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)

We will not get locked out!

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)

Recursive Function

```
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

```
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?

Call Stack Question

```
def skip(s):
    """Returns: copy of s
    Odd letters dropped"""
1   result = ''
2   if (len(s) % 2 = 1):
3       result = skip(s[1:])
4   elif  len(s) > 0:
5       result = s[0]+skip(s[1:])
6   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!

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

```
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"""
```

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
```

```
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
```

This problem is way to long for an exam

```
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
```

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

```
>>> 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

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

Bonus Question: Dispatch-On-Type

```
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)?

Bonus Question: Dispatch-On-Type

```
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)?