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)

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)

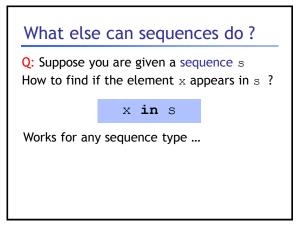
- Update i-th element: s[i] = e
- Update subsequence: s[i:j] = e

Update subsequence: 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

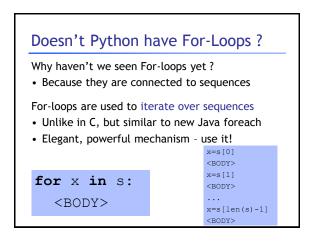
Update subsequence s[i:j]=e >>> z = [1,2,3,4,5,6,7,8,9,10] >>> z[3:6] = [``a", ``b", ``c"] >>> z [1,2,3,"a", "b", "c",7,8,9,10] >>> z[3:6] = [``a", ``b"] * 2 >>> z [1,2,3,"a", "b", "a", "b", 7,8,9,10]

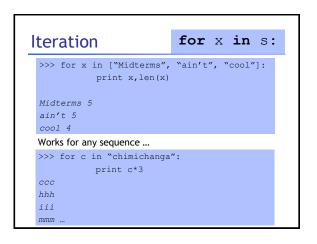
```
[1,2,3,"a","b","a","b",7,8,9,10]
>>> z[4:]=[]
>>> z
[1,2,3,"a"]
>>> z[:0] = ["al", "be"]
>>> z
["al","be",1,2,3,"a","b","a","b",7,8,9,10]
```



| Sequence "contains" x in s |
|----------------------------|
| >>> "a" in "cat" |
| True |
| >>> "a" in "entebbe" |
| False |
| >>> "a" in ("c", "a", "t") |
| True |
| >>> 2 in [1,2,3,4,5] |
| True |
| >>> 2 in [1,4,"92",2.4] |
| False |

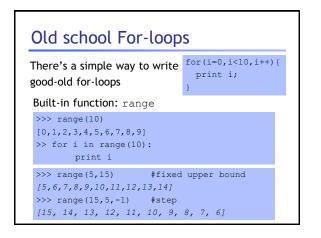
| elect | |
|--------------------------------------|-----------------------------|
| i-th element: s | [i] |
| subsequence ("s | lice"):s[i:j] |
| Jpdate For muta | able sequences (e.g. Lists) |
| Update i-th elem | nent:s[i] = e |
| Update subseque | ence:s[i:j] = e |
| Nember | |
| Is an element in | a sequence: x in s |

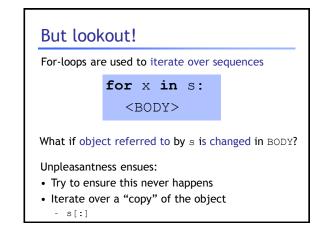


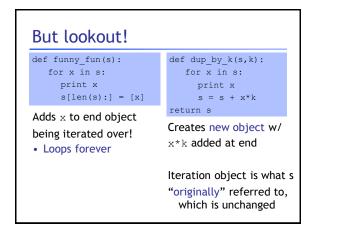


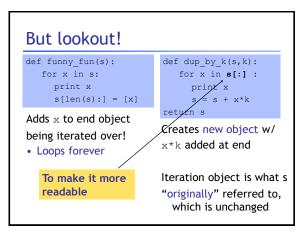
| Iteration | for x in s: |
|--|----------------------------|
| <pre>>>> s=0 >>> z=(1,2,3,4.0,"5") >>> for i in z: s = s + i ERROR >>> s 10</pre> | #tuple |
| Can't add string to float Note that first 4 elts added! Dynamic Types! Run-time Type Error | >>> s=0 >>> for i in z: |

| Itomation , binding |
|--|
| Iteration + binding for x, in s: |
| If s is a sequence of tuples/sequences, then we can |
| Bind to individual elements of "subsequences" |
| <pre>>>>craigslist = [("alien",3.50), ("dinosaur",1.90), ("quiz",100.50), ("quesadilla",3.00), ("good grade in 130","priceless")]</pre> |
| >>>for i,p in craislist: |
| 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 |









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

lteration: for x in s: <body>

What else ?

Three useful functions for lists from ML?

- map
- filter
- fold (a.k.a. reduce)
- Built-in in Python:

| <pre>def dup(x): return 2*x >>> z = range(10) >>> z [0,1,2,3,4,5,6,7,8,9] >>>mp(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]</pre> | ıр | | | |
|--|--|---|------|---|
| <pre>>>> 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","</pre> | ÷ . | | | |
| | >>> z [0,1,2,3, >>> map(c [0,2,4,6, >>>map(du ["cc","hh | 4,5,6,7,8,9] lup,z) 8,10,12,14,1 p,"chimichan t","ii","mm", | ga") | " |

• More flexible ways to call it, see documentation

filter

• Works for all sequences, returns same kind of sequence

```
>>> def even(x): return int(x)%2==0
>>> filter(even,range(10))
[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

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

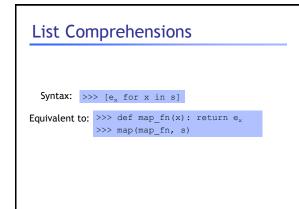
A cleaner, nicer way to do map-like operations

>>> [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: >>> [e_x for x in s]

Equivalent to:

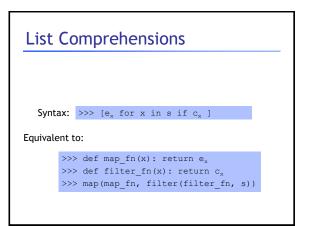


List Comprehensions

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

>>> [x*x 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"]</pre>

| List Co | omprehensions | |
|-----------------------|---|--|
| Syntax: Equivalent | >>> $[e_x \text{ for } x \text{ in } s \text{ if } c_x]$ to: | |
| | | |



List Comprehensions

Can "nest" the for to iterate over multiple sequences

>>>[(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 $\label{eq:comprehensions: [e_x for x in s if c_x]} \mathsf{Comprehensions: [e_x for x in s if c_x]}$

Quicksort in Python

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

Quicksort in Python

```
def sort(L):
    if L==[]: return L
    else:
        l=sort([x for x in L[1:] if x < L[0]])
        r=sort([x for x in L[1:] if x >= L[0]])
        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): $key \rightarrow value$

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: { }

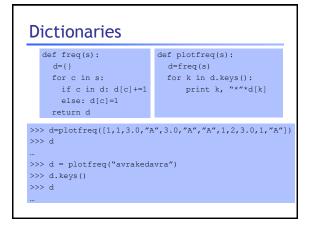
Non-empty Dictionary: {k1:v1,k2:v2,...}

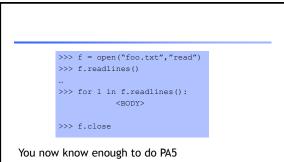
Membership: is k in dict: k in d

Lookup value of key: d[k]

Set value of key: d[k]=v

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





- Python Tutorial: How to open files, read lines
- Use the ${\tt help}$ command
- Document every function: What does it do ?