Review materials

See website for a version of last year’s final with conventions redone to match this year.
Mistakes Were Made

I accidentally posted last semester’s A7 solutions to the Web. (I, Prof. Lee, and not an angel.)

• We are moving the solution to CMS so everyone in the class can see it, but not the world.

• We will grade A7 as if this hadn’t happened.

• You should try to solve each step yourself before you look (but we are permitting you to look). Don’t just copy; yours should be diff. in design.

• Credit Prof. White at top of file if you got/incorporated ideas from his solution.
Implement according to spec

class Document(object):
    """Instances represent scholarly documents."

    Instance variables:
    title [str]: title of the document
    authors [list of str]: names of authors, in the form 'Last, F.'
    works_cited [list of Document]: documents cited by this document.
    """

def __init__(self, title, authors, works_cited):
    """A new document with given title, author list, and bibliography."""
    pass  # IMPLEMENT ME

def biblio_entry(self):
    """A minimal bibliography entry for this document, authors followed by title.
Example: Marschner, S., Lee, L., White, W. "Intro to Python.""""
    pass  # IMPLEMENT ME
class Book(Document):
    """Instances represent books.

    Instance variables:
    publisher [str]: name of publisher
    pub_year [int]: year of publication
    """

def __init__(self, title, authors, works_cited,
             publisher, pub_year):
    """A new book with the given properties."""
    pass  # IMPLEMENT ME

class Article(Document):
    """Instances represent articles that appear in journals.

    Instance variables:
    journal [str]: title of the journal
    month [int]: month (January = 1, etc.) of issue in which article appears
    year [int]: year in which article appeared
    """

def __init__(self, title, authors, works_cited,
             journal, volume, issue, year):
    """A new journal article with the given properties."""
Implement method biblio_entry according to its spec. You may add anything you need to the class above the initializer.

class Article(Document):
    """Instances represent articles that appear in journals.

    Instance variables:
    journal [str]: title of the journal
    month [int]: month (1 = Jan, etc.) of issue in which article appears
    year [int]: year in which article appeared
    """

    def __init__(self, title, authors, works_cited, journal, volume, issue, year):
        """A new journal article with the given properties."""
        Document.__init__(self, title, authors, works_cited)
        self.journal = journal
        self.month = month
        self.year = year

    def biblio_entry(self):
        """A bibliography entry with journal information. Example of format:
        """
        pass  # IMPLEMENT ME
Implement methods `citing_documents` and `works_by` according to spec.

class Library(object):
    """Instances represent library collections."""

    def __init__(self, documents):
        """A library containing the given documents."""
        self.documents = documents[:]

    def citing_documents(self, document):
        """A list of the documents that cite the given document."""

    def works_by(self, author):
        """A list of the documents authored by a particular person."""
Can you refer to the value “Harper” when the indicated statements are executing? How?
Can you refer to the value “Harper” when the indicated statements are executing? How?
Can you refer to the value “Harper” when the indicated statements are executing? How?
Can you refer to the value "Harper" when the indicated statements are executing? How?
Can you refer to the value “Harper” when the indicated statements are executing? How?
Can you refer to the value “Harper” when the indicated statements are executing? How?
Can you refer to the value “Oct” when the indicated statements are executing? How?
Diagram the execution of the call do_test().
What Might You Be Asked

- Create your own Exception class
- Write code to raise an exception
- Follow the path of a raised exception
- Write a simple try-except code fragment
When Do Exceptions Happen?

<table>
<thead>
<tr>
<th>Automatically Created</th>
<th>Manually Created</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>def void foo():</code></td>
<td><code>def void foo():</code></td>
</tr>
<tr>
<td><code>x = 5 / 0</code></td>
<td><code>raise Exception('Test Exc.')</code></td>
</tr>
</tbody>
</table>

Python creates Exception for you automatically

You create Exception manually by raising it
What value does foo() return?

def foo():
    x = 1
    try:
        x = 2
        (A)
        x = x + 5
        (B)
    except StandardError:
        x = x + 10
    raise StandardError()
    (C): nowhere (no raise stmt)
    return x
What value does foo() return?

def foo():
    x = 1
    try:
        x = 2
        x = x+5
    except StandardError:
        x = x+10
    raise StandardError()
    return x

(A): returns 11;
(B): returns 12 – sometimes, only parts of try-blocks are executed
(C): nowhere (no raise stmt)

(C) returns 7 – except-block isn't executed if no exception
def first(x):
    print 'Starting first.'
    try:
        second(x)
    except:
        print 'Caught at first'
    print 'Ending first'

def second(x):
    print 'Starting second.'
    try:
        third(x)
    except:
        print 'Caught at second'
    print 'Ending second'

def third(x):
    print 'Starting third.'
    assert x < 1
    print 'Ending third.'

What is the output of first(2)?
```python
def first(x):
    print 'Starting first.'
    try:
        second(x)
    except:
        print 'Caught at first'
    print 'Ending first'

def second(x):
    print 'Starting second.'
    try:
        third(x)
    except:
        print 'Caught at second'
    print 'Ending second'

def third(x):
    print 'Starting third.'
    assert x < 1
    print 'Ending third.'
```

What is the output of `first(2)`?

```
'Starting first.'
'Starting second.'
'Starting third.'
'Caught at second'
'Ending second'
'Ending first'
```
def first(x):
    print 'Starting first.'
    try:
        second(x)
    except IOError:
        print 'Caught at first'
    print 'Ending first'

def second(x):
    print 'Starting second.'
    try:
        third(x)
    except AssertionError:
        print 'Caught at second'
    print 'Ending second'

def third(x):
    print 'Starting third.'
    if x < 0:
        raise IOError()
    elif x > 0:
        raise AssertionError()
    print 'Ending third.'

What is the output of `first(-1)`?
def first(x):
    print 'Starting first.'
    try:
        second(x)
    except IOError:
        print 'Caught at first'
    print 'Ending first'

def second(x):
    print 'Starting second.'
    try:
        third(x)
    except AssertionError:
        print 'Caught at second'
    print 'Ending second'

def third(x):
    print 'Starting third.'
    if x < 0:
        raise IOError()
    elif x > 0:
        raise AssertionError()
    print 'Ending third.'

What is the output of first(-1)?

Starting first.
Starting second.
Starting third.
Caught at first.
Ending first.
def isfloat(s):
    """Returns: True if string s represents a float.
    False otherwise""
    # Implement me

float(s) returns an error if s does not represent a float
def isfloat(s):
    """Returns: True if string s represents a float.
    False otherwise"""
    try:
        x = float(s)
        return True
    except:
        return False

Conversion to a float might fail
If attempt succeeds, string s is a float
Otherwise, it is not
def zrun(b,n):
    """Returns: [i,j] where b[i..j-1] is the first occurrence of
    n 0's in a row.  (OK if b[j] is 0).
    If there is no such run, i == j.
    Pre: n >= 0.  b a list of ints (possibly empty).
    Examples: zrun([5,0,0,0,0,0,2,0,0,0,0,0,0], 2) returns [1,3]
              zrun([5,0,0,2,0,0,0], 3) returns [4,7]
              zrun([1,0,0,2,0,0,0],5) could return [0,0]
    """

    Suggested invariant: b[i..j-1] is a "candidate run":
    b[i..j-1] are zeroes; b[i-1] not zero (if it exists)
Loop solution

```
j = 0
i = j
# invariant: b[i..j-1] a candidate run. More precisely,
# b[i..j-1] are 0; b[i-1] not zero (might not exist)
while j < len(b):
    if j-i == n:
        return [i,j]
    elif b[j] != 0:  # no longer have a candidate run
        j += 1
        i = j
    else:
        j += 1

# if here, j is len(b)
if j - i == n:
    return [i,j]
else:
    return [0,0]
```