CS 1110
Prelim 1 Practice/Review Session

Some extra commentary stated in class have been added to these slides in *orange*.
Announcements

• A3 tests due Sat Mar 7; A3 fns due Sun Mar 8
  ▪ Remember academic integrity!!!

• Prelim 1 Tues Mar 10 at 7:30pm. Bring your own pens/pencils/erasers (bring several). *Bring Cornell ID.*

<table>
<thead>
<tr>
<th>Room Code</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>AA1-KQA1: Baker 200</td>
<td></td>
</tr>
<tr>
<td>KQZ1-MM4000: Baker 119</td>
<td></td>
</tr>
<tr>
<td>MM5000-RC799: Baker 135</td>
<td></td>
</tr>
<tr>
<td>RC800-TB270: Baker 219</td>
<td></td>
</tr>
<tr>
<td>TC1-ZZZ9999: Baker 335</td>
<td></td>
</tr>
</tbody>
</table>

• Read Prelim 1 Study Guide. Note spring different from fall

• Mar 10 lecture and lab time ➔ office hours
Exam Topics

• String slicing functions
• Call frames and the call stack
• Functions on mutable objects
• Testing and debugging
• Conditionals
• Lists and simple iteration

Today:

• Start with lists and iteration—not in posted old review slides
• Testing and debugging
• Other topics if time allows
def count_non_space_chars(myList):
    """Returns: number of non-space characters in the strings in myList.
    Example: count_non_space_chars(['U ', 'r', '', ' gr8']) returns 5
    Precondition: myList is a list of strings. Each string in myList can contain only spaces, letters, digits."""

You don’t need nested loops to solve this problem, but it’s ok to use them if you want.

For Prelim 1 you should be able to read and understand nested loops, but you won’t need to write them.
# 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.count(s1)</td>
<td>Returns number of times s1 occurs 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>
</tbody>
</table>

- We will give you any methods you need
- But you must know how to slice strings!

As stated during class, you **should know** the methods that we actually have used in assignments and labs. We will give you the less-frequently used methods on the exam.
def count_non_space_chars(myList):
    """Returns: number of non-space characters in the strings in myList. Example: count_non_space_chars(['U ', 'r', '', ' gr8']) returns 5
Precondition: myList is a list of strings. Each string in myList can contain only spaces, letters, digits."""
def count_non_space_chars(myList):
    """Returns: number of non-space characters in the strings in myList.
Example: count_non_space_chars(['U ', 'r', 'gr8']) returns 5
Precondition: myList is a list of strings. Each string in myList can contain only spaces, letters, digits."""
    count = 0
    for s in myList:
        numSp = s.count(' ')
        numNonSp = len(s) - numSp
        count = count + numNonSp
    return count

Remember:
• Be goal oriented—start with return statement (if it is needed). Work your way back up. Be flexible.
• Name a variable for any value you need but don’t know yet
• For-loop
• Accumulation pattern
• How to call string methods
def inflate(myList, p_percent):
    """Inflate each number in myList by p_percent while maintaining the type (int or float). For any int in myList, round down the inflation. Precondition: myList is a list of positive numbers (int and/or float). Precondition: p_percent is a positive number (int or float)."""

An example:

>>> L = [100, 100.0, 1, 1.0]
>>> p = 1.6
>>> inflate(L,p)
>>> L
[101, 101.6, 1, 1.016]
def inflate(myList, p_percent):
    """Inflate each number in myList by p_percent while maintaining the type (int or float). For any int in myList, round down the inflation.
    Precondition: myList is a list of positive numbers (int and/or float).
    Precondition: p_percent is a positive number (int or float)."""
def inflate(myList, p_percent):
    """Inflate each number in myList by p_percent while maintaining the type (int or float). For any int in myList, round to the nearest integer."
    p_frac = p_percent / 100
    for k in range(len(myList)):
        delta = myList[k] * p_frac
        if type(myList[k]) == int:
            delta = int(delta)
        myList[k] += delta

Remember:
- Give yourself an example if question doesn’t provide one
- Using for-loop on list: do you need to modify list? If so you need the indices—use range
- List syntax
- How to work with types (ops, checking, casting)
- In general, read specs again after finishing code. Did you really solve problem asked?
Constructing test cases

```python
def before_space(s):
    """Returns: the substring before the first space character in string s.
    Precondition: string s contains at least one space.""
```

Come up with at least three distinct test cases. Write the test input, expected output, and rationale.
def before_space(s):
    """Returns: the substring before the first space character in string s.
    Precondition: string s contains at least one space."""

• First think about the pre-condition to see what we know about the string s
  • It has at least one space char → it can have more than one
    → adjacent? non-adjacent?
  • No precondition on where the space char appears in s
    → can be anywhere
      → start? middle? end?
• With these ideas, can construct distinct test cases with rationales
  for each one
Constructing test cases

def before_space(s):
    """Returns: the substring before the first space character in string s.
    Precondition: string s contains at least one space."""

Examples:
• "  abc"    – single space char at the start
• "abc  "    – single space char at the end
• "a  bc"    – single space char in the “middle” (not start or end)
• "  abc"    – many space chars at the start
• "abc  "    – many space chars at the end
• "ab   c"    – many space chars in the middle
• "a  b  c"    – many non-adjacent space chars
What should I be testing?

Common Cases: typical usage

Edge Cases: live at the boundaries

- Target location in list: first, middle, last elements
- Input size: 0, 1, 2, many (length of lists, strings, etc.)
- Input Orders: max(big, small), max(small, big)...
- Element values: negative/positive, zero, odd/even
- Element types: int, float, str, etc.
- Expected results: negative, 0, 1, 2, many

Not all categories/cases apply to all functions.
Use your judgement!
Functions on Objects

• Class: Rect
  - Constructor function: Rect(x,y,width,height)
  - Remember constructor is just a function that gives us an object of that type and returns its identifier

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>float, x coord of lower left corner</td>
</tr>
<tr>
<td>y</td>
<td>float, y coord of lower left corner</td>
</tr>
<tr>
<td>width</td>
<td>float, &gt; 0, width of rectangle</td>
</tr>
<tr>
<td>height</td>
<td>float, &gt; 0, height of rectangle</td>
</tr>
</tbody>
</table>
def move(r, xc, yc):
    """Set the attributes of Rect `r` such that its center lies on the x- and y-coordinates `xc` and `yc`, respectively.
    Precondition:  `r` is a Rect object.
    Precondition:  `xc`, `yc` are each a float."""
def move(r, xc, yc):
    """Set the attributes of Rect `r` such that its center lies on the x- and y-coordinates `xc` and `yc`, respectively.
    Precondition:  r is a Rect object.
    Precondition:  xc, yc are each a float."""

(20-8/2, 50-6/2)
def move(r, xc, yc):
    """Set the attributes of Rect `r` such that its center lies on the x- and y-coordinates `xc` and `yc`, respectively.
    Precondition: r is a Rect object.
    Precondition: xc, yc are each a float."
    r.x = xc - r.width/2
    r.y = yc - r.height/2

Remember:
- Draw a diagram to help yourself think
- Label the diagram with example/known values. Then generalize labels using parameter and attribute names.
- Important problem solving step! First use example values to understand the problem and figure out relationships among knowns and unknowns.
- Dot-notation for accessing attributes of an object
Good Luck!