CS 1110

Prelim 2 Review
Fall 2012
Exam Info

• Prelim 1: 7:30–9:00PM, Tuesday, November 6th
  ▪ Last name **A – P** in Kennedy 1116
  ▪ Last name **R – T** in Warren 131
  ▪ Last name **U – Z** in Warren 231

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

• Arrive early! Helps reducing stress

• Grades released the same evening (if possible)
What is on the Exam?

- Five Questions (+2pts for name, netid):
  - Recursion (Lab 6, Lab 9, A4)
  - Iteration (Lab 7, A4)
  - Defining Classes (Lab 8, A5)
  - Drawing class folders (Study Guide)
  - Short Answer (Terminology)

- Roughly equal weight (#4 might be less)
What is on the Exam?

• Recursion (Lab 6, Lab 9, A4)
  - Will be given a function specification
  - Implement it using recursion
  - May have an associated call stack question

• Iteration (Lab 7, A4)
• Defining Classes (Lab 8, A5)
• Drawing class folders (Study Guide)
• Short Answer (Terminology)
```python
def merge(s1, s2):
    """Returns: characters of s1 and s2, in alphabetical order.
    Examples: merge('ab', '') = 'ab'
    merge('abbce', 'cdg') = 'abbccdeg'
    Precondition: s1 a string with characters in alphabetical order
    s2 a string with characters in alphabetical order"
```

Recursive Function

```python
def merge(s1, s2):
    """Returns: characters of s1 and s2, in alphabetical order.
    Examples: merge('ab', '') = 'ab'
    merge('abbce', 'cdg') = 'abbccdeg'
    Precondition: s1 a string with characters in alphabetical order
    s2 a string with characters in alphabetical order"
```

**Hint:**

- Make input “smaller” by pulling off first letter
- Only make one of two strings smaller each call
- Which one should you make smaller each call?

11/4/12 Prelim 2 Review
Recursive Function

```python
def merge(s1, s2):
    """Returns: characters of s1 and s2, in alphabetical order. """
    if s1 == 
        return s2
    if s2 == 
        return s1
    if s1[0] < s2[0]:  # Pick first from s1 and merge the rest
        return s1[0] + merge(s1[1:], s2)
    else:              # Pick first from s1 and merge the rest
        return s2[0] + merge(s1, s2[1:])
```

11/4/12  Prelim 2 Review
def skip(s):
    """Returns: copy of s
    Odd letters dropped"""
    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 reaches line 6
  ▪ Draw the entire call stack at that time
• Do not draw more than four frames!
def skip(s):
    """Returns: copy of s
    Odd letters dropped"
    result = ''
    if (len(s) % 2 == 1):
        result = skip(s[1:])
    elif len(s) > 0:
        result = s[0]+skip(s[1:])
    return result
```python
def skip(s):
    """Returns: copy of s
    Odd letters dropped"
    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')`

  1. `s = 'abc'`
  2. `result = ''`
  3. `s = 'bc'`
  4. `result = ''`

  - Line 6

  - Done
What is on the Exam?

• Recursion (Lab 6, Lab 9, A4)
• Iteration (Lab 7, A4)
  ▪ Again, given a function specification
  ▪ Implement it using a for-loop
  ▪ Challenge is how to use accumulator variables
• Defining Classes (Lab 8, A5)
• Drawing class folders (Study Guide)
• Short Answer (Terminology)
Implement Using Iteration

```python
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
    
    evaluate([1.5, -2.2, 3.1, 0, -1.0], 2) = 1.5 - 2.2(2) + 3.1(4) - 1(16) = -6.5
    evaluate([2, 4]) = 2
    
    Precondition: p is a list (len > 0) of floats, x is a float"""
```
def evaluate(p, x):
    """Returns: The evaluated polynomial p(x)

    Precondition: p is a list (len > 0) of floats, x is a float"
    ""
    sum = 0
    xval = 1
    for c in p:
        sum = sum + c * xval  # coefficient * (x**n)
        xval = xval * x
    return sum
What is on the Exam?

• Recursion (Lab 6, Lab 9, A4)
• Iteration (Lab 7, A4)
• Defining Classes (Lab 8, A5)
  ▪ Given a specification for a class
  ▪ Also given a specification for a subclass
  ▪ Will “fill in blanks” for both
• Drawing class folders (Study Guide)
• Short Answer (Terminology)
class Customer(object):

    """Instance is a customer for our company"""

    # Mutable attributes
    _name = None  # last name (string or None if unknown)
    _email = None  # e-mail address (string or None if unknown)

    # Immutable attributes
    _born = -1  # birth year (int > 1900; -1 if unknown)

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

    # DEFINE CONSTRUCTOR HERE
    # Constructor: Create a new 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
```python
class Customer(object):
    """Instance is a customer for our company"""
    # Mutable attributes
    _name = None  # last name (string or None if unknown)
    _email = None  # e-mail address (string or None if unknown)
    # Immutable attributes
    _born = -1     # birth year (int > 1900; -1 if unknown)

    # DEFINE PROPERTIES HERE
    @property
def name(self):
        return self._name

    @name.setter
def name(self, value):
        assert value is None or type(value) == str
        self._name = value

Actual Exam Question
will not be this long.
Just for this practice.
```
```python
class Customer(object):
    """Instance is a customer for our company"""
    # Mutable attributes
    _name = None  # last name (string or None if unknown)
    _email = None  # e-mail address (string or None if unknown)
    # Immutable attributes
    _born  = -1    # birth year (int > 1900; -1 if unknown)

    # DEFINE PROPERTIES HERE
    ...
    @property
def email(self):
        return self._email

    @email.setter
def email(self, value):
        assert value is None or type(value) == str
        self._email = value
```

Actual Exam Question will not be this long. Just for this practice.
class Customer(object):
    """Instance is a customer for our company"""

    # Mutable attributes
    _name = None  # last name (string or None if unknown)
    _email = None  # e-mail address (string or None if unknown)

    # Immutable attributes
    _born = -1  # birth year (int > 1900; -1 if unknown)

    # DEFINE PROPERTIES HERE

    ...  
    @property
    def born(self):
        return self._born

Actual Exam Question will not be this long. Just for this practice.
class Customer(object):
    """Instance is a customer for our company"""
    # Mutable attributes
    _name = None  # last name (string or None if unknown)
    _email = None  # e-mail address (string or None if unknown)
    # Immutable attributes
    _born  = -1  # birth year (int > 1900; -1 if unknown)

    # DEFINE PROPERTIES HERE

    ...  
    # DEFINE CONSTRUCTOR HERE
    def __init__(self, n, y, e=None):
        assert type(y) == int and (y > 1900 or y == -1)
        self.name = n  # Property, handles asserts
        self.email = e  # Property, handles asserts
        self._born = y  # Field, because no setter

Actual Exam Question will not be this long. Just for this practice.
class Customer(object):
    """Instance is a customer for our company"""
    #: Mutable attributes
    _name = None #: last name (string or None if unknown)
    _email = None #: e-mail address (string or None if unknown)
    #: Immutable attributes
    _born = -1 #: birth year (int > 1900; -1 if unknown)

    #: DEFINE PROPERTIES HERE
    ...
    #: DEFINE CONSTRUCTOR HERE
    ...
    #: OVERLOAD STR() OPERATOR HERE
    def __str__(self):
        if self.email is None:
            return "" if self.name is None else self.name
        else:
            s = "" if self.name is None else self.name
            return s+"('"+self.email+')""
class PrefCustomer(Customer):
    """An instance is a 'preferred' customer"""
    # Mutable attributes
    _level = 'bronze'  # level of preference. One of 'bronze', 'silver', 'gold'

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

    # DEFINE CONSTRUCTOR HERE
    # Constructor: Create 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 super in your definition
class PrefCustomer(Customer):
    """An instance is a 'preferred' customer"""
    # Mutable attributes
    _level = 'bronze'  # level of preference. One of 'bronze', 'silver', 'gold'
    
    # DEFINE PROPERTIES HERE
    @property
    def level(self):
        return self._level

    @level.setter
    def level(self, value):
        assert type(value) == str and (value == 'bronze' or value == 'silver' or value == 'gold')
        self._level = value

Actual Exam Question will not be this long. Just for this practice.
class PrefCustomer(Customer):
    """An instance is a 'preferred' customer""
    # Mutable attributes
    _level = 'bronze'  # level of preference. One of 'bronze', 'silver', 'gold'

    # DEFINE PROPERTIES HERE
    ...
    # DEFINE CONSTRUCTOR HERE
    def __init__(self, n, y, e=None, l='bronze'):
        super(PrefCustomer, self).__init__(n, y, e)
        self.level = l  # Property, handles asserts

    # OVERLOAD STR() OPERATOR HERE
    def __str__(self):
        return super(PrefCustomer, self).__str__() + ', ' + self.level

super calls method in parent class as helper

Actual Exam Question will not be this long. Just for this practice.
What is on the Exam?

• Recursion (Lab 6, Lab 9, A4)
• Iteration (Lab 7, A4)
• Defining Classes (Lab 8, A5)
• Drawing class folders (Study Guide)
  ▪ Given a skeleton for a class
  ▪ Also given several assignment statements
  ▪ Draw all folders and variables created
• Short Answer (Terminology)
Two Classes

class CongressMember(object):
    _name = " " # Member's name

    @property
def name(self):
        return self._name

    @name.setter
def name(self, value):
        assert type(value) == str
        self._name = value

def __init__(self, n):
    self.name = n # Use the setter

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

class Senator(CongressMember):
    _state = " " # Senator's state

    @property
def state(self):
        return self._state

    @property
def name(self):
        return self._name

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

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

    def __str__(self):
        return (super(Senator, self).__str__() +
                ' of ' + self.state)
 Execute the Following Code

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

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

- Draw two columns:
  - Global space
  - Heap space
- Draw both the
  - Variables created
  - Objects (folders) created
- Put each in right space
- If a variable changes
  - Mark out the old value
  - Write in the new value
### Global Space

<table>
<thead>
<tr>
<th>b</th>
<th>586790</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>4356712</td>
</tr>
<tr>
<td>d</td>
<td>4356712</td>
</tr>
</tbody>
</table>

### Heap Space

<table>
<thead>
<tr>
<th>586790</th>
<th>CongressMember</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>'Jack'</td>
</tr>
<tr>
<td><strong>init</strong>(n)</td>
<td><strong>str</strong>()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>4356712</th>
<th>CongressMember</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>'Senator John'</td>
</tr>
<tr>
<td></td>
<td>'Senator Clint'</td>
</tr>
<tr>
<td><strong>init</strong>(n)</td>
<td><strong>str</strong>()</td>
</tr>
</tbody>
</table>

If property overridden, same in both partitions

Note setter always puts string 'Senator' in front
class Senator(CongressMember):
    _state = "" # Senator's state

@property
def state(self):
    return self._state

@property
def name(self):
    return self._name

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

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

def __str__(self):
    return (super(Senator, self).__str__() + ' of ' + self.state)
What is on the Exam?

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

In that order
Next is not on Prelim, but on Final
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.'
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.
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 second.
Ending second.
Ending first.