Midterm Solutions

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 in s
```

Works for any sequence type...

**Sequence “contains”**

```
>>> "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 in s`

**Iteration**

```
>>> for x in ("Midterms", "ain't", "cool"):
    print x, len(x)
Midterms 5
ain't 5
cool 4
```

Works for any sequence ...

```
>>> 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
>>> 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
```

Old school For-loops

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

Built-in function: range

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

But lookout!

For-loops are used to iterate over sequences

```python
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[:]

But lookout!

```python
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[len(s):] = [x]
    return s

Creates new object w/ x*k added at end
• Loops forever

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

But lookout!

```python
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
```

To make it more readable

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 \(x \in s\):

What else?

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

Built-in in Python:

**map**

```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**

```python
>>> def even(x):
    return int(x) % 2 == 0

>>> filter(even, range(10))
[0, 2, 4, 6, 8]
>>> filter(even, "1234096001234125")
"24060242"
>>> filter(even, (1, 2.0, 3.2, 4))
(2, 4)
```

- Works for all sequences, returns same kind of sequence
- Again, note the polymorphism that we get from dynamic types and conversion

**reduce**

```python
>>> def add(x, y):
    return x + y

>>> reduce(add, range(10), 0)
45
>>> def mul(x, y):
    return x * y

>>> reduce(mul, range(1, x+1), 1)
120
```

- i.e. fold

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 \(x \in s\):
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 x for x in s]
```

Equivalent to:

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

List Comprehensions

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

```python
>>> [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"]
```

List Comprehensions

Syntax:

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

Equivalent to:

```python
>>> def map_fn(x): return e x
>>> def filter_fn(x): return c x
>>> map(map_fn, filter(filter_fn, s))
```
List Comprehensions

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, for x in s if c,]

QuickSort in Python

```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

<table>
<thead>
<tr>
<th>Base:</th>
<th>int, float, complex</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence:</td>
<td>string, tuple, list</td>
</tr>
<tr>
<td>Maps (Dictionary):</td>
<td>key → value</td>
</tr>
</tbody>
</table>

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()
```

Dictionaries

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

```python
>>> d = plotfreq([1,1,3.0,"A","A","A",1,2,3.0,1,"A"])
>>> d
{…}
>>> d = plotfreq("avrakedavra")
>>> d.keys()
...
>>> d.values()
```

```python
def plotfreq(s):
    d = freq(s)
    for k in d.keys():
        print k, "*"*d[k]
```

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?