbers. Write a function `g(x)` which returns `inf` if x is a positive floating point number, `-inf` if x is a negative floating point number and 0 if x is 0. Calculate the return value by multiplying x with the number 10 multiple times. Create a for loop to do this multiplication and hard code the number of iterations. Thus, your function should include a loop of the following form:

```
for i in range(...):
```

Think about what maximum number of iterations you need such that your function works with any IEE754 floating point number (we use double precision).
Finally, when you pass your function an integer rather than a floating point number, it should return a finite number. Make sure you understand why this happens (there might be a question about this on the quiz). Hint: look up what dynamically typed means and think about what data types python uses when it executes your function.

Part 3

This part is about floating point numbers versus integers. You'll implement two different ways to calculate Fibonacci numbers to explore this.
The first function, `fib1(n)` should use the analytic formula we derived in the lecture:

$$F_n = \frac{1}{\sqrt{5}}\left(\left(\frac{1}{2}(1+\sqrt{5})\right)^n - \left(\frac{1}{2}(1-\sqrt{5})\right)^n\right). \quad (1)$$

The second function, `fib2(n)` should use the recursive formula with the logarithmic scaling, $O(\log(n))$, that we also derived in the lecture. Don't use any packages to calculate the square root in the above equation, instead use the following precalculated value for $\sqrt{5}$:

```
sqrt5 = 2.2360679774997898
```

Implement the following function, `fibd(n)` which simply returns the difference of the two implementations.

```
def fibd(n):
    return fib1(n)-fib2(n)
```

Play around with different values for n and see when the difference `fibd(n)` reaches order unity.