Python/programming boot camp

02-713

Quick Poll ...

1. Unix+ssh+scp + editing files
2. C, Java, Perl, Ruby?
3. Comfort w/ Python
4. Python setup?
5. Laptop?

Goals

- More in-depth intro to Python (from Prof. Kingsford's slides)
- Overview how to set up Python and required libraries for HW5
- Individual help with getting things installed
- Experiment with heap implementation
- Q & A

Slides & links available on Autolab

From Prof Kingsford's slides

Python Basic Structure

Python is an interpreted language (like Perl).

Programs are in files with the .py extension.

Programs are executed from top to bottom.

Advanced: it's strongly dynamically typed (values have a fixed type, but variables can change type on the fly.)

Most unusual syntax: indenting and newlines are important.

Unlike Perl, there are no { } characters to indicate the start and end of a block. That is done through indenting.

From Prof Kingsford's slides

Example

A function that takes 1 parameter

"Docstring" that documents what the function does.

```
def random_order(n):
  "Create random mapping between [n] and [n]"

  import random
```

Load the "random" library.

```
  R = range(n)
```

R = [0, 1, 2, 3, ..., n-1]

```
  random.shuffle(R)
```

The list R is randomly shuffled to be something like [7, 8, 10, n-1, ..., 4]

```
  return dict(enumerate(R))
```

Turns list of pairs [(i,j)] into a mapping from i → j

Turns shuffled list into a list of pairs:
[(0, 7), (1, 8), (2, 10), ...]

From Prof Kingsford's slides

Built-in Python Data Types

Main Idea: Sequences

From Prof Kingsford's slides

Built-in Basic Data Types

str = string (delimit with 'xyz' or "xyz")

```
>>> str(10)
'10'
```

int = arbitrary-sized integer (see also long)

```
>>> 7**73
492217353521848729599618551903381776068465426225614008572
62407L
```

float = floating point number

```
>>> 1/2
0
>>> 1.0/2
0.5
```

bool = True or False

```
>>> bool(10)
True
>>> bool(0)
False
```

From Prof Kingsford's slides

Collection Data Types

list = mutable list

```
>>> ['a','b',10,10,7]
['a', 'b', 10, 10, 7]
```

tuple = frozen list (can't change)

```
>>> ('a','b',10, 10,7)
('a', 'b', 10, 10, 7)
```

dict = dictionary, aka hash

```
>>> {'a':7, 'b':10, 13:2}
{'a': 7, 'b': 10, 13: 2}
```

set = mutable set of elements

```
>>> set(['a','b','b',10])
set(['a', 10, 'b'])
```

frozenset = frozen set of elements

```
>>> frozenset(['a','b','b',10])
frozenset(['a', 10, 'b'])
```

From Prof Kingsford's slides

Collections

Can contain items of different type.

Can nest them: [(1, 2), (3, 4), [5, 6, 7, 8], {'a': 2}]

Sets do not preserve order.

Dictionary keys must be constant, but can be frozenset or tuples:

```
>>> A = {}
>>> A[(1,2)] = 10
>>> A[frozenset([2,2,2,2])] = 13
>>> A
{(1, 2): 10, frozenset([2]): 13}
>>> A[ [10,2] ] = 3
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'
```

From Prof Kingsford's slides

Slicing Lists and Strings

Can extract subranges from lists and strings:

```
s = "abcdef"
s[0]  == "a"
s[2:4] == "cd"
s[2:] == "cdef"
s[-1] == "f"
```

negative numbers count from the end.

L = [1,2,3,4,5]
L[3:7] == [4,5]
L[:2] == [1,2]
T = (7,8,9,10)
T[1:3] == (8,9)

Note: range i:j gives characters i, i+1,..., j-1.

For range i:j

if i is omitted, it's assumed to be 0.
if j is omitted, it's assumed to be len + 1.

Assignment works for lists (but not strings or tuples):

```
L[2:4] =  [7,8,9,10] → [1, 2, 7, 8, 9, 10, 5]
```

From Prof Kingsford's slides

For Loops

For loops always loop over a <u>sequence</u>.

Collections are sequences.

```
for x in [1,2,3,4]:
    print x
```
Prints 1 2 3 4

```
for key in {'a':10,'b':100}:
    print key
```
Prints a b OR b a

```
for i in set([1,2,3,2]):
    print i
```
Prints 1 2 3 in some order

Generate sequences:

```
range(100) = [0,1,2,...,99]
range(10,50) = [10,11,...,49]
range(10,20,2) = [10, 12, 14, 16, 18]
```

```
for i in range(32):
    print 2**i
```

From Prof Kingsford's slides

List Comprehensions

Can construct lists from rules:

```
L = [i**2 + j**2
        for i in range(10) for j in range(10)
            if i >= j]
```

```
>>> L
[1, 4, 5, 9, 10, 13, 16, 17, 20, 25, 25, 26, 29, 34, 41, 36, 37, 40, 45,
52, 61, 49, 50, 53, 58, 65, 74, 85, 64, 65, 68, 73, 80, 89, 100, 113, 81,
82, 85, 90, 97, 106, 117, 130, 145]
>>> set(L)
set([1, 130, 4, 5, 9, 10, 13, 16, 17, 20, 25, 26, 29, 34, 36, 37, 40, 41,
45, 49, 50, 52, 53, 58, 61, 64, 65, 68, 73, 74, 80, 81, 82, 85, 89, 90,
97, 100, 145, 106, 113, 117])
```

General syntax: [EXPR for ... if ... for ... if]

```
L = []
for i in range(10):
    for j in range(10):
       if i >= j:
          L.append(i**2 + j**2)
```

From Prof Kingsford's slides

Generators

Often it is wasteful to create a list in memory:

```
for i in range(2**20):
    print i
```

First creates a list of ≈ 1 million items, then iterates through it.

```
for i in xrange(2**20):
    print i
```

Creates a <u>generator</u> for the list and iterates through it.

Generators are rules that generate a sequence:

```
(i**2 + j**2 for i in range(10)
    for j in range(10)
       if i >= j)
```

Generator has same syntax as list comprehension, but will only create an item as you iterate through it.

The only thing you can do with generators is iterate through them.

From Prof Kingsford's slides

Composing Generators

Generators and other sequences can be passed to functions that create new generators:

```
s = "abcd"
for c in reversed(s): s → ('d', 'c', 'b', 'a')
    print c

L = ["a", "b", "c", "d"]
for (i, c) in enumerate(L): L → ((0, "a"), (1, "b"), (2, "c"), (3, "d"))
    print i, c
```

9.7. `itertools` — Functions creating iterators for efficient looping

product('ABCD', repeat=2)		AA AB AC AD BA BB BC BD CA CB CC CD DA DB DC DD
permutations('ABCD', 2)		AB AC AD BA BC BD CA CB CD DA DB DC
combinations('ABCD', 2)		AB AC AD BC BD CD
combinations_with_replacement('ABCD', 2)		AA AB AC AD BB BC BD CC CD DD

From Prof Kingsford's slides

Organizing Code

From Prof Kingsford's slides

Functions

Functions can be defined using the syntax:

```
def name(a, b, c=True, d=2*10):
    BODY
```

The syntax "= EXPR" after a parameter gives the parameter's default value.

Functions can be called using:

```
name(10,20, False)
name(10, b=20, d=32)
name(b=10, a=20)
```

Values can be returned from functions using the <u>return</u> statement:

```
def sum(S):
    s = 0.0
    for i in S: s = s + i
    return s
```

From Prof Kingsford's slides

Comments

Comments start with # and go until the end of the line:

```
# this is a comment
```

Strings can be placed as comments as first statement in a file or a function:

```
def bandwidth(M):
    "Compute the Bandwidth of M"
    return max(abs(i-j) for i in xrange(len(M))
      for j in xrange(i,len(M)) if M[i,j] != 0)
```

Strings surrounded by “”” xxx”””” or ‘’‘xxx’’’ can span multiple lines.

From Prof Kingsford's slides

Packages

Code can be imported from other files and standard packages using <u>import</u>:

```
import NAME
from NAME import id1, id2, id3 ...
from NAME import *
```

For example:

```
import math
print math.log(10)
from math import log
print log(10)
```

<u>import</u> will search your current directory, the standard python directories, and directories in your PYTHONPATH environment variable.

From Prof Kingsford's slides

Classes

A class represents a user defined type.

Classes can have functions and variables associated with them.

Classes are *instantiated into objects.*

```
class Species:
    def __init__(self, name):
        self.name = name

    def species_name(self):
        return self.name


Ce = Species("C. elegans")
Hs = Species("H. sapiens")

print Ce.name, Hs.name
print Ce.species_name(), Hs.species_name()
```

Special function called _ _init_ _ is the constructor that says how to build an instance of the class.

All functions in a class take a "self" parameter that represents the object.

New instance of Species created with name = "C. elegans"

From Prof Kingsford’s slides

Other Statements

From Prof Kingsford's slides

Reading Files

"with" statement sets up a context. The main use is to open an file and ensure, no matter what happens, the file will be closed.

```
with open(filename) as inp:
    for line in inp:
        line = line.strip()
        s = line.split()
        …
```

Input file is a sequence of lines & we can iterate over the lines using a for loop

the strip() function removes whitespace from the start and end of the string

split() converts the string into a list of words

From Prof Kingsford's slides

Print

```
print expr1, expr2, ..., exprK
```

will output the result of converting the given expressions into strings.

Expressions will be separated by a space, and a newline will be printed at the end.

```
>>> print 10, 20, “cat”, 2*100-5
10 20 cat 195
```

End with a comma to omit the newline at the end and to smartly separate items with spaces:

```
>>> for a in (1,2,3,4): print “item=”, a,
item= 1 item= 2 item= 3 item= 4
```

Output to a file with the (strange) syntax:

```
print >>F, expr1, expr2, ..., exprK
```

where F is an open file object.

From Prof Kingsford's slides

Math Operators

x + y; x - y; x * y: addition, subtraction, and multiplication

x / y : type-preserving division (if x and y are both integers, the result will be an integer)

x // y : integer division (floor(float(x)/y))

x % y : remainder of x / y

x**y : x raised to the yth power

abs(x) : absolute value of x

round(x) : round x to nearest integer

sum(SEQ) : sum of items in the sequence

max(SEQ) : largest item in the sequence

min(SEQ) : smallest item in the sequence

floor, ceil, log, exp, sin, cos, sqrt, factorial, and others available in the built-in "math" package.

From Prof Kingsford's slides

Boolean Expressions

Comparison operators are: == < > <= >= != <u>in</u> <u>is</u>

```
>>> 1 == 2
False
>>> 1 > 2
False
>>> 1 <= 2
True
>>> 1 != 2
True
>>> "a" in “aeiou”
True
>>> 7 in [7,8,9]
True
```

```
>>> a = [1,2,3]
>>> b = [1,2,3]
>>> a == b
True
>>> a is b
False
>>> 4 not in b
True
>>> i = 10
>>> 0 < i < 100
True
```

Boolean operators are: <u>and</u> <u>or</u> <u>not</u>

“a” in “aeiou” and “z” not in “aeiou”

1 < i < 128 and i*j == 100

From Prof Kingsford's slides

If Statements

```
if 2 in xrange(-3,10,2):
    print "YES"
```

Syntax: if EXPR:

```
if "abc" in "abcde":
    print "YES"
else:
    print "NO"
```

"else" block executed if the if-EXPR is False.

```
if s == "Whitman":
    print "Leaves of Grass"
elif s == "Poe":
    print "The Raven"
elif s == "Hawthorne"
    print "The House of Seven Gables"
else:
    print "Author unknown"
```

"elif" blocks are tested in order if the first if is False and the first elif block that is True is run.

From Prof Kingsford's slides

While Loops

```
while EXPR:
    BLOCK
```

will repeatedly execute BLOCK until EXPR is False.

<u>continue</u>: jump to the next iteration of the while or for loop.

<u>break</u>: exit out of the while or for loop.

From Prof Kingsford's slides

Other Language Features

Lambda functions:

Can define a function without a name that contains a single expression:

```
f = lambda a,b: a**2 + b
f(10,2)
L.sort(cmp=lambda a,b: cmp(a[0],b[0]))
```

The *Easy* Way

```
$ ssh <andrewid>@linux.andrew.cmu.edu
$ ~gduggal/epd_free-7.3-2-rh5-x86_64/bin/ipython
> import networkx
```

You can just use Geet's installation if you stick with using the CMU `linux.andrew` servers

Installing Python

Variety of options for all major platforms

For example, on a Mac:

- Compile from source
- Binaries from Python's web site
- Macports (like apt-get on Ubuntu)
- Our focus, **Enthought**: includes Numpy, Matplotlib (+).

Everyone should have access to a Unix setup

Unix Installation

Get onto a server

```
$ ssh gduggal@linux.andrew.cmu.edu
```

Get the Enthought distribution

```
$ wget http://epd-free.enthought.com/epd_free-7.3-2-rh5-x86_64.sh
```

Set up local Enthought install

```
[gduggal@unix12 ~]$ chmod u+x epd_free-7.3-2-rh5-x86_64.sh
[gduggal@unix12 ~]$ ./epd_free-7.3-2-rh5-x86_64.sh

Welcome to the EPD_free-7.3-2 installer!

To continue the installation, you must review and approve the license term
agreement.
Press Enter to continue
```

```
EPD_free will be installed to this location:
/afs/andrew.cmu.edu/usr19/gduggal/epd_free-7.3-2-rh5-x86_64
```

Unix Installation

Set up .bashrc (optional)

```
As the last step, you should edit your .bashrc or prepend
the EPD_free install path:

    /afs/andrew.cmu.edu/usr19/gduggal/epd_free-7.3-2-rh5-x86_64/bin

Thank you for installing EPD_free!
```

Test ipython interactive shell

```
[gduggal@unix12 ~]$ ./epd_free-7.3-2-rh5-x86_64/bin/ipython
Enthought Python Distribution (free version) -- www.enthought.com
(type 'upgrade' or see www.enthought.com/epd/upgrade to get the full EPD)

Python 2.7.3 |EPD_free 7.3-2 (64-bit)| (default, Apr 11 2012, 17:52:16)
Type "copyright", "credits" or "license" for more information.

IPython 0.12.1 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]:
```

CTRL+D to exit

Unix Installation

Set up and test networkx

```
[gduggal@unix12 ~]$ wget https://pypi.python.org/packages/source/n/networkx/networkx-1.7.tar.gz#md5=1a73da9d571a206aa40f6ef69254f7b4
--2013-03-05 22:50:54--  https://pypi.python.org/packages/source/n/networkx/networkx-1.7.tar.gz
Resolving pypi.python.org... 140.211.10.69
Connecting to pypi.python.org|140.211.10.69|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 728145 (711K) [application/octet-stream]
Saving to: “networkx-1.7.tar.gz”

100%[=====================================================================================>] 728,145     48.2K/s   in 13s

2013-03-05 22:51:08 (54.1 KB/s) - “networkx-1.7.tar.gz” saved [728145/728145]

[gduggal@unix12 ~]$ tar -xvzf networkx-1.7.tar.gz
```

```
[gduggal@unix12 ~/networkx-1.7]$ ../epd_free-7.3-2-rh5-x86_64/bin/python setup.py install
```

```
[gduggal@unix12 ~]$ epd_free-7.3-2-rh5-x86_64/bin/ipython
Enthought Python Distribution (free version) -- www.enthought.com
(type 'upgrade' or see www.enthought.com/epd/upgrade to get the full EPD)

Python 2.7.3 |EPD_free 7.3-2 (64-bit)| (default, Apr 11 2012, 17:52:16)
Type "copyright", "credits" or "license" for more information.

IPython 0.12.1 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.
```

```
In [1]: import networkx

In [2]: G = networkx.Graph()

In [3]: G.add_edge(1,2)

In [4]: G
Out[4]: <networkx.classes.graph.Graph at 0x1a22610>

In [5]: G.edges()
Out[5]: [(1, 2)]

In [6]:
```

From Prof Kingsford's slides

Python Code for a *d*-Heap (Array-based implementation)

ARITY=*d*

```python
class HeapItem(object):
    """Represents an item in the heap"""
    def __init__(self, key, item):
        self.key = key
        self.item = item
        self.pos = None

def makeheap(S):
    """Create a heap from set S, which should
    be a list of pairs (key, item)."""
    heap = list(HeapItem(k,i) for k,i in S)
    for pos in xrange(len(heap)-1, -1, -1):
        siftdown(heap[pos], pos, heap)
    return heap

def findmin(heap):
    """Return element with smallest key,
    or None if heap is empty"""
    return heap[0] if len(heap) > 0 else None

def deletemin(heap):
    """Delete the smallest item"""
    if len(heap) == 0: return None
    i = heap[0]
    last = heap[-1]
    del heap[-1]
    if len(heap) > 0:
        siftdown(last, 0, heap)
    return i

def heapinsert(key, item, heap):
    """Insert an item into the heap"""
    heap.append(None)
    hi = HeapItem(key,item)
    siftup(hi, len(heap)-1, heap)
    return hi
```

```python
def siftup(hi, pos, heap):
    """Move hi up in heap until it's parent is
    smaller than hi.key"""
    p = parent(pos)
    while p is not None and heap[p].key > hi.key:
        heap[pos] = heap[p]
        heap[pos].pos = pos
        pos = p
        p = parent(p)
    heap[pos] = hi
    hi.pos = pos

def siftdown(hi, pos, heap):
    """Move hi down in heap until its smallest
    child is bigger than hi's key"""
    c = minchild(pos, heap)
    while c != None and heap[c].key < hi.key:
        heap[pos] = heap[c]
        heap[pos].pos = pos
        pos = c
        c = minchild(c, heap)
    heap[pos] = hi
    hi.pos = pos
```

```python
def parent(pos):
    """Return the position of the parent of pos"""
    if pos == 0: return None
    return int(math.ceil(pos / ARITY) - 1)

def children(pos, heap):
    """Return a list of children of pos"""
    return xrange(ARITY * pos + 1, min(ARITY * (pos + 1) + 1, len(heap)))

def minchild(pos, heap):
    """Return the child of pos with the smallest key"""
    minpos = minkey = None
    for c in children(pos, heap):
        if minkey == None or heap[c].key < minkey:
            minkey, minpos = heap[c].key, c
    return minpos
```

```
[gduggal@unix12 ~]$ epd_free-7.3-2-rh5-x86_64/bin/ipython
Enthought Python Distribution (free version) -- www.enthought.com
(type 'upgrade' or see www.enthought.com/epd/upgrade to get the full EPD)

Python 2.7.3 |EPD_free 7.3-2 (64-bit)| (default, Apr 11 2012, 17:52:16)
Type "copyright", "credits" or "license" for more information.

IPython 0.12.1 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: import heap

In [2]: H = heap.makeheap([(0,'a'),(1,'b'),(2,'c'),(3,'d')])

In [3]: for i in H: print i.key, i.item
0 a
1 b
2 c
3 d

In [4]:
```

Add all defs into heap.py and run ipython

Optional: modularize heap into class w/ default print ops