CS 1110

Prelim 1 Review
Fall 2014
Exam Info

- Prelim 1: 7:30–9:00PM, Thursday, October 16th
  - Last name A – Gr in Ives 305
  - Last name Gu – z in Statler Auditorium
  - SDS Students will get an e-mail
- To help you study:
  - Study guides, review slides are online
  - Solutions to Assignment 2 are in CMS
- Arrive early! Helps reducing stress
Grading

• We will announce *approximate* letter grades
  ▪ We adjust letter grades based on all exams
  ▪ But no hard guidelines (e.g. mean = grade X)
  ▪ May adjust borderline grades again at final grades

• Use this to determine whether you want to drop
  ▪ **Drop deadline** is next day, October 17th
  ▪ **Goal**: Have everyone graded by noon of that day
  ▪ Will definitely notify you if you made less than C
What is on the Exam?

• **Five** Questions out of Six Topics:
  - String slicing functions (A1)
  - Call frames and the call stack (A2)
  - Functions on mutable objects (A3)
  - Testing and debugging (A1, Lab 3, Lec. 10)
  - Lists and For-Loops (Lab 6 and 7)
  - Short Answer (Terminology)

• + 2 pts for writing your name and net-id
What is on the Exam?

• String slicing functions (A1)
  ▪ Will be given a function specification
  ▪ Implement it using string methods, slicing

• Call frames and the call stack (A2)

• Functions on mutable objects (A3)

• Testing and debugging (A1, Lab 3, Lecture 10)

• Lists and For-Loops (Lab 6 and 7)

• Short Answer (Terminology)
def make_netid(name,n):

    """Returns a netid for name with suffix n

    Netid is either two letters and a number (if the student has no
    middle name) or three letters and a number (if the student has
    a middle name). Letters in netid are lowercase.

    Example: make_netid('Walker McMillan White',2) is 'wmw2'
    Example: make_netid('Walker White',4) is 'ww4'

    Precondition: name is a string either with format '<first-name>
    <last-name>' or '<first-name> <middle-name> <last-name>'';
    names are separated by spaces. n > 0 is an int."""
# Useful String Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.find(s1)</td>
<td>Returns first position of s1 in s; -1 if not there.</td>
</tr>
<tr>
<td>s.rfind(s1)</td>
<td>Returns LAST position of s1 in s; -1 if not there.</td>
</tr>
<tr>
<td>s.lower()</td>
<td>Returns copy of s with all letters lower case</td>
</tr>
<tr>
<td>s.upper()</td>
<td>Returns copy of s with all letters upper case</td>
</tr>
</tbody>
</table>

- We will give you any methods you need
- But you must know how to slice strings!
```python
def make_netid(name, n):
    """Returns a netid for name with suffix n."""
    name = name.lower()  # switch to lower case
    fpos = name.find(' ')  # find first space
    first = name[:fpos]
    last = name[fpos+1:]
    mpos = last.find(' ')  # see if there is another space
    if mpos == -1:
        return first[0] + last[0] + str(n)  # remember, n is not a string
    else:
        middle = last[:mpos]
        last = last[mpos+1:]
        return first[0] + middle[0] + last[0] + str(n)
```

What is on the Exam?

• String slicing functions (A1)
• Call frames and the call stack (A2)
  ▪ **Very** similar to A2 (see solution in CMS)
  ▪ May have to draw a full call stack
  ▪ See lectures 4 and 9 (slide typos corrected)
• Functions on mutable objects (A3)
• Testing and debugging (A1, Lab 3, Lecture 10)
• Lists and For-Loops (Lab 6 and 7)
• Short Answer (Terminology)
Call Stack Example

• Given functions to right
  - Function `fname()` is not important for problem
  - Use the numbers given

• Execute the call: `lname_first('John Doe')`

• Draw **entire** call stack when helper function `lname` completes line 1
  - Draw nothing else

```python
def lname_first(s):
    """**Precondition:** s in the form
    <first-name> <last-name>"""
    first = fname(s)
    last = lname(s)
    return last + ',' + first

def lname(s):
    """**Prec:** see last_name_first""
    end = s.find(' ')
    return s[end+1:]
```

10/9/14 Prelim 1 Review
Call Stack Example: `lname_first('John Doe')`

```
def lname_first(s):
    """Precondition: s in the form <first-name> <last-name>"
    first = fname(s)
    last = lname(s)
    return last + ',' + first

def lname(s):
    """Prec: see last_name_first"
    end = s.find(' ')
    return s[end+1:]
```
Call Stack Example: `lname_first('John Doe')`

```
def lname_first(s):
    first = fname(s)
    last = lname(s)
    return last + ', ' + first
```

**Precondition:** `s` in the form `<first-name> <last-name>`

```
def lname(s):
    end = s.find(' ')  
    return s[end+1:]
```

No variable `last`. Line 2 is not complete.

Line 1 is **complete**. Counter is next line.
Example with a Mutable Object

```python
def cycle_left(p):
    """Cycle coords left
    Precondition: p a point""
    temp = p.x
    p.x = p.y
    p.y = p.z
    p.z = temp

>>> p = Point(1.0, 2.0, 3.0)
>>> cycle_left(p)
```

- May get a function on a mutable object
  ```
  >>> p = Point(1.0, 2.0, 3.0)
  >>> cycle_left(p)
  ```

- You are not expected to come up w/ the “folder”
  - Will provide it for you
  - You just track changes

- **Diagram all steps**
Example with a Mutable Object

```python
def cycle_left(p):
    """Cycle coords left
    Precondition: p a point"
    temp = p.x
    p.x = p.y
    p.y = p.z
    p.z = temp

>>> p = Point(1.0,2.0,3.0)
>>> cycle_left(p)
```
Example with a Mutable Object

```python
def cycle_left(p):
    '''Cycle coords left
    Precondition: p a point'''
    temp = p.x
    p.x = p.y
    p.y = p.z
    p.z = temp

>>> p = Point(1.0,2.0,3.0)
>>> cycle_left(p)
```

10/9/14 Prelim 1 Review
```python
def cycle_left(p):
    """Cycle coords left
    Precondition: p a point"
    temp = p.x
    p.x = p.y
    p.y = p.z
    p.z = temp

>>> p = Point(1.0,2.0,3.0)
>>> cycle_left(p)
```

Example with a Mutable Object
Example with a Mutable Object

```python
def cycle_left(p):
    """Cycle coords left
    Precondition: p a point"
    temp = p.x
    p.x = p.y
    p.y = p.z
    p.z = temp

>>> p = Point(1.0, 2.0, 3.0)
>>> cycle_left(p)
```
Example with a Mutable Object

def cycle_left(p):
    """Cycle coords left
Precondition: p a point"
    temp = p.x
    p.x = p.y
    p.y = p.z
    p.z = temp

>>> p = Point(1.0,2.0,3.0)
>>> cycle_left(p)
def cycle_left(p):
    """Cycle coords left
    Precondition: p a point"""
    temp = p.x
    p.x = p.y
    p.y = p.z
    p.z = temp

>>> p = Point(1.0, 2.0, 3.0)

>>> cycle_left(p)
def cycle_left(p):
    """Cycle coords left
    Precondition: p a point"
    temp = p.x
    p.x = p.y
    p.y = p.z
    p.z = temp

>>> p = Point(1.0,2.0,3.0)
>>> cycle_left(p)

>>> p = Point(1.0,2.0,3.0)
What is on the Exam?

• String slicing functions (A1)
• Call frames and the call stack (A2)
• Functions on mutable objects (A3)
  ▪ Given an object type (e.g. class)
  ▪ Attributes will have invariants
  ▪ Write a function respecting invariants
• Testing and debugging (A1, Lab 3, Lecture 10)
• Lists and For-Loops (Lab 6 and 7)
• Short Answer (Terminology)
Example from Assignment 3

- **Class: RGB**
  - Constructor function: RGB(r,g,b)
  - Remember constructor is just a function that gives us back a mutable object of that type
  - Attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Invariant</th>
</tr>
</thead>
<tbody>
<tr>
<td>red</td>
<td>int, within range 0..255</td>
</tr>
<tr>
<td>green</td>
<td>int, within range 0..255</td>
</tr>
<tr>
<td>blue</td>
<td>int, within range 0..255</td>
</tr>
</tbody>
</table>
def lighten(rgb):
    """Lighten each attribute by 10%
    Attributes get lighter when they increase.
    Precondition: rgb an RGB object"
    pass # implement me
def lighten(rgb):
    """Lighten each attribute by 10%"""
    red = rgb.red   # puts red attribute in local var
    red = 1.1*red   # increase by 10%
    red = int(round(red,0))  # convert to closest int
    rgb.red = min(255,red)   # cannot go over 255
    # Do the others in one line
    rgb.green = min(255,int(round(1.1*rgb.green,0))))
    rgb.blue = min(255,int(round(1.1*rgb.blue,0))))
Another Example

• Class: Length
  ▪ Constructor function: Length(ft,in)
  ▪ Remember constructor is just a function that gives us back a mutable object of that type
  ▪ Attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Invariant</th>
</tr>
</thead>
<tbody>
<tr>
<td>feet</td>
<td>int, non-negative, = 12 in</td>
</tr>
<tr>
<td>inches</td>
<td>int, within range 0..11</td>
</tr>
</tbody>
</table>
Function that Does Not Modify Object

```python
def difference(len1, len2):
    """Returns: Difference between len1 and len2
    Result is returned in inches
    Precondition: len1 and len2 are length objects
    len1 is longer than len2"

    pass  # implement me
```

def difference(len1,len2):

    """Returns: Difference between len1 and len2
    Result is returned in inches
    Precondition: len1 and len2 are length objects
    len1 is longer than len2""

    feetdif = (len1.feet-len2.feet)*12
    inchdif = len1.inches-len2.inches  # may be negative

    return feetdif+inchdif
What is on the Exam?

- String slicing functions (A1)
- Call frames and the call stack (A2)
- Functions on mutable objects (A3)
- Testing and debugging (A1, Lab 3, Lecture 10)
  - Coming up with test cases
  - Tracing program flow
  - Understanding assert statements
- Lists and For-Loops (Lab 6 and 7)
- Short Answer (Terminology)
def pigify(w):

"""Returns: copy of w converted to Pig Latin
'y' is a vowel if it is not the first letter
If word begins with a vowel, append 'hay'
If word starts with 'q', assume followed by 'u';
move 'qu' to the end, and append 'ay'
If word begins with a consonant, move all
consonants up to first vowel to end and add 'ay'
Precondition: w contains only (lowercase) letters"""
Picking Test Cases

```python
def pigify(w):
    """Returns: copy of w converted to Pig Latin""
```

• Test Cases (Determined by the rules):
  - are => arehay (Starts with vowel)
  - quiet => ietquay (Starts with qu)
  - ship => ipshay (Starts with consonant(s))
  - bzzz => bzzzzay (All consonants)
  - yield => ieldyay (y as consonant)
  - byline => ylinebay (y as vowel)
### Tracing Control Flow

<table>
<thead>
<tr>
<th>def first(n):</th>
<th>def second(n):</th>
</tr>
</thead>
<tbody>
<tr>
<td>print 'Starting first'</td>
<td>print 'Starting second'</td>
</tr>
<tr>
<td>try:</td>
<td>try:</td>
</tr>
<tr>
<td>second(n)</td>
<td>assert n &lt;= 0, 'is not &lt;= 0'</td>
</tr>
<tr>
<td>print 'Done first try'</td>
<td>print 'Done second try'</td>
</tr>
<tr>
<td>except:</td>
<td>except:</td>
</tr>
<tr>
<td>print 'In first except'</td>
<td>print 'In second except'</td>
</tr>
<tr>
<td>print 'Ending first'</td>
<td>assert n &gt;= 0, 'not &gt;= 0'</td>
</tr>
<tr>
<td></td>
<td>print 'Ending second'</td>
</tr>
</tbody>
</table>

**What is printed during the call first(-1)?**

Not guaranteed to have a try-except. Might have an if or a for-loop instead. But this example is the hardest type.
Tracing Control Flow

```
def first(n):
    print 'Starting first'
    try:
        second(n)
        print 'Done first try'
    except:
        print 'In first except'
    print 'Ending first'

def second(n):
    print 'Starting second'
    try:
        assert n <= 0, 'is not <= 0'
        print 'Done second try'
    except:
        print 'In second except'
    assert n >= 0, 'not >= 0'
    print 'Ending second'
```

What is printed during the call `first(-1)`?

- 'Starting first'
- 'Starting second'
- 'Done second try'
- 'In first except'
- 'Ending first'
What is printed during the call first(1)?
Tracing Control Flow

def first(n):
    print 'Starting first'
    try:
        second(n)
        print 'Done first try'
    except:
        print 'In first except'
    print 'Ending first'

def second(n):
    print 'Starting second'
    try:
        assert n <= 0, 'is not <= 0'
        print 'Done second try'
    except:
        print 'In second except'
    assert n >= 0, 'not >= 0'
    print 'Ending second'

What is printed during the call first(1)?

'Starting first'
'Starting second'
'In second except'
'Ending second'
'Done first try'
'Ending first'

Error messages do not show up if an except forces a recovery

Error

Recovers; no error msg
```python
def first(n):
    print 'Starting first'
    try:
        second(n)
        print 'Done first try'
    except:
        print 'In first except'
    print 'Ending first'

def second(n):
    print 'Starting second'
    try:
        assert n <= 0, 'is not <= 0'
        print 'Done second try'
    except:
        print 'In second except'
    assert n >= 0, 'not >= 0'
    print 'Ending second'
```

What is printed during the call `first(0)`?
Tracing Control Flow

```
def first(n):
    print 'Starting first'
    try:
        second(n)
        print 'Done first try'
    except:
        print 'In first except'
    print 'Ending first'

def second(n):
    print 'Starting second'
    try:
        assert n <= 0, 'is not <= 0'
    except:
        print 'In second except'
    print 'Done second try'
    assert n >= 0, 'not >= 0'
    print 'Ending second'
```

What is printed during the call `first(0)`?

'Starting first'
'Starting second'
'Done second try'
'Ending second'
'Done first try'
'Ending first'

Normal execution with no errors happening
What is on the Exam?

• String slicing functions (A1)
• Call frames and the call stack (A2)
• Functions on mutable objects (A3)
• Testing and debugging (A1, Lab 3, Lecture 10)
• Lists and For-Loops (Lab 6 and 7)
  ▪ Given a function specification
  ▪ Implement it using a for-loop
  ▪ Challenge is how to use accumulators
• Short Answer (Terminology)
### Useful List Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x.index(a)</td>
<td>Returns first position of a in x; error if not there</td>
</tr>
<tr>
<td>x.append(a)</td>
<td>Modify x to add element a to the end</td>
</tr>
<tr>
<td>x.insert(a,k)</td>
<td>Modify x to put a at position k (and move rest to right)</td>
</tr>
<tr>
<td>x.remove(a)</td>
<td>Modify x to remove first occurrence of a</td>
</tr>
<tr>
<td>x.sort()</td>
<td>Modify x so that elements are in sorted order</td>
</tr>
</tbody>
</table>

- We will give you any methods you need
- But you must know how to slice lists!
def replace(thelist, a, b):
    """Returns: COPY of thelist with all occurrences of a replaced by b.
    Example: replace([1,2,3,1], 1, 4) = [4,2,3,4].
    Precondition: thelist is a list of ints; a and b are ints"
    return [] # Stub return. IMPLEMENT ME
def replace(thelist,a,b):
    """Returns: COPY of thelist with all occurrences of a replaced by b.
    Example: replace([1,2,3,1], 1, 4) = [4,2,3,4].
    Precondition: thelist is a list of ints; a and b are ints""
    result = []  # Accumulator
    for x in thelist:
        if x == a:
            result.append(b)
        else:
            result.append(x)
    return result
def pairswap(seq):
    """MODIFIES thelist, swapping each two elements with each other
    Example: if a = [0,2,4,5], pairswap(a) makes a into [2,0,5,4]
    if a = [1,2], pairswap(a) turns a into [2,1]
    Precondition: thelist is a list with an even number of elements."""
    pass  # implement me
For-Loop in a Procedure

```python
def pairswap(thelist):
    """MODIFIES thelist, swapping each two elements with each other
    Example: if a = [0,2,4,5], pairswap(a) makes a into [2,0,5,4]
    if a = [1,2], pairswap(a) turns a into [2,1]
    Precondition: thelist is a list with an even number of elements."""
    for k in range(len(thelist)):
        if k % 2 == 0:
            tmp = thelist[k]  # Store old value
            thelist[k] = thelist[k+1]  # Get next value
        else:
            thelist[k] = tmp  # Value stored in previous step
```

10/9/14 Prelim 1 Review
What is on the Exam?

- String slicing functions (A1)
- Call frames and the call stack (A2)
- Functions on mutable objects (A3)
- Testing and debugging (A1, Lab 3, Lecture 10)
- Lists and For-Loops (Lab 6 and 7)
- Short Answer (Terminology)
  - See the study guide
  - Look at the lecture slides
  - Read relevant book chapters

In that order
Any More Questions?
Good Luck!