Welcome!

- **About me**
  - Max Peysakhov, Adjunct Faculty, CS
  - Email: mpeysakhov@gmail.com
  - Office: N/A
  - Office hours: Wed 5-6PM, or email for appt.

- **About this course**
  - Syllabus, timeline, & resources on-line...

http://edge.cs.drexel.edu/people/peysakhov/Classes/CS260/
Who / what / where / when / why / how

- Prerequisites
- Lectures
- Readings
- Assignments
- Exams
- Grading
- Communication
- Policies
- Questions?
Course Policies

• Late policy: NO late homeworks, No excuses. (Some considerations may be given for a valid medical reason)
• NO Cell phones! Cell phone ring = extra homework
• NO cheating. You caught you get “F”. No mercy( http://drexel.edu/cs/academics/undergrad/policies/academic-integrity/ ).
• No extra homeworks, projects or tests to “improve the grade” will be given after the final.
• Makeup exams.
  • Advanced notice only.
  • Medical reasons only.
  • Midterm/Final grade will be assigned to final/midterm test.
Ways to Fail this Class

• Fail to hand in more than 50% of the homeworks. (regardless of the exam grades)

• Receive a failing grade for both exams. (regardless of the homework grades)

• Falsify any results.

• Misrepresent another's work as your own (i.e., plagiarism).
Questions?

• If you do not understand ask questions.
• Otherwise I will continue the lecture.
• And we end up like this:
Someone told me your presentation was confusing and unpersuasive.

Sometimes one person's inability to understand looks like another person's inability to explain.

I don't understand what you just said. See?
CSC 9010: Natural Language Processing
Python Intro

Based on presentation from www.cis.upenn.edu/~cse391/cse391_2004/PythonIntro1.ppt
What is Python?

- A programming language with strong similarities to PERL, but with powerful typing and object oriented features.
  - Commonly used for producing HTML content on websites. Great for text files.
  - Useful built-in types (lists, dictionaries).
  - Clean syntax, powerful extensions.
Why Python?

• Natural Language ToolKit
• Ease of use; interpreter
• AI Processing: Symbolic
  – Python’s built-in datatypes for strings, lists, and more.
  – Java or C++ require the use of special classes for this.
• AI Processing: Statistical
  – Python has strong numeric processing capabilities: matrix operations, etc.
  – Suitable for probability and machine learning code.
Learning Python

• Unfortunately, we won’t have time to cover all of Python in class; so, we’re just going to go over some highlights.
  – You’ll need to learn more on your own.
  – Homework 0 asks you to install Python or get comfortable using it here in the lab. It also asks you to read some online Python tutorials.
  – Later homeworks will include Python programming exercises to help you practice.
  – We will fairly rapidly move to using the Natural Language ToolKit.

Based on presentation from www.cis.upenn.edu/~cse391/cse391_2004/PythonIntro1.ppt
Python Tutorials

Things to read through

• “Dive into Python” (Chapters 2 to 4)
  http://diveintopython.org/

• Python 101 – Beginning Python

Things to refer to

• The Official Python Tutorial
  http://www.python.org/doc/current/tut/tut.html

• The Python Quick Reference
  http://rgruet.free.fr/PQR2.3.html
Installing Python

• Python is installed on the PCs in 156.
• Python for Win/Mac/Unix/Linux is available from www.python.org.
  – Generally an easy install.
  – On macs, already part of OS X.
  – For NLTK you need Python 2.3 or higher.
• GUI development environment: IDLE.
IDLE Development Environment

• Shell for interactive evaluation.
• Text editor with color-coding and smart indenting for creating python files.
• Menu commands for changing system settings and running files.
• We will use IDLE in class.
Look at a sample of code...

```python
x = 34 - 23            # A comment.
y = "Hello"            # Another one.
z = 3.45
if z == 3.45 or y == "Hello":
    x = x + 1
    y = y + " World"   # String concat.
print x
print y
```

Based on presentation from www.cis.upenn.edu/~cse391/cse391_2004/PythonIntro1.ppt
Look at a sample of code...

```python
x = 34 - 23  # A comment.
y = "Hello"  # Another one.
z = 3.45
if z == 3.45 or y == "Hello":
    x = x + 1
    y = y + " World"  # String concat.
print x
print y
```

Based on presentation from
www.cis.upenn.edu/~cse391/
cse391_2004/PythonIntro1.ppt
Enough to Understand the Code

• Assignment uses = and comparison uses ==.
• For numbers +-*/% are as expected.
  – Special use of + for string concatenation.
  – Special use of % for string formatting.
• Logical operators are words (and, or, not)
  not symbols (&&, ||, !).
• The basic printing command is “print.”
• First assignment to a variable will create it.
  – Variable types don’t need to be declared.
  – Python figures out the variable types on its own.
Basic Datatypes

• Integers (default for numbers)
  \[ z = 5 / 2 \]  # Answer is 2, integer division.

• Floats
  \[ x = 3.456 \]

• Strings
  Can use """" or ""'" to specify. "abc" ‘abc’ (Same thing.)
  Unmatched ones can occur within the string.
  “matt’s”
  Use triple double-quotes for multi-line strings or strings than contain both ‘ and “ inside of them:
  """"""a‘b“c""""""
Whitespace

• Whitespace is meaningful in Python: especially indentation and placement of newlines.
  – Use a newline to end a line of code. (Not a semicolon like in C++ or Java.) (Use \ when must go to next line prematurely.)
  – No braces \{}\} to mark blocks of code in Python… Use consistent indentation instead. The first line with a new indentation is considered outside of the block.
  – Often a colon appears at the start of a new block. (We’ll see this later for function and class definitions.)
Comments

• Start comments with # – the rest of line is ignored.

• Can include a “documentation string” as the first line of any new function or class that you define.

• The development environment, debugger, and other tools use it: it’s good style to include one.

```python
def my_function(x, y):
    """This is the docstring. This function does blah blah blah blah."""
    # The code would go here...
```

Based on presentation from www.cis.upenn.edu/~cse391/cse391_2004/PythonIntro1.ppt
Look at a sample of code...

```
x = 34 - 23          # A comment.
y = "Hello"          # Another one.
z = 3.45
if z == 3.45 or y == "Hello":
    x = x + 1
    y = y + " World"    # String concat.
print x
print y
```
Python and Types

Python determines the data types in a program automatically. “Dynamic Typing”

But Python’s not casual about types, it enforces them after it figures them out. “Strong Typing”

So, for example, you can’t just append an integer to a string. You must first convert the integer to a string itself.

```python
x = "the answer is "  # Decides x is string.
y = 23              # Decides y is integer.
print x + y         # Python will complain about this.
```

Based on presentation from www.cis.upenn.edu/~cse391/cse391_2004/PythonIntro1.ppt
Naming Rules

• Names are case sensitive and cannot start with a number. They can contain letters, numbers, and underscores.
  
  bob  Bob  _bob  _2_bob_  bob_2  BoB

• There are some reserved words:

  and, assert, break, class, continue, def, del, elif, else, except, exec, finally, for, from, global, if, import, in, is, lambda, not, or, pass, print, raise, return, try, while
Accessing Non-existent Name

• If you try to access a name before it’s been properly created (by placing it on the left side of an assignment), you’ll get an error.

```python
>>> y

Traceback (most recent call last):
  File "<pyshell#16>", line 1, in -toplevel-
    y
NameError: name 'y' is not defined

>>> y = 3

>>> y
3
```
Multiple Assignment

• You can also assign to multiple names at the same time.

>>> x, y = 2, 3
>>> x
2
>>> y
3
String Operations

• We can use some methods built-in to the string data type to perform some formatting operations on strings:

    >>> "hello".upper()
    'HELLO'

• There are many other handy string operations available. Check the Python documentation for more.
Printing with Python

• You can print a string to the screen using “print.”

• Using the % string operator in combination with the print command, we can format our output text.

```python
>>> print "%s xyz %d" % ("abc", 34)
abc xyz 34
```

“Print” automatically adds a newline to the end of the string. If you include a list of strings, it will concatenate them with a space between them.

```python
>>> print "abc" >>> print "abc", "def"
abc abc def
```
Hands On

Okay, let’s try it.

- Or on these PCs under Documentation.
- First two sections you should read. For now:
- Start tonight with Section 3, *An Informal Introduction to Python*, and work through the examples given.
Algorithm Analysis (Big O)

CS-341
Dick Steflik
Complexity

• In examining algorithm efficiency we must understand the idea of complexity
  – Space complexity
  – Time Complexity
Space Complexity

• When memory was expensive we focused on making programs as space efficient as possible and developed schemes to make memory appear larger than it really was (virtual memory and memory paging schemes)

• Space complexity is still important in the field of embedded computing (hand held computer based equipment like cell phones, palm devices, etc)
Time Complexity

• Is the algorithm “fast enough” for my needs
• How much longer will the algorithm take if I increase the amount of data it must process
• Given a set of algorithms that accomplish the same thing, which is the right one to choose
Algorithm Efficiency

• a measure of the amount of resources consumed in solving a problem of size $n$
  – time
  – space
• Benchmarking: implement algorithm,
  – run with some specific input and measure time taken
  – better for comparing performance of processors than for comparing performance of algorithms
• Big Oh (asymptotic analysis)
  – associates $n$, the problem size,
  – with $t$, the processing time required to solve the problem
Cases to examine

- **Best case**
  - if the algorithm is executed, the fewest number of instructions are executed

- **Average case**
  - executing the algorithm produces path lengths that will on average be the same

- **Worst case**
  - executing the algorithm produces path lengths that are always a maximum
Worst case analysis

• Of the three cases, only useful case (from the standpoint of program design) is that of the worst case.

• Worst case helps answer the software lifecycle question of:
  – If its good enough today, will it be good enough tomorrow?
Frequency Count

- examine a piece of code and predict the number of instructions to be executed

  for each instruction predict how many times each will be encountered as the code runs

<table>
<thead>
<tr>
<th>Inst</th>
<th>Code</th>
<th>F.C.</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>for (int i=0; i&lt; n ; i++)</td>
<td>n+1</td>
</tr>
<tr>
<td>1</td>
<td>{ cout &lt;&lt; i;</td>
<td>n</td>
</tr>
<tr>
<td>2</td>
<td>p = p + i;</td>
<td>n</td>
</tr>
<tr>
<td>3</td>
<td>}</td>
<td></td>
</tr>
</tbody>
</table>

totaling the counts produces the F.C. (frequency count)
Order of magnitude

• In the previous example:
  – best_case = avg_case = worst_case
  – Example is based on fixed iteration $n$

• By itself, Freq. Count is relatively meaningless

• Order of magnitude -> estimate of performance vs. amount of data

• To convert F.C. to order of magnitude:
  – discard constant terms
  – disregard coefficients
  – pick the most significant term

• Worst case path through algorithm ->
  – order of magnitude will be Big O (i.e. $O(n)$)
Another example

<table>
<thead>
<tr>
<th>Inst</th>
<th>Code</th>
<th>F.C.</th>
<th>F.C.</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>for (int i=0; i&lt; n ; i++)</td>
<td>n+1</td>
<td>n+1</td>
</tr>
<tr>
<td>1</td>
<td>for int j=0 ; j &lt; n; j++)</td>
<td>n(n+1)</td>
<td>n²+n</td>
</tr>
<tr>
<td>2</td>
<td>{ cout &lt;&lt; i;</td>
<td>n*n</td>
<td>n²</td>
</tr>
<tr>
<td>3</td>
<td>p = p + i;</td>
<td>n*n</td>
<td>n²</td>
</tr>
<tr>
<td>4</td>
<td>}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

discarding constant terms produces: 3n²+2n

clearing coefficients: n²+n

picking the most significant term: n²

Big O = O(n²)
What is Big O

- Big O
  - rate at which algorithm performance degrades as a function of the amount of data it is asked to handle

- For example:
  - $O(n)$ -> performance degrades at a linear rate $O(n^2)$ -> quadratic degradation
Common growth rates
Big Oh - Formal Definition

• Definition of "big oh":
• \( f(n) = O(g(n)) \), iff there exist constants \( c \) and \( n_0 \) such that: \( f(n) \leq c \cdot g(n) \) for all \( n \geq n_0 \)
• Thus, \( g(n) \) is an upper bound on \( f(n) \)
• Note:
  \[
  f(n) = O(g(n))
  \]
  is NOT the same as
  \[
  O(g(n)) = f(n)
  \]
• The '=' is not the usual mathematical operator "=" (it is not reflexive)
Asymptotic Growth Rates
– “Big-O” (upper bound)

- $f(n) = O(g(n))$ [f grows at the same rate or slower than g] iff:

There exists positive constants $c$ and $n_0$ such that $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$

- $f$ is bound above by $g$
  - Note: Big-O does not imply a tight bound
- Ignore constants and low order terms
Big-O, Examples

• E.G. 1: \(5n^2 = O(n^3)\)
  \(c = 1, n_0 = 5: 5n^2 \leq n \times n^2 = n^3\)

• E.G. 2: \(100n^2 = O(n^2)\)
  \(c = 100, n_0 = 1\)

• E.G. 3: \(n^3 = O(2^n)\)
  \(c = 1, n_0 = 12\)
  \(n^3 \leq (2^{n/3})^3, n \leq 2^{n/3}\) for \(n \geq 12\) [use induction]
Little-o
Loose upper bound

• \( f(n) = o(g(n)) \) [f grows strictly slower than g]

\( \Leftrightarrow f(n) = O(g(n)) \) and \( g(n) \neq O(f(n)) \)

\[ \lim_{n \to \infty} f(n)/g(n) = 0 \]

• “f is bound above by g, but not tightly”
Little-o, restatement

- \( \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0 \implies f(n) = o(g(n)) \)

- \( \forall \varepsilon > 0, \exists n_0 \text{ s.t. } \forall n \geq n_0, \frac{f(n)}{g(n)} < \varepsilon \)
Equivalence - Theta

- $f(n) = \Theta(g(n))$ [grows at the same rate]
  - $\iff f(n) = O(g(n))$ and $g(n) = O(f(n))$
  - $\iff g(n) = \Theta(f(n))$

- $\lim_{n \to \infty} \frac{f(n)}{g(n)} = c$, $c \neq 0$
  - $\iff f(n) = \Theta(g(n))$

- “f is bound above by g, and below by g”
Common Results

• \([j < k]\) \(\lim_{n \to \infty} \frac{n^j}{n^k} = \lim_{n \to \infty} \frac{1}{n^{(k-j)}} = 0\)
  \(\Rightarrow n^j = o(n^k), \text{ if } j<k\)

• \([c < d]\) \(\lim_{n \to \infty} \frac{c^n}{d^n} = \lim_{n \to \infty} (c/d)^n = 0\)
  \(\Rightarrow c^n = o(d^n), \text{ if } c<d\)

• \(\lim_{n \to \infty} \frac{\ln(n)}{n} = \infty/\infty\)
  \(\Rightarrow \lim_{n \to \infty} \frac{\ln(n)}{n} = \lim_{n \to \infty} \frac{1/n}{1} = 0 \text{ [L’Hopital’s Rule]}\)
  \(\Rightarrow \ln(n) = o(n)\)

• \([\varepsilon > 0]\) \(\ln(n) = o(n^\varepsilon) \text{ [similar calculation]}\)
Common Results

• \([c > 1, \text{ k an integer}]\)

\[
\lim_{n \to \infty} \frac{n^k}{c^n} = \frac{\infty}{\infty}
\]

\[\Rightarrow \lim_{n \to \infty} \frac{kn^{k-1}}{c^n \ln(c)}\]

\[\Rightarrow \lim_{n \to \infty} \frac{k(k-1)n^{k-2}}{c^n \ln(c)^2}\]

\[\Rightarrow \ldots\]

\[\Rightarrow \lim_{n \to \infty} \frac{k(k-1)\ldots(k-1)}{c^n \ln(c)^k} = 0\]

\[\Rightarrow n^k = o(c^n)\]
Asymptotic Growth Rates

• $\Theta(\log(n))$ – logarithmic [$\log(2n)/\log(n) = 1 + \log(2)/\log(n)$]
• $\Theta(n)$ – linear [double input $\Rightarrow$ double output]
• $\Theta(n^2)$ – quadratic [double input $\Rightarrow$ quadruple output]
• $\Theta(n^3)$ – cubit [double input $\Rightarrow$ output increases by factor of 8]
• $\Theta(n^k)$ – polynomial of degree $k$
• $\Theta(c^n)$ – exponential [double input $\Rightarrow$ square output]
Asymptotic Manipulation

• $\Theta(cf(n)) = \Theta(f(n))$

• $\Theta(f(n) + g(n)) = \Theta(f(n))$ if $g(n) = O(f(n))$