Homework 2 for Math 480A

http://wiki.wstein.org/2008/480a

Due Wednesday April 16, 2008

Each problem has equal weight, and parts of problems are worth the same amount as each other.

1. Below I've defined **ten** Python objects $a, b, c, d, e, f, g, h, i, k$. Which are *immutable*, meaning the Python object itself can't be changed from the Python interpreter (without using Cython)?

```
a = 'this is a string'
b = int(123456789)
c = 2/3
d = [1,2,3,4,5]
e = (1,2,3,4,5)
f = (1,2,3,4,[5])
g = {1:'a', 2:'theorem'}
h = set([1,2,3])
class i: pass
k = i()
```

2. Write[1] a function called `rotate_word` that takes a string and an integer as parameters, and that returns a new string that contains the letters from the original string "rotated" by the given amount. To rotate a letter means to shift it through the alphabet, wrapping around to the beginning if necessary. For example, `Y` shifted by 1 is `Z`, and `Z` shifted by 1 is `A`. `A` shifted by 5 is `F`. For example, "cheer" rotated by 7 is "jolly" and "melon" rotated by -10 is "cubed". You might want to use the built-in functions `ord`, which converts a character to a numeric code, and `chr`, which converts numeric codes to characters.

3. (a) Input and play around with the following *recursive* implementation of the factorial function in Sage:

```
def factorial(n):
    if n <= 2: return n
    return n*factorial(n-1)
```

   In particular, show that it works for the first few values of n, and give timings.

   (b) What is the largest n for which the above function actually works? What goes wrong? (Let this be a warning to you...)

4. Write a function called `frequent_letters` that takes a string and a positive integer n and returns the n most common letters in the string. For example

```
sage: frequent_letters('this is a string', 3)
['s', 'i', ' ']
sage: frequent_letters('william stein', 2)
['i', 'l']
sage: frequent_letters('a viable open source alternative to maple, mathematica, and matlab', 5)
['a', ' ', 't', 'e', 'm']
```

[1]From http://www.greenteapress.com/thinkpython/

5. Write a fast Cython function to determine whether or not a positive integer (that is at most 2^{31}) is a perfect power of 3. How does its speed compare to the speed of a Python function that does the same thing? [Note: In order to use Cython you have to have a C compiler installed on your computer. In OS X this means installing XCode; with Windows/VMware it is automatic; with Linux it is highly likely you have a compiler.]

6. (a) Create a new Python class `hlist` that derives from the builtin `list` type but adds a new method `__hash__`:

   ```
   def __hash__(self):
       ???
   ```

   (b) Can you create dictionaries with `hlist` instances as keys? Illustrate your answer.

   (c) If the answer to (b) was "yes", illustrate one thing that goes horribly wrong as a result. That is, show why it is *a bad idea* to define a hash method for mutable objects.

7. List *three* ideas for projects you personally could conceivably do as your final project for this course. Be creative.