Announcements

- Guerrilla Section 2 is on Monday 2/16
- RSVP on Piazza if you want to come!
- Homework 3 due Thursday 2/19 @ 11:59pm (extended)
- Homework Party on Tuesday 2/17 5pm-6:30pm in 2050 VLSB
- Optional Hog Contest due Wednesday 2/18 @ 11:59pm

Sequences

The Sequence Abstraction

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

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]
>>> pairs = [(10, 20), (30, 40)]
>>> pairs[1] # pairs[0][1] + 10
30
```

Sequence iteration

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

For Statements

```python
(Demo)
```
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

[1, 2, 2, 2, 4, 4]

A name for each element in a fixed-length sequence

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

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

same_count = 2

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 Comprehensions

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 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)'

(List Comprehensions)

[<map exp> for <name> in <iter exp> if <filter exp>]

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

Ranges

Ranges can actually represent more general integer sequences.

List Comprehensions

letters = ['a', 'b', 'c', 'd', 'e', 'f', 'm', 'n', 'o', 'p']

[letters[i] for i in [3, 4, 6, 8]]

['d', 'e', 'm', 'o']
### String Literals Have Three Forms

```python
>>> 'I am string!
'I am string!
>>> "I've got an apostrophe"
"I've got an apostrophe"
>>> '''Hello
World'''
'Hello
World'''
```

- Single-quoted and double-quoted strings are equivalent
- A backslash escapes the following character
- A 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!

```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
```

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

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

---

### Dictionaries

**Dictionaries**

- Unordered collections of key-value pairs

```python
{'Dem': 0}
```

---

### Limitations on Dictionaries

Dictionaries are unordered collections of key-value pairs

- Dictionary keys do have two restrictions:
  1. A key of a dictionary cannot be a list or a dictionary (or any mutable type)
  2. 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