A crash course in Python

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 < \( j-i \)

What else can sequences do?

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

\[ x \ in \ s \]

Works for any sequence type
Sequence “contains” \( x \text{ in } s \)

```python
>>> "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
```

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/Java/…
- Elegant, powerful mechanism - use it!

```python
for x in s:
  <BODY>
```

Iteration

```python
>>> for x in ["how", "many", "cannonballs"]:
    print x,len(x)
how 3
many 4
cannonballs 11
```

Works for any sequence ...

```python
>>> for c in "chimichanga":
    print c*3
ccc
hhh
iii
mmm ...
```
## Iteration

```python
>>> s=0
>>> z=(1,2,3,4.0,"5")    #tuple
>>> for i in z:
    s  =  s  +  i
ERROR
>>> s
10
```

- Can't add string to float
- Note that first 4 elts added!
- Dynamic Types!
- Run-time Type Error

## Iteration + binding

```python
>>> s=0
>>> z=(1,2,3,4.0,"5")    #tuple
>>> for i in z:
    s = s + float(i)
>>> s
15
```

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

```python
>>> mexmenu = [("burrito",3.50),("taco",1.90),("sopa",2.50),("quesadilla",3.00),
            ("frijole",0.75)]
```

```python
>>> for i,p in menu:
    print "One", i, "costs", p
One burrito costs 3.50
One taco costs 1.90
One sopá costs 2.50
One quesadilla costs 3.00
One frijole costs 0.75
```

## Old school For-loops

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

```python
for(i=0,i<10,i++){
    print i;
}
```

Built-in function: `range`

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

```python
>>> range(5,15)    #fixed upper bound
[5,6,7,8,9,10,11,12,13,14]
>>> range(15,5,-1) #step
[15, 14, 13, 12, 11, 10, 9, 8, 7, 6]
```

## range(len(...)) to get position

```python
def printPos(s):
    for i in range(len(s)):
        print "at position ", i, "we have", s[i]
```

```python
>>> printPos((5,"15",[2,5]))
at position 0 we have 5
at position 1 we have 15
at position 2 we have [2,5]
```

```python
>>> printPos("chimichanga")
at position 0 we have c
at position 1 we have h
at position 2 we have i ...
```

## Polymorphism via Dynamic Typing
But lookout!

For-loops are used to iterate over sequences
for x in s:
  <BODY>
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[:]

Thursday, November 10, 2011

To make it more readable...

def dup1(s,k):
  for x in s:
  print x
  s[len(s):] = x*k
  return s

Adds x*k to end object being iterated over!
• Loops forever

def dup2(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 x “originally” referred to
• Unchanged

def dup3(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 x “originally” referred to
• Unchanged

Thursday, November 10, 2011

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>
What else?

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

Built-in in Python:

```python
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, 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**

- i.e. **fold**

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

- Again, note the polymorphism that we get from dynamic types and conversion
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 \( s \):
map, filter, reduce

List Comprehensions

A cleaner, nicer way to do map-like operations

\[
\begin{align*}
\text{Syntax:} & \quad \{ x^2 \text{ for } x \text{ in range(10)} \} \\
& \quad \{ 2x \text{ for } x \text{ in } \\
& \quad \quad \text{"avra kedavra"} \}
\end{align*}
\]

List Comprehensions

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

\[
\begin{align*}
\text{Syntax:} & \quad \{ x^2 \text{ for } x \text{ in range(10) if even(x)} \} \\
& \quad \{ 2x \text{ for } x \text{ in } \\
& \quad \quad \text{"0123456" if even(x)} \}
\end{align*}
\]

List Comprehensions

Can “nest” the “for” to iterate over multiple sequences

\[
\begin{align*}
\text{Syntax:} & \quad \{ x, y \text{ for } x \text{ in range(3) for } y \text{ in range(3)} \} \\
& \quad \{ (x,0), (0,1), (0,2), (1,0), (1,1), (1,2), (2,0), \\
& \quad \quad (2,1), (2,2) \}
\end{align*}
\]
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: \( x \text{ in } s \)
Iteration: for \( x \text{ in } s \): <body>
map, filter, reduce
Comprehensions: \( [e_x \text{ for } x \text{ in } s \text{ if } c_x] \)

Quicksort in Python

```python
def sort(L):
    if L==[]:
        return L
    else:
        l=sort([x for x in s[1:] if x < s[0])
        r=sort([x for x in s[1:] if x >= s[0])
        return (l+L[0:1]+r)
```

Next: 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

```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
...  
```

Dictionaries

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

You now know enough to do PA6

- Python Tutorial: How to open files, read lines
- Use the help command
- Document every function: What does it do?
Recap: Python Tutorial

• Base Types
• Sequence Types
• Loops
• Functions
• List comprehensions
• map, filter, reduce

... enough to do PA 5