Announcements

Extended profs. office hours
Thursday 9:05-12:05
Thurston 102

Prelim study tips
See Piazza @168
Exam Info

• Prelim 1: 7:30–9:00PM, Thursday, March 7
  ▪ Location: Kennedy 116 (Call Auditorium)
• To help you study:
  ▪ Study guides, review slides are online
  ▪ Solutions to Assignment 2 are online
• Arrive early! Helps reduce stress
• Grades will be released as soon as practical
  ▪ CMS will let you know; hopefully by the weekend
  ▪ Possibly not by drop deadline
What is on the Exam?

• Five Topics (+2pts for name, NetID, lab):
  ▪ String manipulation (A1, Lab 2)
  ▪ Call frames and the call stack (A2)
  ▪ Functions on mutable objects (A3, Lab 3 & 5)
  ▪ Testing and debugging (A1, Lab 3)
  ▪ 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.index(s1)</td>
<td>Returns first position of s1 in s; error if not there</td>
</tr>
<tr>
<td>s.count(s1)</td>
<td>Returns number of occurrences of s1 in s</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>
<tr>
<td>s.strip()</td>
<td>Returns copy of s with whitespace removed</td>
</tr>
</tbody>
</table>

- We will give you any methods you need
- But you must know how to slice strings!
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]+`n` # remember, n is not a string
    else:
        middle = last[:mpos]
        last = last[mpos+1:]
        return first[0]+middle[0]+last[0]+`n`
Given functions to right
- Function \texttt{fname}() is not important for problem
- Use the numbers given

Execute the call:
\texttt{lname\_first('John Doe')}\newpage

Draw \textbf{entire} call stack when helper function \texttt{lname} completes line 1
- Draw nothing else

\texttt{def \textbf{name}\_first(s):}

\hspace{1em} "\textbf{Precondition}: s in the form \\
\hspace{1em} \texttt{<first-name> <last-name>}"

\hspace{1em} 1 \hspace{1em} \texttt{first} = \texttt{fname(s)}

\hspace{1em} 2 \hspace{1em} \texttt{last} = \texttt{lname(s)}

\hspace{1em} 3 \hspace{1em} \texttt{return last + ', ' + first}

\texttt{def \textit{name}(s):}

\hspace{1em} "\textbf{Prec}: see \texttt{last\_name\_first}"

\hspace{1em} 1 \hspace{1em} \texttt{end} = \texttt{s.find(' ')}

\hspace{1em} 2 \hspace{1em} \texttt{return s[end+1:]}
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:]
```

**Prelim 1 Review**

<table>
<thead>
<tr>
<th>Call Stack</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>lname_first</strong>:</td>
<td></td>
</tr>
<tr>
<td>s</td>
<td>'John Doe'</td>
</tr>
<tr>
<td>first</td>
<td>'John'</td>
</tr>
<tr>
<td>last</td>
<td></td>
</tr>
<tr>
<td><strong>lname</strong>:</td>
<td></td>
</tr>
<tr>
<td>s</td>
<td>'John Doe'</td>
</tr>
<tr>
<td>end</td>
<td>4</td>
</tr>
</tbody>
</table>
Call Stack Example: `lname_first('John Doe')`

```python
def lname_first(s):
    # 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(' ') + 1
    return s[end:]
```

Omitting this is okay. Line 2 is not complete.

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

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

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

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

```python
def shift(p):
    '''Shift 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)
>>> shift(p)
```

Prelim 1 Review
Objects: example from A3

- Type: RGB in colormodel.py
  - Constructor call: colormodel.RGB(r,g,b)
  --- assuming prior line import colormodel, and r, g, b are ints in interval 0..255

<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 increase10(rgb):
    """Increase each attribute by 10% (up to 255)
    Precondition: rgb an RGB object"""
    pass  # implement me
Sample step

# store in t the value of rgb's red attribute
# Which of these is correct? What do the others do?

```
t = colormodel.RED
```
```
t = rgb.red()
```
```
t = rgb.r
```
```
t = rgb.red
```
```
t = colormodel.rgb.red
```
Sample step – answer in bold

# store in t the value of rgb's red attribute
# Which of these is correct? What do the others do?
t = colormodel.RED # refers to something in colormodel

t = rgb.red() # call to function "in" rgb

t = rgb.r # attribute r of rgb, but there's no such attribute

t = rgb.red # <obj name>.<attr name> is the way to access

t = colormodel.rgb.red # refers to something in rgb in
  #colormodel
Should `increase10` have return statement?
Should increase10 have return statement?

No; the spec doesn't say so.
def increase10(rgb):
    
    """Increase each attribute by 10% (up to 255)"""

    red = rgb.red  # puts red attribute value in local var
    red = 1.1*red  # increase by 10%
    red = int(round(red))  # 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))))
    rgb.blue = min(255,int(round(1.1*rgb.blue))))
def multcap(x):
    """Returns: min of nearest int to x*1.1 and 255.
    Precond: x a number"
    return min(int(round(x*1.1)), 255)

def increase10(rgb):
    """Increase each attribute by 10% (up to 255)"
    # alternate solution with massive map
    alist = map(multcap, [rgb.red, rgb.green, rgb.blue])
    rgb.red = alist[0]
    rgb.green = alist[1]
    rgb.blue = alist[2]
Code up a test case for increase10
(assume in module reviewp1)

testcolor = colormodel.RGB(10,100,255)

reviewp1.increase10(testcolor)

cunittest2.assert_equals(colormodel.RGB(11,110,255),
                       testcolor)

Why not this?
cunittest2.assert_equals(colormodel.RGB(11,110,255),
reviewp1.increase10(testcolor))

No return value to compare against.
• Type: Length in module `ell`
  - Constructor call: `ell.Length(ft, in)`

--- assuming prior line `import ell` and `ft` and `in` are ints, given:

<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 inclusive</td>
</tr>
</tbody>
</table>

```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
```

Prelim 1 Review
Function that Does Not Modify Object

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
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 (or to end of w, if
none) to end and add 'ay'

    Precondition: w contains only (lowercase)
letters, and at least one letter"
"""
Picking Test Cases

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

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

```python
def first(x):
    1. print 'Starting first.'
    2. second(x)
    3. print 'Ending first'

def second(x):
    1. print 'Starting second.'
    2. if third(x):
       3. pass
       4. else:
          5. print 'Caught False at second'
    6. print 'Ending second'

def third(x):
    1. print 'Starting third.'
    2. print 'Ending third.'
    3. return x < 1
```

What is the output of `first(2)`?
Tracing Control Flow

```python
def first(x):
    1. print 'Starting first.'
    2. second(x)
    3. print 'Ending first'

def second(x):
    1. print 'Starting second.'
    2. if third(x):
        3. pass
    4. else:
        5. print 'Caught False at second'
    6. print 'Ending second'

def third(x):
    1. print 'Starting third.'
    2. print 'Ending third.'
    3. return x < 1
```

What is the output of `first(2)`?

'Starting first.'
'Starting second.'
'Starting third.'
'Ending third'
'Caught False at second'
'Ending second'
'Ending first'
Tracing Control Flow

```python
def first(x):
    1. print 'Starting first.'
    2. second(x)
    3. print 'Ending first'

def second(x):
    1. print 'Starting second.'
    2. if third(x):
    3.     pass
        elif:
    5.     print 'Caught False at second'
    6. print 'Ending second'

def third(x):
    1. print 'Starting third.'
    2. print 'Ending third.'
    3. return x < 1
```

What is the output of `first(0)`?
Tracing Control Flow

```
def first(x):
    1. print 'Starting first.'
    2. second(x)
    3. print 'Ending first'

def second(x):
    1. print 'Starting second.'
    2. if third(x):
       3. pass
    4. else:
       5. print 'Caught False at second'
    6. print 'Ending second'

def third(x):
    1. print 'Starting third.'
    2. print 'Ending third.'
    3. return x < 1
```

What is the output of `first(0)`?

'Starting first.'
'Starting second.'
'Starting third.'
'Ending third'
'Ending second'
'Ending first'
Looking for inspiration?

"""What most schools don't teach: Learn about a new "superpower" that isn't being taught in 90% of US schools."""

https://www.youtube.com/watch?feature=player_embedded&v=nKIu9yen5nc