Here are some questions that the TAs have written to help you review for the final. This review sheet covers only some of the topics after the second midterm – be sure to review other course material as well!

**Exceptions**

**What would Python print?**

```python
def a():
    print("Steven says")
    int("what happens here")
    print("hello")

def b():
    try:
        print(1)
        a()
        print(2)
        return 3
    except ValueError:
        print(4)
        return 5

def c():
    b()
    a()
    return 6
```

```python
>>> a()

>>> b()

>>> c()
```
Dynamic Scope

```python
x = 10
def car(x):
    return van()

def van():
    if x == 10:
        return "honk"
    return "tumble"
```

What would `car(11)` return in lexical scope?

What would `car(11)` return in dynamic scope? Draw an environment diagram.

Concurrency

```python
x = 10
Assume the following 2 lines are run in parallel.
>>> x = x + x
>>> x = x * x
```

Correct values of `x`:

Other possible values of `x`:  

---

CS61A Final Review
**Parsing**

Write the function `tokenize_html`. Given a string that includes HTML tags and text between the HTML tags, return a list which consists of HTML tags and the text between the HTML tags. See the doc-string for examples.

```python
def tokenize_html(str):
    """Given a well-formed HTML string, return a list where each element is a token.

    HTML tags are considered elements, and text between tags are elements.

    An HTML tag starts with a < and ends with a >. Any time a < appears, you
    can assume there is a corresponding >, and another < will not appear before the
    corresponding >.

    For example, you will NOT have to worry about a string like "<html <asdf>>".

    You may assume that the string starts and ends with an HTML tag.
    """
    >>> tokenize_html("<html>hi</html>")
    ['<html>', 'hi', '</html>']
    >>> tokenize_html("<body>hello there    </body>")
    ['<body>', 'hello there    ', '</body>']
Interpreters
Given a list returned by a call to `tokenize_html`, turn that list into a Tree structure. For example,

```python
```

should turn into the tree structure:

```
     ‘<a>’
    /     \
‘hello there’ ‘<b>’      ‘rumble’
   |        
‘lower’  
```

The tag `<a>` corresponds to the closing tag `</a>`, so the elements between those two tags become children of the Tree whose datum is `‘<a>’`. Relevant Tree code has been included below.

class Tree(object):
    def __init__(self, datum, children=[]):
        self.datum = datum
        self.children = children

    def print_tree(t, prefix=""):
        print prefix+t.datum
        for child in t.children:
            new_prefix = "  |___";
            for _ in prefix:
                new_prefix = " " + new_prefix
            print_tree(child, new_prefix)

    def is_tag(token):
        return token.startswith("<")

    def is_end_tag_of(tag, token):
        return token == tag[0] + "/" + tag[1:]

#continued on next page
def list_to_tree(lst):
Iterators

The basic idea behind the Sieve of Eratosthenes that you saw in class was that a number $n$ is prime if it is not divisible by any primes less than $n$. Write an iterator that produces prime numbers in order forever.

class Primes(object):

Streams

Write interleave, a function that takes two streams and returns a stream where the elements of each stream are interleaved.

def interleave(s1, s2):

Now create a stream of all positive and negative integers. Hint: The first element in the stream should be 0.

def all_integers_stream():