Exam Info

• Prelim 2: 7:30–9:00PM, Thursday, Nov. 10th
  ▪ Last name A – K in Uris G01
  ▪ Last name L – O in Phillips 101
  ▪ Last name P – W in Ives 305
  ▪ Last name X – Z in Ives 105

• To help you study:
  ▪ Study guides, review slides are online
  ▪ Review solution to prelim 1 (esp. call stack!)
What is on the Exam?

• Five questions from the following topics:
  ▪ Recursion (Lab 8, A4)
  ▪ Iteration and Lists (Lab 7, A4, A6)
  ▪ Defining classes (Lab 9, Lab 10, A6)
  ▪ Drawing folders (Lecture, A5)
  ▪ Exceptions (Lectures 11 and 21)
  ▪ Short Answer (Terminology, Potpourri)

• +2 points for name, netid AND SECTION
What is on the Exam?

• Recursion (Lab 8, A4)
  ▪ Will be given a function specification
  ▪ Implement it using recursion
  ▪ May have an associated call stack question

• Iteration and Lists (Lab 7, A4, A6)

• Defining classes (Lab 9, Lab 10, A6)

• Drawing folders (Lecture, A5)

• Exceptions (Lectures 11 and 21)

• Short Answer (Terminology, Potpourri)
def histogram(s):

    """Return: a histogram (dictionary) of the # of letters in string s.

    The letters in s are keys, and the count of each letter is the value. If
    the letter is not in s, then there is NO KEY for it in the histogram.

    Example: histogram('') returns {},
    histogram('abracadabra') returns {'a':5,'b':2,'c':1,'d':1,'r':2}

    Precondition: s is a string (possibly empty) of just letters."""
def histogram(s):

"""Return: a histogram (dictionary) of the # of letters in string s.

The letters in s are keys, and the count of each letter is the value. If the letter is not in s, then there is NO KEY for it in the histogram.

Precondition: s is a string (possibly empty) of just letters."""

**Hint:**

- Use divide-and-conquer to break up the string
- Get two dictionaries back when you do
- Pick one and insert the results of the other
Call Stack Question

```python
def skip(s):
    """Returns: copy of s
    Odd (from end) skipped"
    result = ''
    if (len(s) % 2 == 1):
        result = skip(s[1:])
    elif len(s) > 0:
        result = s[0] + skip(s[1:])
    return result
```

- **Call**: `skip('abc')`
- Recursive call results in four frames (why?)
  - Consider when 4th frame completes line 6
  - Draw the entire call stack at that time
- Do not draw more than four frames!
What is on the Exam?

- Recursion (Lab 8, A4)
- Iteration (Lab 7, A4, A6)
  - Again, given a function specification
  - Implement it using a for-loop
  - May involve 2-dimensional lists
- Defining classes (Lab 9, Lab 10, A6)
- Drawing folders (Lecture, A5)
- Exceptions (Lectures 11 and 21)
- Short Answer (Terminology, Potpourri)
def evaluate(p, x):

"""Returns: The evaluated polynomial p(x)

We represent polynomials as a list of floats. In other words

\[1.5, -2.2, 3.1, 0, -1.0\] is \(1.5 - 2.2x + 3.1x^2 + 0x^3 - x^4\)

We evaluate by substituting in for the value x. For example

\[\text{evaluate([1.5, -2.2, 3.1, 0, -1.0], 2)}\] is \(1.5 - 2.2(2) + 3.1(4) - 1(16) = -6.5\)
\[\text{evaluate([2], 4)}\] is 2

Precondition: p is a list (len > 0) of floats, x is a float"""
def max_cols(table):
    """Returns: Row with max value of each column
    We assume that table is a 2D list of floats (so it is a list of rows and each row has the same number of columns. This function returns a new list that stores the maximum value of each column.
    Examples:
    max_cols([ [1,2,3], [2,0,4], [0,5,2] ]) is [2,5,4]
    max_cols([ [1,2,3] ]) is [1,2,3]
    Precondition: table is a NONEMPTY 2D list of floats"""
What is on the Exam?

• Recursion (Lab 8, A4)
• Iteration (Lab 7, A4, A6)
• Defining Classes (Lab 9, Lab 10, A6)
  ▪ Given a specification for a class
  ▪ Also given a specification for a subclass
  ▪ Will “fill in blanks” for both
• Drawing folders (Lecture, A5)
• Exceptions (Lectures 11 and 21)
• Short Answer (Terminology, Potpourri)
class Customer(object):
    """Instance is a customer for our company
    Mutable attributes:
    _name: last name [string or None if unknown]
    _email: e-mail address [string or None if unknown]
    Immutable attributes:
    _born: birth year [int > 1900; -1 if unknown]"

# DEFINE GETTERS/SETTERS HERE
# Enforce all invariants and enforce immutable/mutable restrictions

# DEFINE INITIALIZER HERE
# Initializer: Make a Customer with last name n, birth year y, e-mail address e.
# E-mail is None by default
# Precondition: parameters n, b, e satisfy the appropriate invariants

# OVERLOAD STR() OPERATOR HERE
# Return: String representation of customer
# If e-mail is a string, format is 'name (email)'
# If e-mail is not a string, just returns name
class PrefCustomer(Customer):

    """An instance is a 'preferred' customer
    Mutable attributes (in addition to Customer):
        _level: level of preference [One of 'bronze', 'silver', 'gold'] """

    # DEFINE GETTERS/SETTERS HERE
    # Enforce all invariants and enforce immutable/mutable restrictions

    # DEFINE INITIALIZER HERE
    # Initializer: Make a new Customer with last name n, birth year y,
    # e-mail address e, and level l
    # E-mail is None by default
    # Level is 'bronze' by default
    # Precondition: parameters n, b, e, l satisfy the appropriate invariants

    # OVERLOAD STR() OPERATOR HERE
    # Return: String representation of customer
    # Format is customer string (from parent class) +', level'
    # Use __str__ from Customer in your definition
What is on the Exam?

- Recursion (Lab 7, A4)
- Iteration and Lists (Lab 6, A4, A5)
- Defining classes (Lab 8, Lab 9, A5)
- Drawing class folders (Lecture, A5)
  - Given a skeleton for a class
  - Also given several assignment statements
  - Draw all folders and variables created
- Exceptions (Lectures 11 and 21)
- Short Answer (Terminology, Potpourri)
Two Example Classes

class CongressMember(object):
    '''Instance is legislator in congress
    Instance attributes:
    _name: Member's name [str]'''

def getName(self):
    return self._name

def setName(self,value):
    assert type(value) == str
    self._name = value

__init__(self,n):
    self.setName(n)  # Use the setter

__str__(self):
    return 'Honorable '+self.name

class Senator(CongressMember):
    '''Instance is legislator in congress
    Instance attributes (plus inherited):
    _state: Senator's state [str]'''

def getState(self):
    return self._state

def setName(self,value):
    assert type(value) == str
    self._name = 'Senator '+value

__init__(self,n,s):
    assert type(s) == str and len(s) == 2
    CongressMember.__init__(self,n)
    self._state = s

__str__(self):
    return (CongressMember.__str__(self)+
            ' of '+self.state)
‘Execute’ the Following Code

```python
>>> b = CongressMember('Jack')
>>> c = Senator('John', 'NY')
>>> d = c
>>> d.setName('Clint')
```

**Remember:**
Commands outside of a function definition happen in global space

- Draw two columns:
  - Global space
  - Heap space

- Draw both the
  - Variables created
  - Object folders created
  - Class folders created

- If an attribute changes
  - Mark out the old value
  - Write in the new value
What is on the Exam?

- Recursion (Lab 8, A4)
- Iteration and Lists (Lab 7, A4, A6)
- Defining classes (Lab 9, Lab 10, A6)
- Drawing class folders (Lecture, A5)
- Exceptions (Lectures 11 and 21)
  - Try-except tracing (skipped on Prelim 1)
  - But now with dispatch on type
  - Will give you exception hierarchy
- Short Answer (Terminology, Potpourri)
Exceptions and Dispatch-On-Type

```python
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)`?

HINT:

- StandardError
- AssertionError
- IOError
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)?
What is on the Exam?

• Recursion (Lab 7, A4)
• Iteration and Lists (Lab 6, A4, A5)
• Defining classes (Lab 8, Lab 9, A5)
• Drawing class folders (Lecture, Study Guide)
• Exceptions (Lectures 11 and 21)
• Short Answer (Terminology, Potpourri)
  ▪ See the study guide
  ▪ Look at the lecture slides
  ▪ Read relevant book chapters

In that order

11/6/16
Prelim 2 Review