Python Open Labs

Lists

Acknowledgement: Kunal Baweja

Files: one thing left

- with keyword
- with - ensures that clean up code is called right after with block
- Advantage - open files using with, no need to explicitly close files

```
with open(“file.txt”, ‘r’) as handle:
    print handle.read()
print “end of with block”
```

List: A linear collection

- Put many values in a single variable
- user = [‘abc’, ‘xyz’, ‘lmn’]
- Formally: a collection of python objects

List: A linear collection

- Put many values in a single variable
- user = [‘abc’, ‘xyz’, ‘lmn’]
- Formally: a collection of python objects

- objcts = [‘abc’, ‘xyz’, 123, [‘abcd’, 123]] (notice more than one type)

Lists: Iterable

- Lists and definite loops are best friends

```
users = [“abc123”, “xyz456”, “asd234”]

for user in users:

    print user + “@columbia.edu”
```

Indexing Lists

- 0 indexed
- 0, 1, 2, 3 ------ n-1 for a list of length n

```
users = [“abc123”, “xyz456”, “asd234”]

print users[0], users[1]
```

- Length of a list of unkown length ?

range: generate numbers

- Python built-in function
- Generates a list of numbers in a given range
- 1 to 3 arguments
- range(n) - 0 to n-1 integers in a list
- range(a, b) - optional two arguments
    - [a, b) range of integers
- range(a, b, c) - [a, b) integers
    - step size of c

range: generate numbers

```
print range(10)

print range(3, 7)

print range(0, 10, 3)
```

- Does it work on negative numbers ?
- 10 billion numbers ?

xrange: generator

- Exactly same semantics and external working as range()
- Internal working differs
- It's a generator
- "Generates" numbers one at a time

- How to access generated numbers ?
- Iterate and access one at a time

```
a = range(10)

b = xrange(10)

print a, type(a)

print b, type(b)
```

Lists: Mutable

- Mutable = modifiable
- Insert / Delete / Modify list elements

```
nums = [1, 2, 3]

for i in range(0, len(nums)):

    nums[i] = nums[i] ** 2
```

```
nums = [1, 2, 3]

for num in nums:

    num = num ** 2
```

Which snippet is correct ? Why ?

Concatenate lists

- + operator

```
l1 = [1, 2, 3]

l2 = [4, 5, 6]

l3 = l1 + l2

for i in l3:

    print i
```

Lists can be sliced as well

- Do you remember string slicing ?

```
l = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

print l[:4]

print l[4:10]

print l[6:]
```

- Try manually before running in python !!

Sort a list

- List is an ordered set of objects
- Ordering is maintained unless changed
- Usually ordering is same as order of insertion

```
l = [3, 2, 1]

l.sort()

print l
```

Building a list, from scratch

- Declare an empty list
- append elements to the list

```
l = [] #one way

l.append(5)

l.append(4)

l.append(2)

print len(l)
```

```
l = list() #another way

l.append(3)

l.append(7)

l.append(2)

print len(l)
```

List: As a stack of objects

- append elements to the list
- pop from the end of list
- Stacks are used in graph/ tree searches

```
l = []

l.append(5)

l.append(4)

print l.pop()
```