CS 61A Lecture 11

Friday, September 26
Announcements

• Midterm 1 has been graded...
  ▪ Many of you did very well. Nice work!
  ▪ High scores on homework and projects balance out exam scores
  ▪ Typically, around 3 out of 4 students receive A’s & B’s in 61A
  ▪ Don't fall behind! Come to class (discussion, lab, & office hours)!
  ▪ Regrades are due by Sunday 9/29 @ 11:59pm
• Guerrilla Section 2 is on Saturday. RSVP on Piazza if you want to come!
• Homework 3 due Wednesday 10/1 @ 11:59pm
  ▪ Homework Party on Monday 9/29, time and place TBD
• Optional Hog Contest due Wednesday 10/1 @ 11:59pm
Sequences
The Sequence Abstraction

The sequence abstraction is a collection of behaviors:

- **Length.** A sequence has a finite length.
- **Element selection.** A sequence has an element corresponding to any non-negative integer index less than its length, starting at 0.

There isn't just one sequence class or data abstraction (in Python or in general).

There is built-in syntax associated with this behavior, or we can use functions.

A list is a kind of built-in sequence.
Lists

['Demo']
Lists are Sequences

>>> digits = [1, 8, 2, 8]
>>> len(digits)
4
>>> digits[3]
8

Length. A sequence has a finite length.

Element selection. A sequence has an element corresponding to any non-negative integer index less than its length, starting at 0.

>>> [2, 7] + digits * 2
[2, 7, 1, 8, 2, 8, 1, 8, 2, 8]

>>> pairs = [[10, 20], [30, 40]]
>>> pairs[1]
[30, 40]
>>> pairs[1][0]
30
For Statements

(Demo)
def count(s, value):
    total = 0
    for element in s:
        if element == value:
            total = total + 1
    return total
For Statement Execution Procedure

```
for <name> in <expression>:
    <suite>
```

1. Evaluate the header `<expression>`, which must yield an iterable value (a sequence)

2. For each element in that sequence, in order:
   
   A. Bind `<name>` to that element in the current frame
   
   B. Execute the `<suite>`
Sequence Unpacking in For Statements

A sequence of fixed-length sequences

```python
>>> pairs = [[1, 2], [2, 2], [3, 2], [4, 4]]
>>> same_count = 0

>>> for x, y in pairs:
...     if x == y:
...         same_count += 1
```

Each name is bound to a value, as in multiple assignment

```python
>>> same_count
2
```
Ranges
The Range Type

A range is a sequence of consecutive integers.*

..., -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, ...

Length: ending value - starting value

Element selection: starting value + index

>>> list(range(-2, 2))
[-2, -1, 0, 1]

>>> list(range(4))
[0, 1, 2, 3]

* Ranges can actually represent more general integer sequences.
List Comprehensions

```python
>>> letters = ['a', 'b', 'c', 'd', 'e', 'f', 'm', 'n', 'o', 'p']
>>> [letters[i] for i in [3, 4, 6, 8]]
['d', 'e', 'm', 'o']
```
List Comprehensions

\[
\langle \text{map exp} \rangle \text{ for } \langle \text{name} \rangle \text{ in } \langle \text{iter exp} \rangle \text{ if } \langle \text{filter exp} \rangle \\
\]

Short version: \[
\langle \text{map exp} \rangle \text{ for } \langle \text{name} \rangle \text{ in } \langle \text{iter exp} \rangle \\
\]

A combined expression that evaluates to a list using this evaluation procedure:

1. Add a new frame with the current frame as its parent
2. Create an empty result list that is the value of the expression
3. For each element in the iterable value of \langle \text{iter exp} \rangle:
   A. Bind \langle \text{name} \rangle to that element in the new frame from step 1
   B. If \langle \text{filter exp} \rangle evaluates to a true value, then add the value of \langle \text{map exp} \rangle to the result list
Higher-Order Sequence Functions
Functions that Perform List Comprehensions

```python
def apply_to_all(map_fn, s):
    """Apply map_fn to each element of s."""
    return [map_fn(x) for x in s]

def keep_if(filter_fn, s):
    """List all elements x of s for which filter_fn(x) is true."""
    return [x for x in s if filter_fn(x)]
```

```python
>>> apply_to_all(lambda x: x*3, range(5))
[0, 3, 6, 9, 12]
```

```python
>>> keep_if(lambda x: x>5, range(10))
[6, 7, 8, 9]
```
Reducing a Sequence to a Value

```python
def reduce(reduce_fn, s, initial):
    """Combine elements of s pairwise using reduce_fn, starting with initial.

    E.g., reduce(mul, [2, 4, 8], 1) is equivalent to mul(mul(1, 2), 4), 8).

    >>> reduce(mul, [2, 4, 8], 1)
    64
    ""
    reduced = initial
    for x in s:
        reduced = reduce_fn(reduced, x)
    return reduced
```

reduce_fn is ...
  a two-argument function
s is ...
  a sequence of values that can be the second argument
initial is ...
  a value that can be the first argument

(Demo)
Typical Names for Higher-Order Sequence Functions

apply_to_all is usually called map
keep_if is usually called filter
reduce is usually called reduce (but sometimes fold or accumulate)

map and filter are built into Python, but they don't return lists

reduce is in the standard library in a module called functools

Most Python programmers just use list comprehensions