Introduction

September 2011

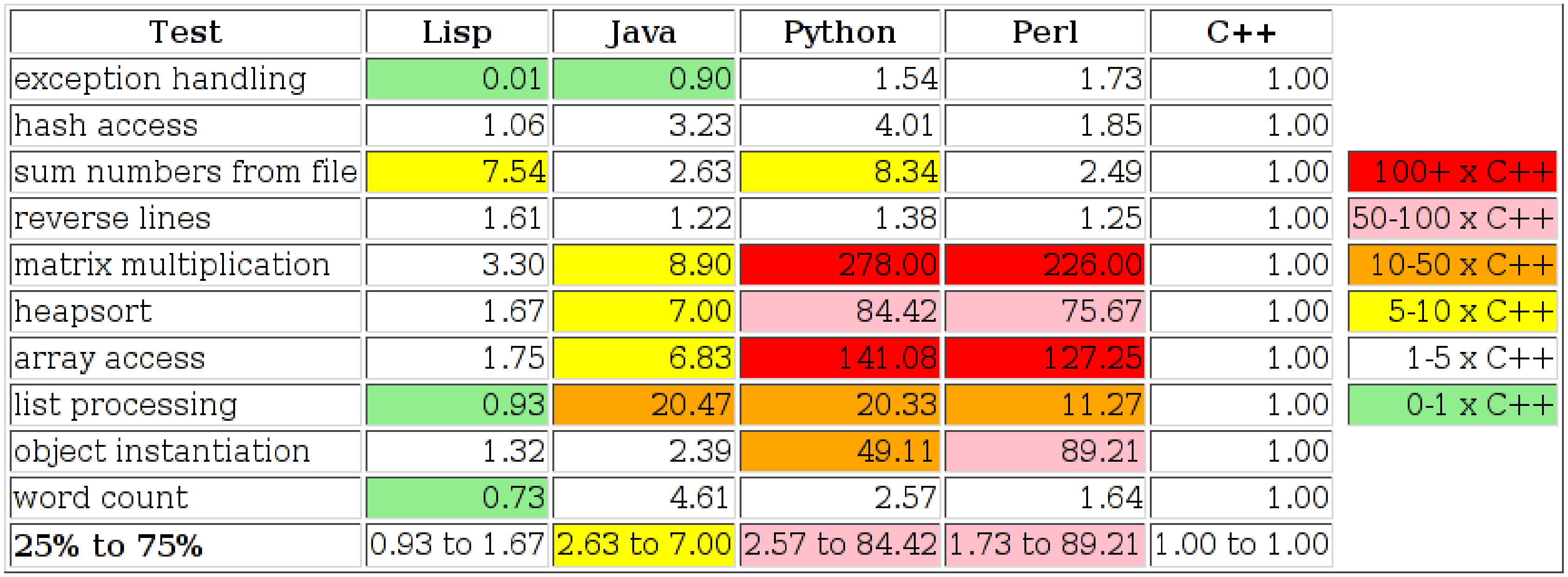
Topic Outline

- Python
- Writing Code
  - Variables
  - Loops and ifs
  - Printing
- Scaling
  - Sample runs
  - Count total steps
  - Average number?

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 a learning 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

```
n=5
m=2*n+1
j=n+1 # halfway point
print n,m,j

j+=1
print j
j-=1
print j
```

Blocking and Indentation (1)

```
from random import random

while 1<=j<=m:
   #
   if random()<0.5:
      j+=1
   else:
      j-=1
   #
```

Blocking and Indentation (2)

```
k=1
while k<=m:
   if k==j:
      print ‘X’,
   elif k==n+1:
      print ‘|’,
   else:
      print ‘-’,
   k+=1
print
```

Sample Output (1)

```
- - - - - X - - - - -
- - - - - | X - - - -
- - - - - | - X - - -
- - - - - | - - X - -
- - - - - | - - - X -
- - - - - | - - - - X
- - - - - | - - - X -
- - - - - | - - - - X
- - - - - | - - - - -
```

Sample Output (2)

```
- - - - - X - - - - -
- - - - - | X - - - -
- - - - - X - - - - -
- - - - X | - - - - -
- - - - - X - - - - -
- - - - - | X - - - -
- - - - - X - - - - -
- - - - X | - - - - -
- - - X - | - - - - -
```

Sample Output (3)

```
- - - - - X - - - - -
- - - - X | - - - - -
- - - X - | - - - - -
- - - - X | - - - - -
- - - X - | - - - - -
- - X - - | - - - - -
- X - - - | - - - - -
- - X - - | - - - - -
- X - - - | - - - - -
```

Lab Assignment: Random Walk

- Initialize $n = 1$, $m = 2n + 1$, and $j = n + 1$ as shown.
- Loop until $j = 0$ or $j = m + 1$ as shown. At each step:
  - Move j left or right at random.
  - Display the current state of the walk, as shown.
- Make another variable to count the total number of steps.
  - Initialize this variable to zero before the loop.
  - However j moves always increase the step count.
- After the loop ends, report the total steps.
  - How does the average number of steps scale with n?