CS 61A Lecture 10
Announcements
Sequences
The Sequence Abstraction

red, orange, yellow, green, blue, indigo, violet.

0, 1, 2, 3, 4, 5, 6.

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

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 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

```python
>>> 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.

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

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

(Demo)
Sequence Iteration

def count(s, value):
    total = 0
    for element in s:
        if element == value:
            total = total + 1
    return total

Name bound in the first frame of the current environment (not a new frame)

if element == value:
    total = total + 1
return total
For Statement Execution Procedure

\[
\text{for } \langle\text{name}\rangle \text{ in } \langle\text{expression}\rangle:\ \langle\text{suite}\rangle
\]

1. Evaluate the header \langle\text{expression}\rangle, which must yield an iterable value (a sequence)

2. For each element in that sequence, in order:

   A. Bind \langle\text{name}\rangle to that element in the current frame

   B. Execute the \langle\text{suite}\rangle
Sequence Unpacking in For Statements

A sequence of fixed-length sequences

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

A name for each element in a fixed-length sequence

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

```python
>>> same_count = 0

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

>>> 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

\[
[\text{map exp} \ for \ <name> \ in \ <iter \ exp> \ if \ <filter \ exp>]
\]

Short version: \[<map \ exp> \ for \ <name> \ in \ <iter \ exp>\]

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 \(<iter \ exp>\):
   A. Bind \(<name>\) to that element in the new frame from step 1
   B. If \(<filter \ exp>\) evaluates to a true value, then add the value of \(<map \ exp>\) to the result list
Strings
Strings are an Abstraction

Representing data:

'200'    '1.2e-5'    'False'    '[1, 2]'  

Representing language:

""""""And, as imagination bodies forth  
The forms of things to unknown, and the poet's pen  
Turns them to shapes, and gives to airy nothing  
A local habitation and a name.  
""""""

Representing programs:

'curry = lambda f: lambda x: lambda y: f(x, y)'  

(Demo)
String Literals Have Three Forms

```python
>>> 'I am string!'
'I am string!'

>>> "I've got an apostrophe"
"I've got an apostrophe"

>>> '您好'
'您好'

>>> """The Zen of Python
claims, Readability counts.
Read more: import this."""
'The Zen of Python
claims, Readability counts.
Read more: import this.'
```

- Single-quoted and double-quoted strings are equivalent
- A backslash "escapes" the following character
- "Line feed" character represents a new line
**Strings are Sequences**

Length and element selection are similar to all sequences

```python
>>> city = 'Berkeley'
>>> len(city)
8
>>> city[3]
'k'
```

Careful: An element of a string is itself a string, but with only one element!

However, the "in" and "not in" operators match substrings

```python
>>> 'here' in "Where's Waldo?"
True
>>> 234 in [1, 2, 3, 4, 5]
False
>>> [2, 3, 4] in [1, 2, 3, 4, 5]
False
```

When working with strings, we usually care about whole words more than letters
Dictionaries

{"Dem": 0}
Limitations on Dictionaries

Dictionaries are **unordered** collections of key-value pairs

Dictionary keys do have two restrictions:

- A key of a dictionary **cannot be** a list or a dictionary (or any *mutable type*)

- Two **keys cannot be equal**; There can be at most one value for a given key

This first restriction is tied to Python's underlying implementation of dictionaries

The second restriction is part of the dictionary abstraction

If you want to associate multiple values with a key, store them all in a sequence value