Introduction

Fall Semester

Topic Outline

- Python
- Graphs
  - Neighbor list
  - Hash tables
  - Recursion VS heapq
- Optimization
  - Search
  - Heuristics
  - Localized
- Scaling
  - Speed
  - Memory
  - List algorithms
- Vision
- Simulation
- Generic code
- Learning

www.python.org

"Python is a dynamic object-oriented programming language that can be used for many kinds of software development. It offers strong support for integration with other languages and tools, comes with extensive standard libraries, and can be learned in a few days. Many Python programmers report substantial productivity gains and feel the language encourages the development of higher quality, more maintainable code."

www.norvig.com

Test	Lisp	Java	Python	Perl	C++	
exception handling	0.01	0.90	1.54	1.73	1.00	
hash access	1.06	3.23	4.01	1.85	1.00	
sum numbers from file	7.54	2.63	8.34	2.49	1.00	100+ x C++
reverse lines	1.61	1.22	1.38	1.25	1.00	50-100 x C++
matrix multiplication	3.30	8.90	278.00	226.00	1.00	10-50 x C++
heapsort	1.67	7.00	84.42	75.67	1.00	5-10 x C++
array access	1.75	6.83	141.08	127.25	1.00	1-5 x C++
list processing	0.93	20.47	20.33	11.27	1.00	0-1 x C++
object instantiation	1.32	2.39	49.11	89.21	1.00	
word count	0.73	4.61	2.57	1.64	1.00	
25% to 75%	0.93 to 1.67	2.63 to 7.00	2.57 to 84.42	1.73 to 89.21	1.00 to 1.00	

Relative speeds of 5 languages on 10 benchmarks from The Great Computer Language Shootout. Speeds are normalized so the g++ compiler for C++ is 1.00, so 2.00 means twice as slow; 0.01 means 100 times faster. Background colors are coded according to legend on right. The last line estimates the 25% to 75% quartiles by throwing out the bottom two and top two scores for each language.

So why Python?

- Easy, fun, quick to learn.
- Good for prototyping algorithms.
- Helps get from one algorithm to a better algorithm.
- Development time reduced so in an education environment more complicated problems can be covered in a given time frame.
- Execution time longer so in a production environment port your best algorithm to C or something faster for the real run.
- Exception, if compute time is not the bottleneck (e.g., application runs across Internet and bandwidth is real bottleneck).

Dynamic Variable Typing

```
a=[1,2,3]                    # comments
b='cat'
c=10
print a,b,c                  # [1,2,3] cat 10

a[1]='dog'                   # heterogeneous list
c=3.14
print a,b,c                  # [1,'dog',3] cat 3.14
```

Blocking and Indentation (1)

```
a=[1,2,3]
for elem in a:
    print elem**3,
    if elem%2==0:
        print 'EVEN'
    else:
        print 'ODD'
    print '*'*20
```

Output:

```
1 ODD
********************
8 EVEN
********************
27 ODD
********************
```

Blocking and Indentation (2)

```
a=[1,2,3]
k=0
while k<len(a):
    print 'k=%d'%k
    if a[k]%2==0:
        print 'Ha!'
    k+=1
print '*'*20
```

Output:

```
k=0
k=1
Ha!
k=2
********************
```

List Slices

```
a=range(10)                    [0,1,2,3,4,5,6,7,8,9]
b=range(5,10)                  [5,6,7,8,9]
c=range(10,100,30)             [10,40,70]
print a[0]                     # 0
print a[-1]                    # 9
print a[4:8]                   # [4,5,6,7]
print b[1:]+c[:-1]             # [6,7,8,9,10,40]
```

Pointers Passed by Value

```
def f(a):
    a=[1,2,3]

a=[‘a’,‘b’,‘c’]
print a
f(a)
print a # SAME
```

```
def g(a):
    a[:]=[1,2,3]

a=[‘a’,‘b’,‘c’]
print a
g(a)
print a # DIFFERENT
```

More Pointers (1)

```
def f(a):
    b=a
    b[0],b[1]=b[1],b[0]

a=[1,2,3]
f(a)
print a # [2,1,3]
```

```
def g(a):
    b=[2,1,3]
    a=b

a=[1,2,3]
g(a)
print a # [1,2,3]
```

More Pointers (2)

```
a=[1,2,3]
b=a[:]
b[0]=7
print a # [1,2,3]
print b # [7,2,3]
b=a
b[0]=7
print a # [7,2,3]
```

```
from copy import *
a=[[1,2,3],‘hi’]
b=deepcopy(a)
b[0][0]=7
print a # [[1,2,3],‘hi’]
b=a[:]
b[0][0]=7
print a # [[7,2,3],‘hi’]
```

Variable Scope (1)

```
def f():
    x=5

x=7
print x # 7
f()
print x # 7
```

```
def g():
    global x
    x=5
x=7
print x # 7
g()
print x # 5
```

Variable Scope (2)

```
def f():
   if x%2==0:
      x=-1
   elif x==1:
      pass
   else:
      x=0
```

```
from sys import exit
if __name__ == \
          '__main__':
   x=7
   f()
   exit(0)
ERROR, line 2, in f
x referenced b/f assigned
```

Return Values (1)

```
from random import *
def f(a):                          # default return value
    shuffle(a)

a=range(8)
a=f(a)                             # like a "void" method
print a # None
```

Return Values (2)

```
def f(a):
    shuffle(a)
    return a
b=f(a)                          # a, b both shuffled

def f(a):
    x=a[:]
    shuffle(x)
    return x
b=f(a)                          # a original, b shuffled
```

Return Values (3)

```
def f():
   return 2,'*'                    # implicit tuple

x,y=f()                            # implicit unpack
a=f()
x,y=a[0],a[1]
x,y=a                              # implicit unpack
```

Classes

```
class Point:
   def __init__(self,x,y): # explicit argument
      self.x=x
      self.y=y

p=Point(3,4)
p.y=7                      # calls setattr
print p.x                  # calls getattr
```

Example Programs

- Spell Checker
  - file input, split, string, list
  - user input, booleans, while-loop
  - break statement, print formatting
- Letter Frequency
  - string module, built-in hash table
  - iterator for-loop on list or string
  - use of “in” and “not” commands

Lab Assignment: Neighbors

- Input as in spell checker:
  - A dictionary of six-letter words.
  - A string from the user.
- String B is a neighbor of String A if exactly one letter is different.
- If the user's input is not a word output NO SUCH WORD.
- If the user's input has no neighbors output NO MATCH.
- Otherwise output the list of neighbors.
- Repeat until they want to quit.
- How efficient is your search for neighbors?