Strings, Lists and Sequences

- Sequences are potentially **unbounded**
  - Number of elements inside them is not fixed
- Cannot process with **fixed** number of lines
  - Each line of code can handle at most one element
  - What if # of elements > # of lines of code?
- This is why we used recursion to process them
  - Each function call handles one element
  - Recursive call handles the remainder of sequence
- Is there an easier way?

For Loops: Processing Sequences

```
# Print contents of seq
x = seq[0]
print x
x = seq[1]
print x
...
x = seq[len(seq)-1]
print x
```

- **Remember**:
  - Cannot program …
  - Reason for recursion

The for-loop:

```
for x in seq:
    print x
```

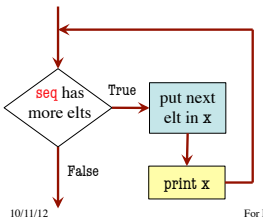
- Key Concepts
  - **loop sequence:** seq
  - **loop variable**: x
  - **body**: print x
  - Also called **repetend**

For Loops: Processing Sequences

The for-loop:

```
for x in seq:
    print x
```

- **loop sequence:** seq
- **loop variable**: x
- **body**: print x



To execute the for-loop:

1. Check if there is a "next" element of **loop sequence**
2. If not, terminate execution
3. Otherwise, put the element in the **loop variable**
4. Execute all of **the body**
5. Repeat as long as 1 is true

More Complex For-Loops

- Combine with a *counter*
  - Variable that increments each time body executed
  - Tracks position in seq
- **Example**:

```
cnt = 0
for x in seq:
    print `x`+' at '+`cnt`
    cnt = cnt + 1 # incr
```

- Nest conditionals inside
  - Body is all indented code
  - Can put other control structures inside the body
- **Example**:

```
nints = 0 # num of ints
for x in seq:
    if type(x) == int:
        nints = nints + 1
```


For Loops Instead of Recursion

```
def deblank(s):
    """Returns: s w/o blanks
       Precondition: s a string"""
    if s == '':
        return s
    # s is not empty
    if s[0] in string.whitespace:
        return deblank(s[1:])
    # s not empty, s[0] not blank
    return (s[0] +
            deblank(s[1..]))
```

```
def no_blanks(s):
    """Returns: s w/o blanks
    Precondition: s a string"""
    result = ''
    # glue nonblanks onto result
    for c in s:
        if not c in string.whitespace:
            result = result+c

    return result
```


For Loops: Processing Ranges of Integers

```
total = 0;
# add the squares of ints
# in range 2..200 to total
total = total + 2*2
total = total + 3*3
...
total = total + 200*200
```

- For each x in the range 2..200, add x*x to total

The for-loop:

```
for x in range(2,201):
    total = total + x*x
```

- **The range function:**
  - range(x): List of ints 0 to x-1
  - range(a,b): List of ints a to b-1

Important Concept in CS: Doing Things Repeatedly

1. Process each item in a sequence
   - Compute aggregate statistics for a dataset, such as the mean, median, standard deviation, etc.
   - Send everyone in a Facebook group an appointment time
2. Perform *n* trials or get *n* samples.
   - A4: draw a triangle six times to make a hexagon
   - Run a protein-folding simulation for 10^6 time steps
3. Do something an unknown number of times
   - CUAUV team, vehicle keeps moving until reached its goal

Dictionaries (Type dict)

Description

- List of key-value pairs
  - Keys are unique
  - Values need not be
- Example: net-ids
  - net-ids are **unique** (a key)
  - names need not be (values)
  - js1 is John Smith (class '13)
  - js2 is John Smith (class '16)
- Many other application

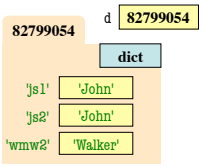
Python Syntax

- Create with format: {k1:v1, k2:v2, ...}
- Keys must be non-mutable
  - ints, floats, bools, strings
  - **Not** lists or custom objects
- Values can be anything
- Example:
  d = {'js1':'John Smith', 'js2':'John Smith', 'wmw2':'Walker White'}

Using Dictionaries (Type dict)

- Access elts. like a list
  - d['js1'] evaluates to 'John'
  - But cannot slice ranges!
- Dictionaries are **mutable**
  - Can reassign values
  - d['js1'] = 'Jane'
  - Can add new keys
  - d['aa1'] = 'Allen'
  - Can delete keys
  - del d['wmw2']

d = {'js1':'John','js2':'John', 'wmw2':'Walker'}

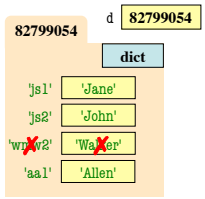


Key-Value order in folder is not important

Using Dictionaries (Type dict)

- Access elts. like a list
  - d['js1'] evaluates to 'John'
  - But cannot slice ranges!
- Dictionaries are **mutable**
  - Can reassign values
  - d['js1'] = 'Jane'
  - Can add new keys
  - d['aa1'] = 'Allen'
  - Can delete keys
  - del d['wmw2']

d = {'js1':'John','js2':'John', 'wmw2':'Walker'}



Deleting key deletes both

Dictionaries and For-Loops

- Dictionaries != sequences
  - Cannot slice them
  - Cannot use in for-loop
- But have methods to give you related sequences
  - Seq of keys: d.keys()
  - Seq of values: d.values()
  - Seq of key-value pairs: d.items()
- Use these in for-loops
  - Example: grades.py

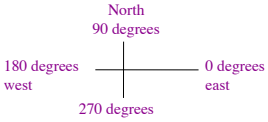
```
for k in d.keys():
    print k
    print d[k]

for v in d.values():
    print v

for k,v in d.items():
    print k
    print v
```


A4: Drawing with the Turtle

- Turtle Attributes
  - x and y: where "Turtle" is
  - heading: direction it faces
  - color: the Turtle pen color
  - drawmode: if True, Turtle draws whenever it moves
- Draw using methods
  - t.forward(s) moves turtle
    - Draws if drawmode True
  - t.left(a), t.right(a) turn
    - a is angle in degrees



Draw equilateral triangle:

```
for x in range(3):
    t.forward(30)
    t.left(120)
```