CS 1110

Prelim 2 Review
Fall 2015
Exam Info

• Prelim 2: 7:30–9:00PM, Thursday, Nov. 12th
  ▪ Last name A – J in Uris G01
  ▪ Last name K – Z in Statler Auditorium
  ▪ SDS Students will get an e-mail

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

• Grades will be released before next class
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)**
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"
```

11/8/15 Prelim 2 Review
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?
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:])
```

11/8/15 Prelim 2 Review
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!
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 Stack Question

Call: skip('abc')

• Call: skip('abc')
  skip  s  'abc'  3
  result  ""  

• Call: skip('bc')
  skip  s  'bc'  5
  result  ""  

• Call: skip('c')
  skip  s  'c'  3
  result  ""  

• Call: skip('')
  skip  s  ''  3
  result  ""  RETURN  ""

11/8/15 Prelim 2 Review
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 Stack Question

Call: skip('abc')
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

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

evaluate([2], 4) is 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
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"
"""
```python
def max_cols(table):
    """Returns: Row with max value of each column
    Precondition: table is a NONEMPTY 2D list of floats"""
    # Use the fact that table is not empty
    result = table[0][:] # Make a copy, do not modify table.
    # Loop through rows, then loop through columns
    for row in table:
        for k in range(len(row))
            if row[k] > result[k]
                result[k] = row[k]
    return result
```
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 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
def getName(self):
    return self._name

def setName(self, value):
    assert value is None or type(value) == str
    self._name = value
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
....
def getEmail(self):
    return self._email

def setEmail(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: 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
....
def getBorn(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: 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
    ...

    # DEFINE INITIALIZER HERE
    def __init__(self, n, y, e=None):
        assert type(y) == int and (y > 1900 or y == -1)
        self.setName(n)  # Setter handles asserts
        self.setEmail(e)  # Setter handles asserts
        self._born = y    # 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: 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
    ...
    # DEFINE INITIALIZER 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 (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
```python
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
    def getLevel(self):
        return self._level

    def setLevel(self, value):
        assert type(value) == str
        assert (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 (in addition to Customer):
    _level: level of preference [One of 'bronze', 'silver', 'gold'] '''

# DEFINE GETTERS/SETTERS HERE
...
# DEFINE INITIALIZER HERE
def __init__(self, n, y, e=None, l='bronze'):
    Customer.__init__(self, n, y, e)
    self.setLevel(l)  # Setter handles asserts

# OVERLOAD __str__() OPERATOR HERE
def __str__(self):
    return Customer.__str__(self) + ', ' + self._level

Actual Exam Question
will not be this long.
Just for this practice.

explicit calls uses method
in parent class as helper
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

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

    def __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

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

    def __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
Senator John

'Senator Clint'

'NY'

'Jack'

CongressMember

Senator

__init__(n)  getName()
__str__()  setName(value)

__init__(n,s)  getState()
__str__()  setName(value)
Senator

Clint

NY

Jack

Prelim 2 Review 31
```
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

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

    def __str__(self):
        return (Senator.__str__(self) +
                ' of ' + self.state)
```

---

**Method Overriding**

**Heap Space**

```
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

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

    def __str__(self):
        return (Senator.__str__(self) +
                ' of ' + self.state)
```
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)
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.'
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.
Caught at second.
Ending second.
Ending first.
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

11/8/15 Prelim 2 Review
Any More Questions?