Recap from last Python lecture

Interpreted, imperative, OO Language
- Everything is an object
- Dynamic Typing

Programs are made up of:
- Expressions
- Statements
  - Assignment
  - if/elif/else
  - while-loops
  - Functions
- Classes (still to come)

Show recap code

Today: Revisit some objects
- Exploit features and build powerful expressions

Base: int, float, complex
Sequence: string, tuple, list

What can sequences do?
Select
- i-th element: s[i]
- subsequence (“slice”): s[i:j]

Update -- For mutable sequences (e.g. Lists)
- Update i-th element: s[i] = e
- Update subsequence: s[i:j] = e

Update subsequence

Update subsequence: s[i:j] = e

- Changes the “object” referred to by s
- May change the length of the sequence
  - Increase: if RHS length > j-i
  - Decrease: if RHS length < j-i

Update subsequence

>>> z = [1,2,3,4,5,6,7,8,9,10]
>>> z[3:6] = ["a","b","c"]
>>> z
[1,2,3,"a","b","c",7,8,9,10]
>>> z[3:6] = ["a", "b"] * 2
>>> z
[1,2,3,"a","b","a","b",7,8,9,10]
>>> z[4:]=[]
>>> z
[1,2,3,"a"]
>>> z[:0] = ["al", "be"]
>>> z
["al","be",1,2,3,"a","b","a","b",7,8,9,10]
What else can sequences do?

Q: Suppose you are given a sequence \( s \). How to find if the element \( x \) appears in \( s \)?

\[ x \text{ in } s \]

Works for any sequence type ...

---

Sequence “contains” \( x \text{ in } s \)

\[ \begin{align*}
& \text{"a" in "cat"} \\
& \text{True} \\
& \text{"a" in "entebbe"} \\
& \text{False} \\
& \text{"a" in ("c", "a", "t")} \\
& \text{True} \\
& \text{2 in [1,2,3,4,5]} \\
& \text{True} \\
& \text{2 in [1,4,"92",2.4]} \\
& \text{False}
\end{align*} \]

---

What can sequences do?

Select
- \( i \)-th element: \( s[i] \)
- Subsequence (“slice”): \( s[i:j] \)

Update -- For mutable sequences (e.g. Lists)
- Update \( i \)-th element: \( s[i] = e \)
- Update subsequence: \( s[i:j] = e \)

Member
- Is an element in a sequence: \( x \text{ in } s \)

---

Doesn’t Python have For-Loops?

Why haven’t we seen For-loops yet?
- Because they are connected to sequences

For-loops are used to iterate over sequences
- Unlike in C, but similar to new Java foreach
- Elegant, powerful mechanism - use it!

\[ \text{for } x \text{ in } s: \]

\[ \text{<BODY>} \]

---

Iteration

\[ \text{for } x \text{ in } s: \]

\[ \begin{align*}
& \text{for } x \text{ in ["Midterms", "ain’t", "cool"]:} \\
& \hspace{1cm} \text{print } x, \text{len}(x)
\end{align*} \]

\[ \begin{align*}
\text{Midterms } 5 \\
\text{ain’t } 5 \\
\text{cool } 4
\end{align*} \]

Works for any sequence ...

\[ \begin{align*}
& \text{for } c \text{ in "chimichanga":} \\
& \hspace{1cm} \text{print } c^3
\end{align*} \]

\[ \begin{align*}
\text{ccc} \\
\text{hhh} \\
\text{iii} \\
\text{mmm }
\end{align*} \]

---

Iteration

\[ \text{for } x \text{ in } s: \]

\[ \begin{align*}
& \text{for } i \text{ in } z: \\
& \hspace{1cm} s = s + i
\end{align*} \]

\[ \text{ERROR} \]

\[ \begin{align*}
& \text{s = 0} \\
& \text{for } i \text{ in } z: \\
& \hspace{1cm} s = s + i
\end{align*} \]

\[ \text{ERROR} \]

\[ \begin{align*}
& \text{s = 0} \\
& \text{for } i \text{ in } z: \\
& \hspace{1cm} s = s + \text{float}(i)
\end{align*} \]

\[ \begin{align*}
& \text{s = 0} \\
& \text{for } i \text{ in } z: \\
& \hspace{1cm} s = s + \text{float}(i)
\end{align*} \]

\[ \begin{align*}
& \text{s = 0} \\
& \text{for } i \text{ in } z: \\
& \hspace{1cm} s = s + \text{float}(i)
\end{align*} \]

Can’t add string to float
- Note that first 4 elts added!
- Dynamic Types!
- Run-time Type Error
Iteration + binding for x,... in S:

If S is a sequence of tuples/sequences, then we can bind to individual elements of “subsequences”

>>> craigslist = [('alien', 3.50), ('dinosaur', 1.90), ('quiz', 100.50), ('quesadilla', 3.00), ('good grade in 130', 'priceless')]
>>> for i, p in craigslist:
    print "One", i, "costs", p
One alien costs 3.5
One dinosaur costs 1.9
One quiz costs 100.5
One quesadilla costs 3.0
One good grade in 130 costs priceless

But lookout!

For-loops are used to iterate over sequences

for x in S:

What if object referred to by S is changed in BODY?

Unpleasantness ensues:
- Try to ensure this never happens
- Iterate over a “copy” of the object
  - S[:]

Old school For-loops

There’s a simple way to write good-old for-loops

for i in range(10):
    print i

Built-in function: range

>>> for i in range(10):
    print i
0 1 2 3 4 5 6 7 8 9

But lookout!

def funny_fun(s):
    for x in s:
        print x
        s[len(s):] = [x]

Adds x to end object being iterated over!
- Loops forever

def dup_by_k(s, k):
    for x in s:
        print x
        s = s + x*k
    return s

Creates new object w/ x*k added at end

Iteration object is what s “originally” referred to, which is unchanged

What can sequences do?

Select
- i-th element: s[i]
- subsequence (“slice”): s[i:j]

Update:
- For mutable sequences (e.g. Lists)
  - Update i-th element: s[i] = e
  - Update subsequence: s[i:j] = e

Member: x in S

Iteration:
for x in S: <body>

To make it more readable
What else?

Three useful functions for lists from ML?
- `map`
- `filter`
- `fold` (a.k.a. `reduce`)

Built-in in Python:

```python
# Example

def dup(x):
    return 2*x

>>> z = range(10)
>>> z
[0,1,2,3,4,5,6,7,8,9]
>>> map(dup,z)
[0,2,4,6,8,10,12,14,16,18]
>>> map(dup,"chimichanga")
["cc","hh","ii","mm","ii","cc","hh","aa","nn","gg","aa"]
```

- Works for all sequences, returns a list
- More flexible ways to call it, see documentation

Filter

- Works for all sequences, returns same kind of sequence

```python
>>> def even(x): return int(x)%2==0
>>> filter(even, [0,2,4,6,8])
[0,2,4,6,8]
>>> filter(even,"1234096001234125")
"240600242"
>>> filter(even, (1,2.0,3.2,4))
(2,4)
```

- Again, note the polymorphism that we get from dynamic types and conversion

Reduce

- i.e. fold

```python
>>> def add(x,y): x+y
>>> reduce(add,range(10),0)
45
>>> def fac(x):
    def mul(x,y): return x*y
    return reduce(mul,range(1, x+1),1)
>>> fac(5)
120
```

What can sequences do?

Select
- i-th element: `s[i]`
- subsequence ("slice"): `s[i:j]`

Update
- For mutable sequences (e.g. Lists)
  - Update i-th element: `s[i] = e`
  - Update subsequence: `s[i:j] = e`

Member: `x in s`

Iteration: `for x in s: <body>`

map, filter, reduce

List Comprehensions

A cleaner, nicer way to do map-like operations

```python
>>> [ x*x for x in range(10) ]
[0,1,4,9,16,25,36,49,64,81]
>>> [2*x for x in "yogurt cheese"]
["yy","oo","gg","uu","rr","tt",...]
```
List Comprehensions

Syntax:

```python
>>> [e for x in s]
```

Equivalent to:

```python
>>> def map_fn(x): return e
>>> map(map_fn, s)
```

A cleaner, nicer way to do map+filter-like operations

```python
>>> [x**2 for x in range(10) if even(x)]
[0, 4, 16, 36, 64]
>>> [2*x for x in "0123456" if even(x)]
["00", "22", "44", "66"]
>>> [z[0] for z in craigslist if z[1]<3.0]
["dinosaur"]
```

Can “nest” the for to iterate over multiple sequences

```python
>>> [(x,y) for x in range(3) for y range(3)]
[(0,0), (0,1), (0,2), (1,0), (1,1), (1,2), (2,0), (2,1), (2,2)]
>>> [(x,y) for x in range(3) for y in range(3) if x < y]
[(1,0), (2,0), (2,1)]
```
What can sequences do?

Select
• i-th element: $s[i]$
• subsequence ("slice"): $s[i:j]$

Update -- For mutable sequences (e.g. Lists)
• Update i-th element: $s[i] = e$
• Update subsequence: $s[i:j] = e$

Member: $x$ in $s$

Iteration: for $x$ in $s$: <body>
map, filter, reduce

Comprehensions: $[e_x$ for $x$ in $s$ if $c_x]$  

Quicksort in Python

```python
def sort(L):
    if L==[]: return L
    else:
        l=sort([...])
        r=sort([...])
        return(l+L[0:1]+r)
```  

Today: Revisit some objects

• Exploit features and build powerful expressions

Base: int, float, complex

Sequence: string, tuple, list

Maps (Dictionary): key → value

Key data structure: Dictionaries

Associative arrays, Hash tables ...

A table storing a set of “keys”,
And a “value” for each key.

Any (immutable) object can be a key!
• int, float, string, tuples...

Very useful!

Using Dictionaries

Unsorted list of key,value pairs

Empty Dictionary: {} 

Non-empty Dictionary: {k1:v1,k2:v2,...}

Membership: is $k$ in dict: $k$ in $d$

Lookup value of key: $d[k]$

Set value of key: $d[k]=v$
Dictionaries

```python
>>> d = {}
>>> d = dict(mexmenu)
>>> d["ceviche"] = 3.95
>>> d
{…}
>>> d["burrito"]
3.50
>>> d.keys()
...
>>> d.values()
```

```python
def freq(s):
    d = {}
    for c in s:
        if c in d:
            d[c] += 1
        else:
            d[c] = 1
    return d

>>> d = plotfreq([1,1,3.0,"A",3.0,"A","A",1,2,3.0,1,"A"])
>>> d
...
>>> d = plotfreq("avrakedavra")
>>> d.keys()
...
>>> d
...
```

You now know enough to do PA5
- Python Tutorial: How to open files, read lines
- Use the help command
- Document every function: What does it do?

```python
>>> f = open("foo.txt","read")
>>> f.readlines()
...
>>> for l in f.readlines():
    <BODY>
>>> f.close
```