CS3101: Programming Languages - Python

(CVN) Spring 2009

Agenda

- Administrative Details
- Course Description
- Getting Started
  - The Role of Scripting Languages in a Heterogeneous Software World
  - Language Overview
  - Writing your first scripts
- Assignment 1

Administrative Details

Your instructor

- Joshua Gordon
- Second semester PhD student
- Research interests: AI / NLP / Software Systems Engineering
- Life interests: biking, hiking
- Contact:
  - joshua@cs.columbia.edu
  - Telephone number by request

Office Hours

- Feel free to contact me anytime by email - always happy to help
- For more involved questions let me know a day or so in advance and we can schedule a phone appointment
- Essential to begin a dialog for the course project

Course resources

- Website:
  - www.cs.columbia.edu/~joshua/teaching/cs3101/cvn
  - Lecture notes, references, assignments and the course project
  - Courseworks - discussion board

Grading

- Three homework assignments (worth 40% total)
- Optional Challenge problems excuse you from the regular HW
- Final project of your own choosing (with my input) - worth 60%
- Why no final
  - Practical vs. academic knowledge
  - Syntax vs. semantics
- Grace days: 2, after which late assignments are accepted at -10% / day

Academic Honesty

- http://www.cs.columbia.edu/education/honesty
- Special notes:
  - Just about everything Python related can be found on the web
  - This is a good thing, however, do NOT fall into two pitfalls
  - 1) copying a solution or segment of a solution - always reference your sources
  - 2) learning only to concatenate code

CVN

- How this course differs from the campus version
  - Caveat: audience, interactivity
    - Will do my best to find students
  - Pace / surveys
  - From the basics up
    - second year CS student as the baseline
  - Accelerate as we go

Course Description

Project

- Opportunity to leverage Python to accomplish something useful to YOU
  - Requirements:
    - draft proposal and approval
- Grading criteria
  - Concept
  - Implementation
  - Demonstrated proficiency, library use

Previous Projects

- Genetic algorithms to tackle NP problems
- Solar system animation via programmatic input to MAYA
- Music recommendation system
- A personal resource page for elementary students
- Financial engineering utilities
- Software library database builder
- Checkers

Course Content

- + Learning the core language
- + Acquiring proficiency in solving common scripting tasks
- - We are not a course in advanced programming
- Objective: Becoming a good Python programmer
  - What does that mean?
    - Efficiency w.r.t. to your time
    - Ability to locate the resources you need
    - Understanding the appropriate solution for the job
  - Python is all about the libraries

Topics

- The course will at a minimum cover (in addition to the essentials)
  - Libraries galore, including: database and network connectivity, system utilities, numeric computation, C / C ++ integration, compression, win32 API (basics), xmlrpc, threading...
  - Regular expressions
  - Debugging and Optimization
  - Appropriate use cases
  - Performance (yours vs. your code's)

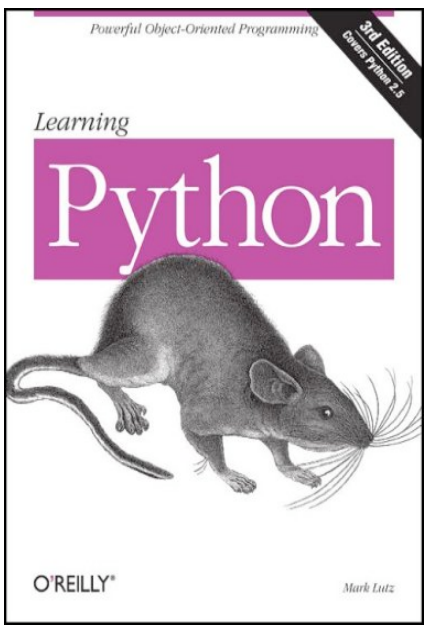
Where Python has been useful to me

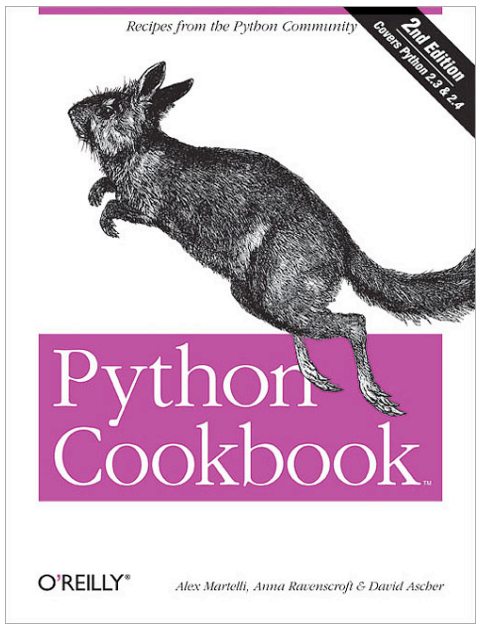
- Developing an iPhone application
  - the server side core language is Python
  - reason: easily leverages the C and JAVA components needed, facilitates writing the remainder
- Enhancing a Spoken Dialog System
  - Everywhere really: setting up the database connection, parsing recognition hypothesis, creating and mining log files, network communication
- Sport recommender - leveraging code from the previous course
- *Removing copyrighted content from slides

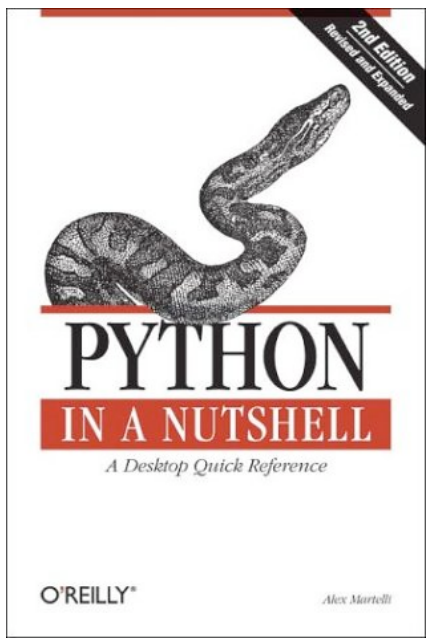
References

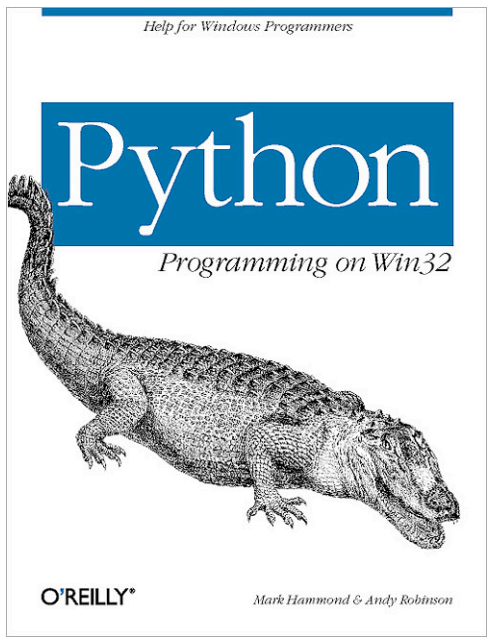
- The online documentation is outstanding and your best bet as a library reference
  - www.python.org/doc
- Useful books (depending on your skill level)
  - Learning Python by Mark Lutz, 3rd Edition
  - **Python in a Nutshell by Alex Martelli, 2nd Edition**
  - Python Cookbook (multiple authors), 2nd Edition (dated but still useful)

Ordered by technical complexity - notice anything?









(images copyright O'Reilly Media)

Word on Editions

- Python 3000 (released early 2009)
- Intentionally backwards incompatible
  - Major changes:
    - print is a function (previously a statement), API modifications (often views and iterators instead of lists), text vs. data instead of unicode vs. 8-bit
    - 2to3 tool available
- Reference: http://docs.python.org/3.0/whatsnew/3.0.html
- Which version the course assumes, and a note on assignments

Learning a language

- Finally, strategies that have proven valuable to last semesters students
  - Iteration and refinement - first make it work, then make it elegant
  - Copious examples
  - Side by side comparison against a familiar language
  - Think before using the web as a crutch

The Zen of Python

- From calling: “python -c "import this”

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.

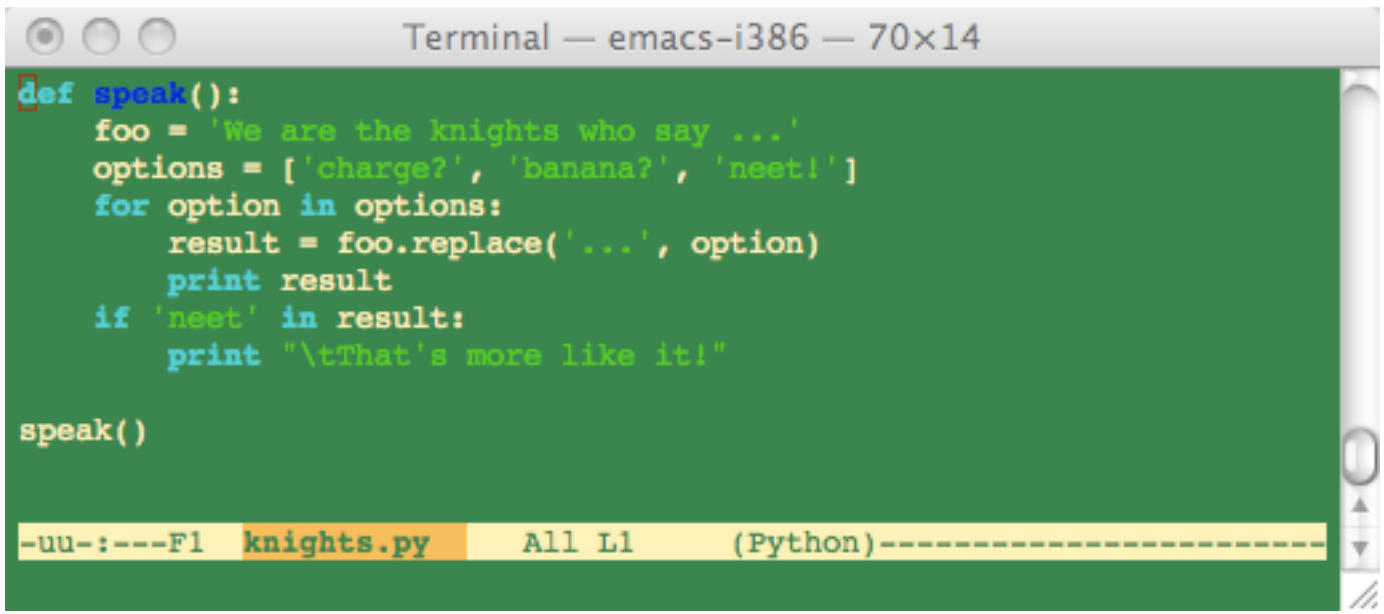
Pythonic Style: Self documenting code

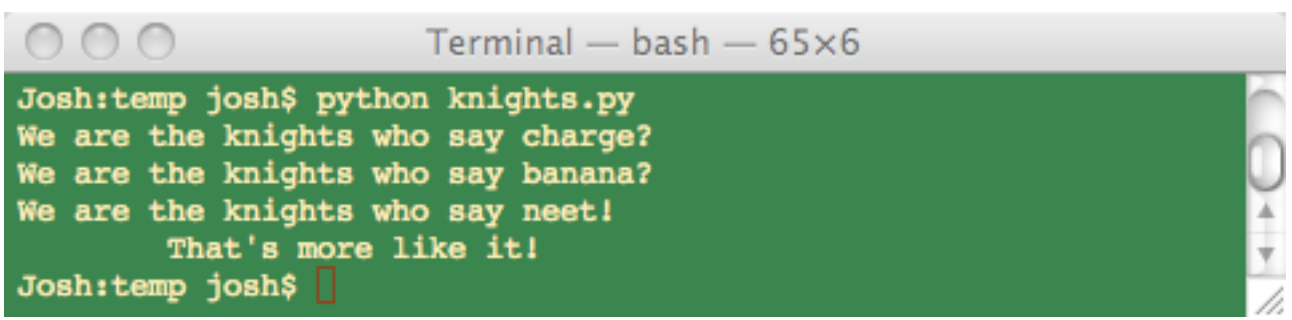
```
foo = [‘a’, ‘b’, ‘c’]
length = len(foo)
```

```
@foo = (“a", “b“, “c");
my $length =
   scalar(@list);
my $anotherWay =
   $#list + 1;
$length = @foo;
if(@list > 0) {
…
}
```

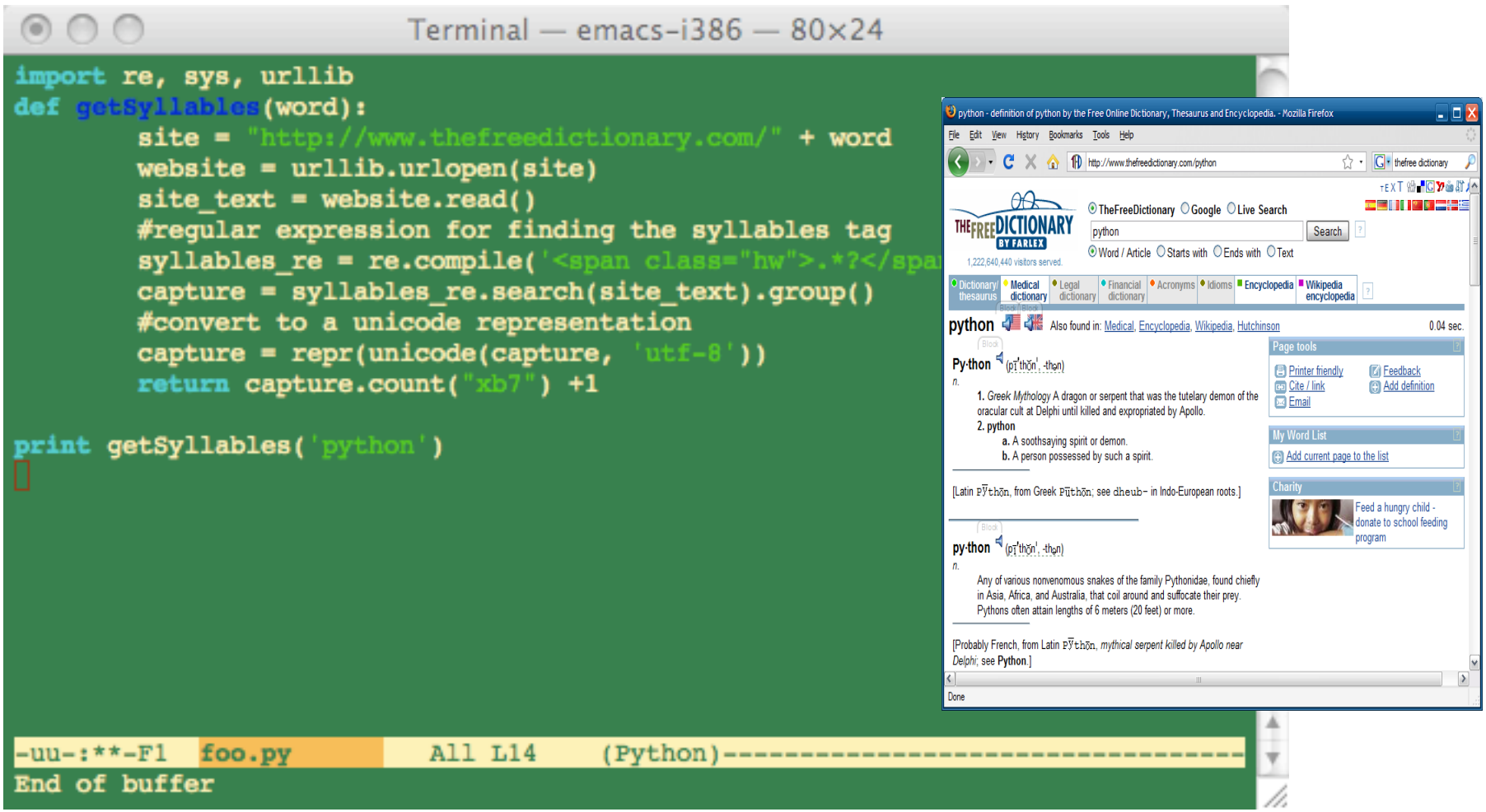
Discussion: Does anyone have any experience with Perl? Horror stories? Ever tried reading someone else’s code?

General Syntax





Basic Libraries Preview



Notice dynamic typing, indentation based scope, end of a line indicates the end of a statement

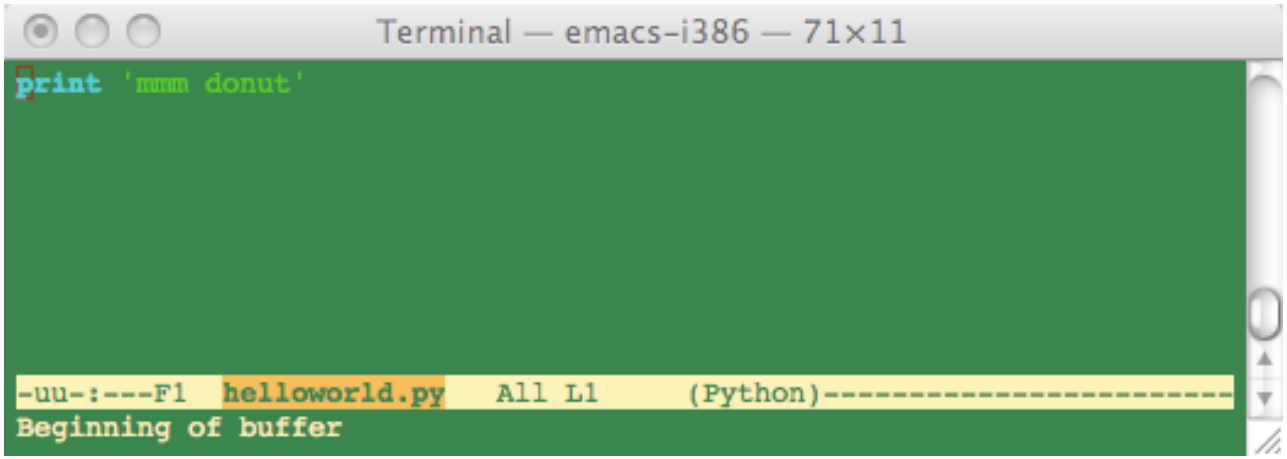
Getting Started

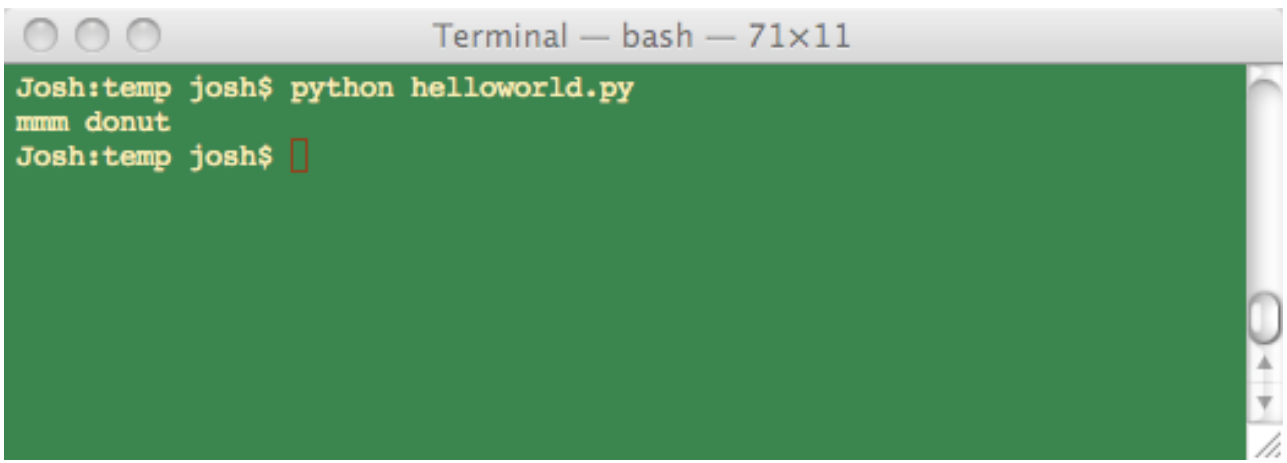
Available Environments

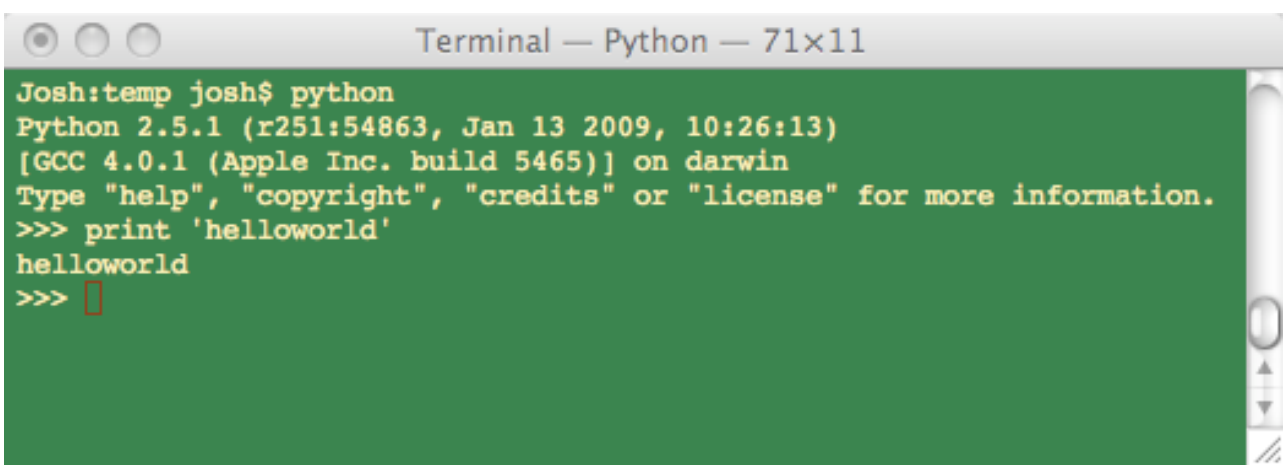
- To develop software in Python you create text files that contain source code and documentation
- Any text editor is allowable, though you're better off using one with Python support
- IDEs (integrated development environments are recommended for heavier code)
- Terminology: script, a standalone program, module: a file intended to be imported from another program

Running programs

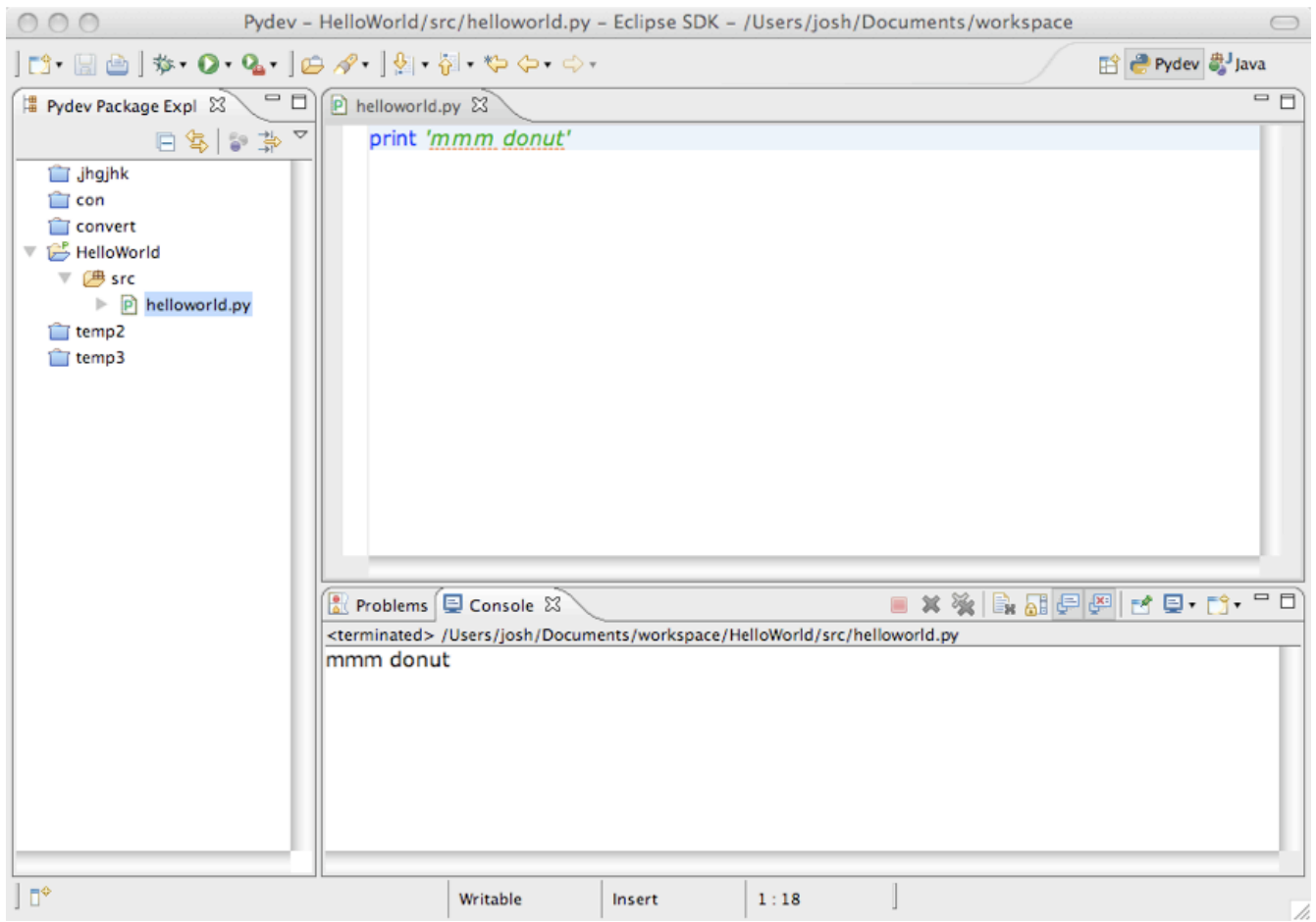
- Python programs are run by passing them to the interpreter (or via an interactive session)
- Code has the extension (.py) - compiled bytecode (.pyc)
- Python interpreters compile statements to byte code and execute them on a virtual machine
- Compilation occurs automatically the first time the interpreter reads code which imports modules, and subsequently upon modification
  - Standalone scripts are recompiled when run directly
- Compilation and execution are transparent to the user


Terminal — emacs-i386 — 71×11
print 'mmm donut'
-uu-:---F1 helloworld.py All L1 (Python)----------
Beginning of buffer


Terminal — bash — 71×11
Josh:temp josh$ python helloworld.py
mmm donut
Josh:temp josh$


Terminal — Python — 71×11
Josh:temp josh$ python
Python 2.5.1 (r251:54863, Jan 13 2009, 10:26:13)
[GCC 4.0.1 (Apple Inc. build 5465)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print 'helloworld'
helloworld
>>>

Eclipse with PyDev



http://pydev.sourceforge.net/

Integrated Development Environments

- Simplify: debugging, project management, code completion, library browsing, repository integration
- Disadvantages: heavyweight, cumbersome, learning curve
- Optional for this course per your preference, though recommended unless you're already an emacs expert

Language fundamentals

Caveat: these next few slides are low level - familiarity with these topics is essential to understanding compiler errors

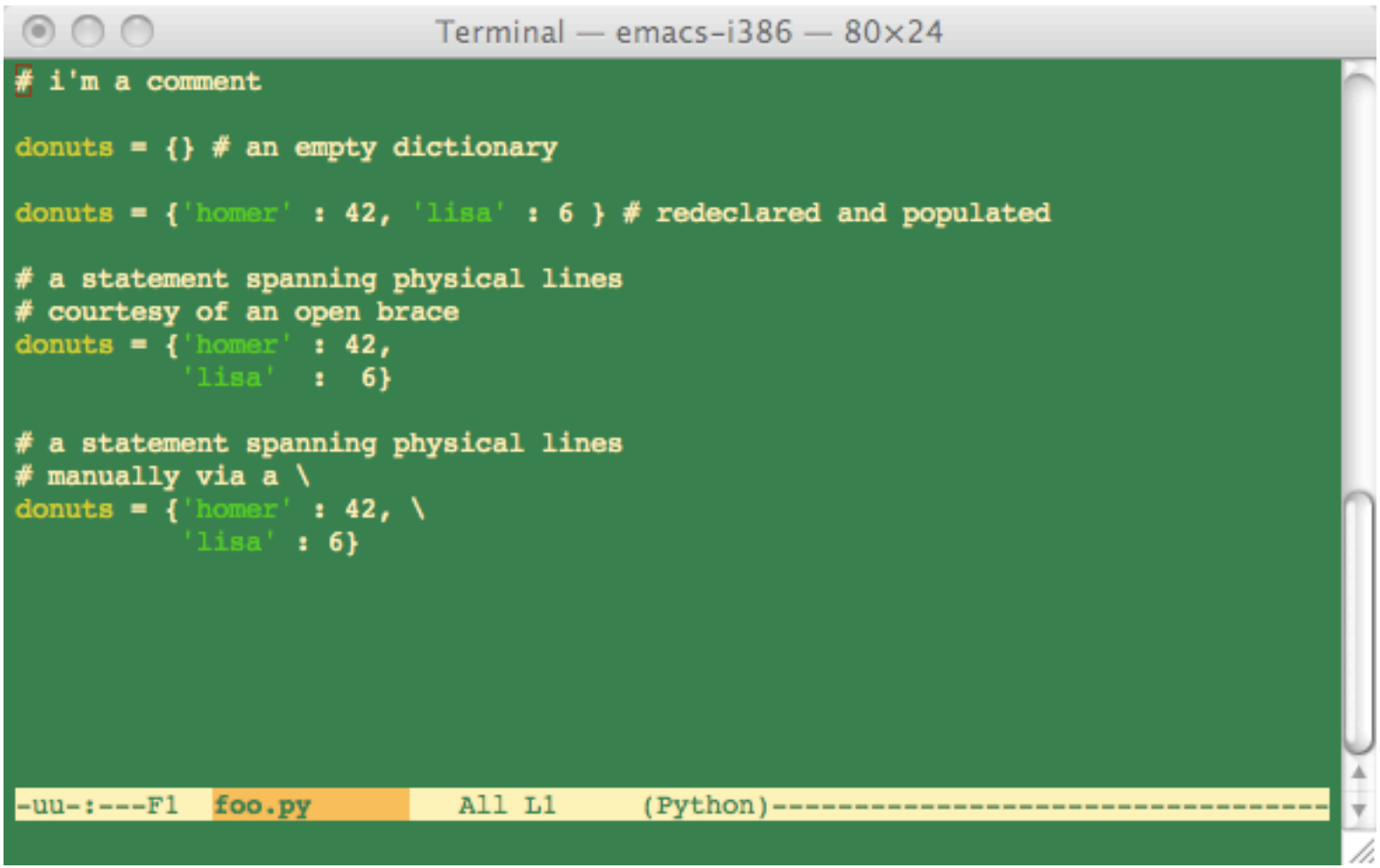
Lexical Structure

- The lowest level syntax of the language - specifies, for instance, how comments and variable names appear
- Python programs are composed of a set of plain text ASCII encoded source files
- Like other language, Python source files are a sequence of characters, however:
  - unlike in C or JAVA, in Python we're interested in lines - whitespace counts

Logical vs. Physical Lines

- Python programs consist of a sequence of logical lines, which may contain one or more physical lines
  - lines may end with a comment
  - blank lines are ignored by the compiler
  - the end of a physical line marks the end of a statement
- Producing readable code:
  - physical lines may be joined by a (\) character
  - if an open parenthesis ((), brace ({), or bracket ([) has not yet been closed, lines are joined by the compiler)

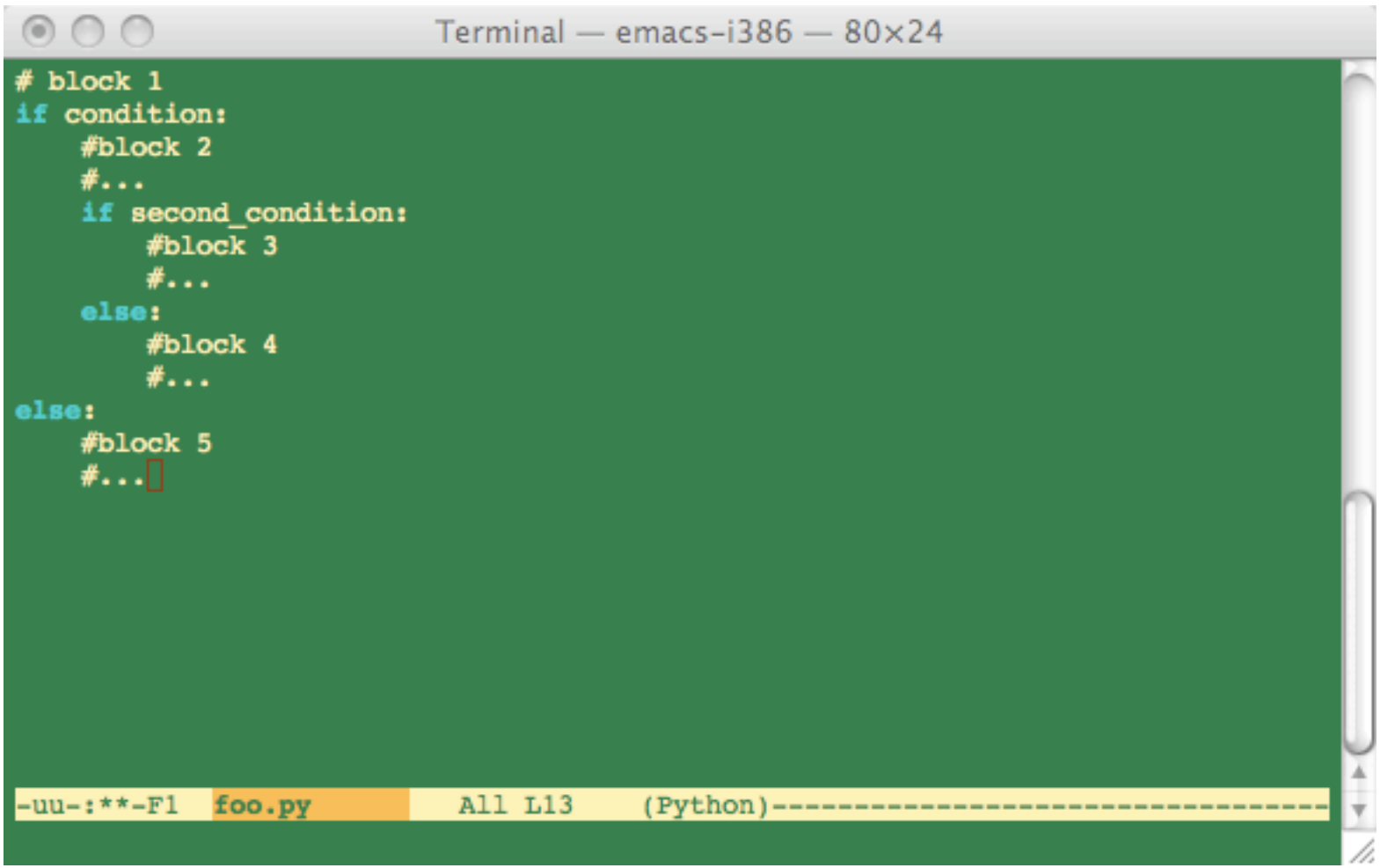
Let's see how this looks



Indentation

- Indentation is used to express block structure
- Unlike C or JAVA (or most languages in fact) indentation is the only way to denote blocks
- Blocks are delineated by contiguous whitespace sequence, typically in units of four spaces, immediately to the left of a statement
- All statements indented by the same amount belong to an identical block
- Indentation applies only to the first physical line in a logical block
- The first physical line in a source file must have no indentation

Let's see how this looks



Tokens and Identifiers

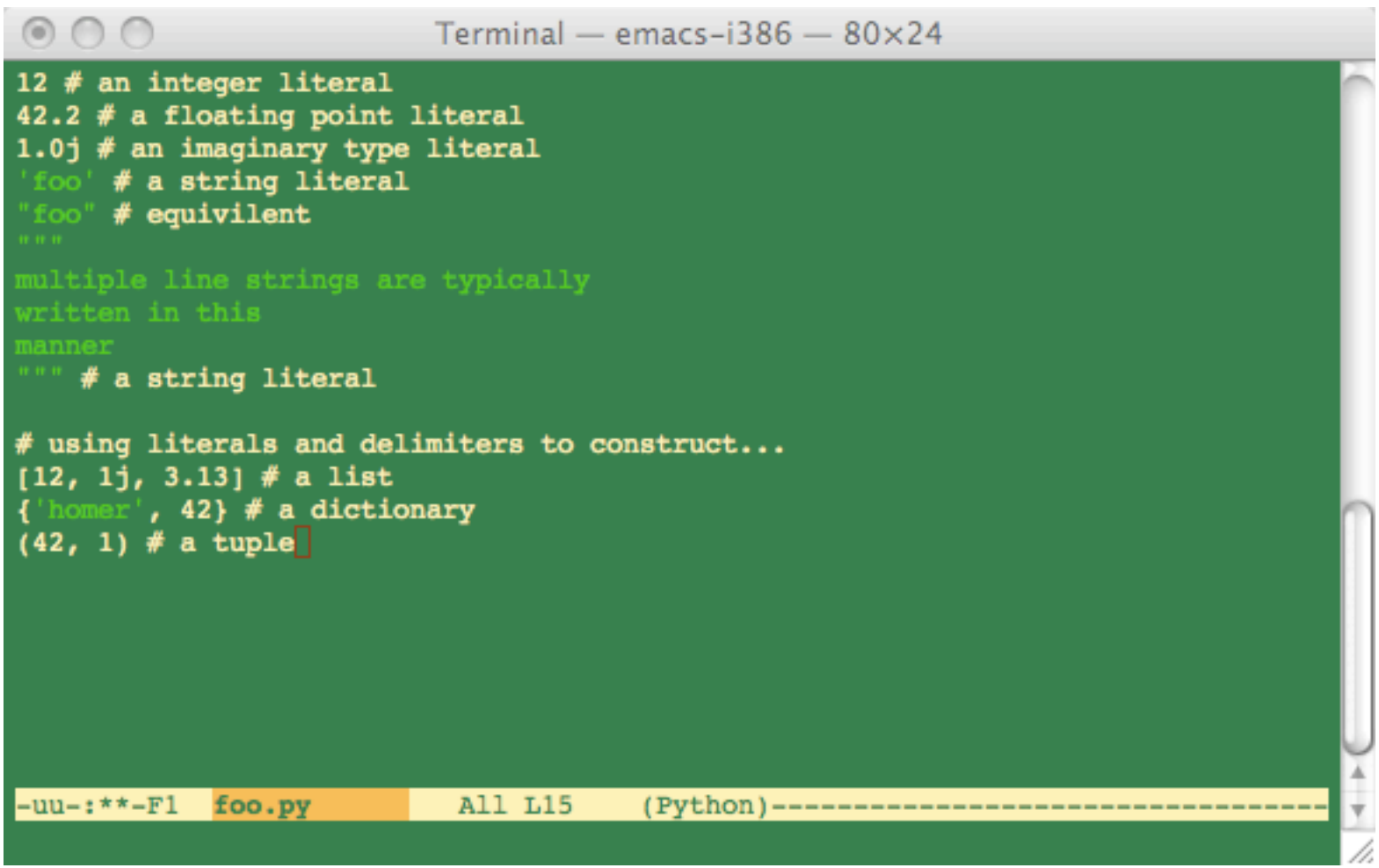
- Logical lines are understood as sequences of tokens
- Tokens are substrings of the line, which correspond to identifiers, keywords, operators, delimiters, and literals
- Identifiers name functions, variables, classes, or modules
  - Identifiers start with a character or an underscore
  - Python is case sensitive
  - Punctuation is disallowed within identifiers
  - Convention: start a class name with an uppercase character, and everything else with a lower

Keywords, Operators, and Delimiters

- Keywords are reserved identifiers - Python has about 30 of them, many should be familiar to other languages
  - (and, assert, break, class, continue, def, del, and so forth)
- Non-alphanumeric characters (and combinations) are used by Python as operators
  - (+, -, *, /, <=, <>, !=, and so forth)
- These symbols are used as delimiters in expressions, lists, dictionaries, and sets
  - ((,), [,], {, }, , and so forth)

Literals

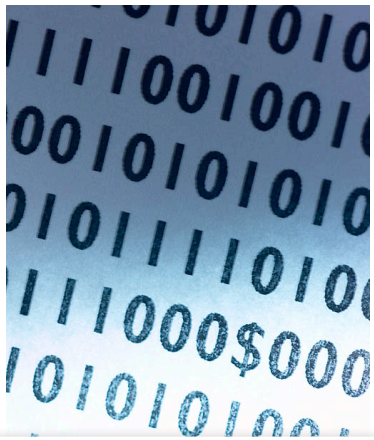
- A literal is a string or numerical value that appears directly in program text



Statements and Expressions

- A Python program may be understood as a sequence of simple and compound statements
- Unlike C or JAVA, Python does not have any forward declarations or other top level syntax
- The general rule is one statement per line (statements can be terminated with (;), but it's unusual style)
- Statements may be expressions (a phrase which evaluates to produce a value) or assignments
- The simplest expressions are literals and identifiers
- Expressions are built by joining subexpressions with operators and identifiers

Assignments



Python

- Python is dynamically typed and garbage collected
  - Variables need not be explicitly declared or allocated
- Foo = 42
- Foo = 42.42
- Foo = “dolphin”
- Foo = [“one”, “two”, “three”]
- Foo = homer()

C

- int foo = 42;
- float foo = 42.42;
- char *cp = “dolphin";

Java

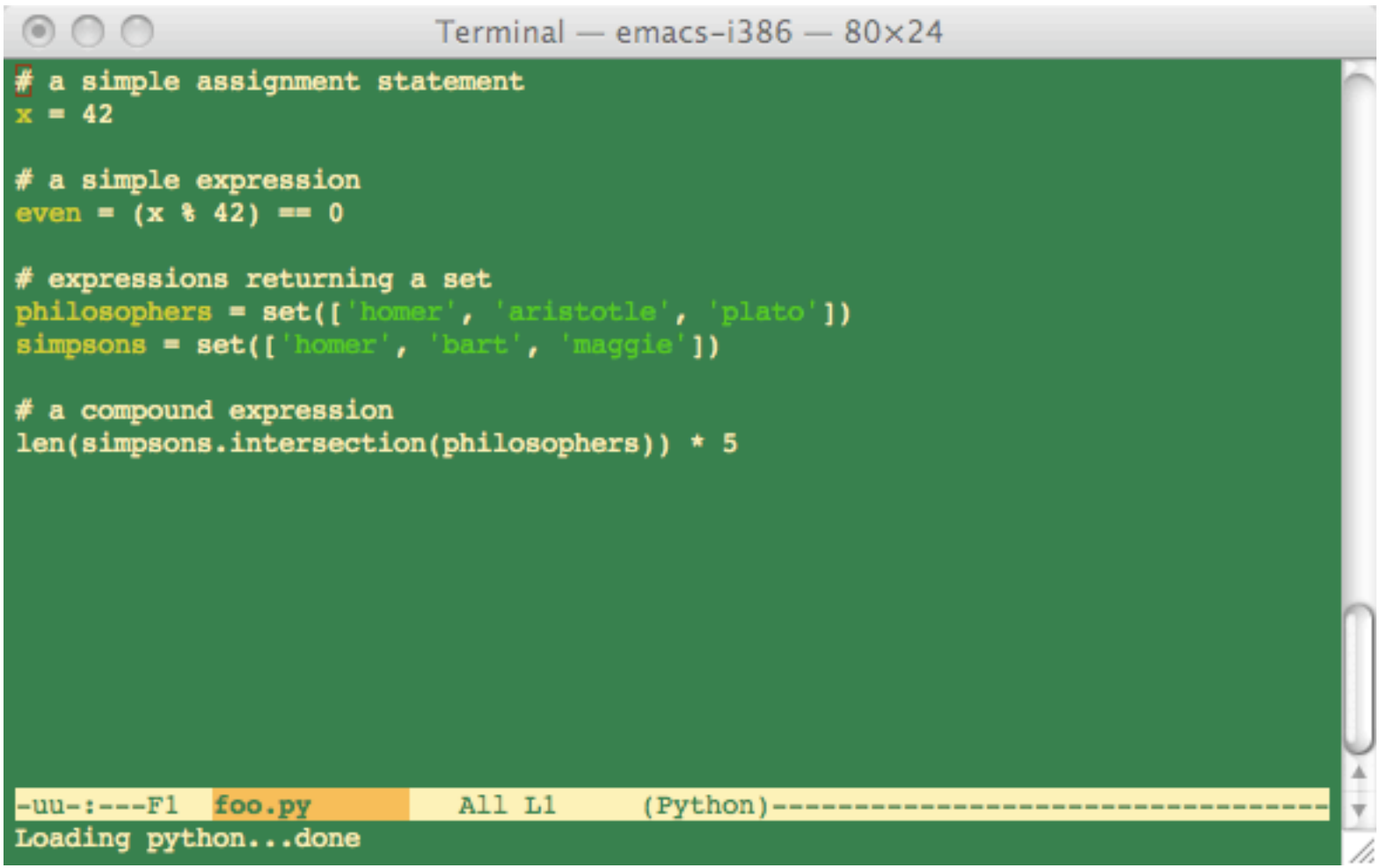
List<String> foo = new ArrayList<String>();

Homer foo = new Homer();

Compound statements

- Contain a set of statements and control their execution
- Compound statements contain a set of clauses
  - Each clause has a header starting with a keyword, and ending with a (:)
  - Followed by a body, which is itself a sequence of statements, terminated when indentation returns to the outer level
- Also legal are simple statements following the (:)

Let's see how this looks



Scripting Philosophy

Scripting

- Historically scripting refers to authoring simple tasks, generally in a high level interpreted language. Today the meaning is less clear.
- Scripting probably better refers to Pythons approach to development, rather than what is capable in the language
  - Scripting fosters an exploratory, incremental approach to programming
  - The developer scales up in complexity and power as necessary

Philosophy

- Coordination in a heterogeneous software world
  - linking components written in diverse languages
  - Python code is often deployed in the context of larger applications
- Rapid prototyping
  - exploring algorithmic performance before a detailed implementation
- Concatenative programming
  - creating software by intertwining libraries

Performance

- Key notion: developer vs. computational efficiency
  - common misconception that code is often CPU bound
  - direct your time where it's valuable
- Like JAVA, Python is compiled to byte-code
  - Portability at the expense of speed
  - The core language, however is highly optimized
    - built-in data types are implemented in C
    - built-in methods are thoughtful - sort is approximately 1200 lines of C in later versions of Python
- More commonly than JAVA, you'll see Python deployed in high performance environments - Boost / LLNL

The Core Language: Data types

Overview

- All data values in Python are objects, and each object has a type
- Type determines supported operations
  - For instances, lists support .reverse(), but strings (as immutable objects) do not
- Mutable vs. immutable objects
- Useful functions: type(obj) and isinstance(obj, type)
- Built-in types cover numbers, strings, lists, tuples, and dictionaries
- User defined type are supported via classes

Numbers

- Support for integers (including long, if you're coming from C) floating point, and complex numbers
- All numbers in Python are immutable - so any operation on a number always produces a new object
- Up casting is automatic
- Unlimited precision

Numeric Literals

- Beginning from the basics
- Dynamic typing
- Python offers unlimited precision. You may indeed compute 2^1000,000 (but not if you're in a rush)

Syntax	Result
4, -24, 0	Standard Integer (corresponds to C longs)
12.4, 3.14e-10, 4.0e+210	Floating Point (corresponds to C doubles)
1e10, 1E1	Scientific Notation
9L	Unlimited Precision Long Integer
3+4j	Complex
0177, 0x9ff, 0xFF	Octal and hex literals for integers

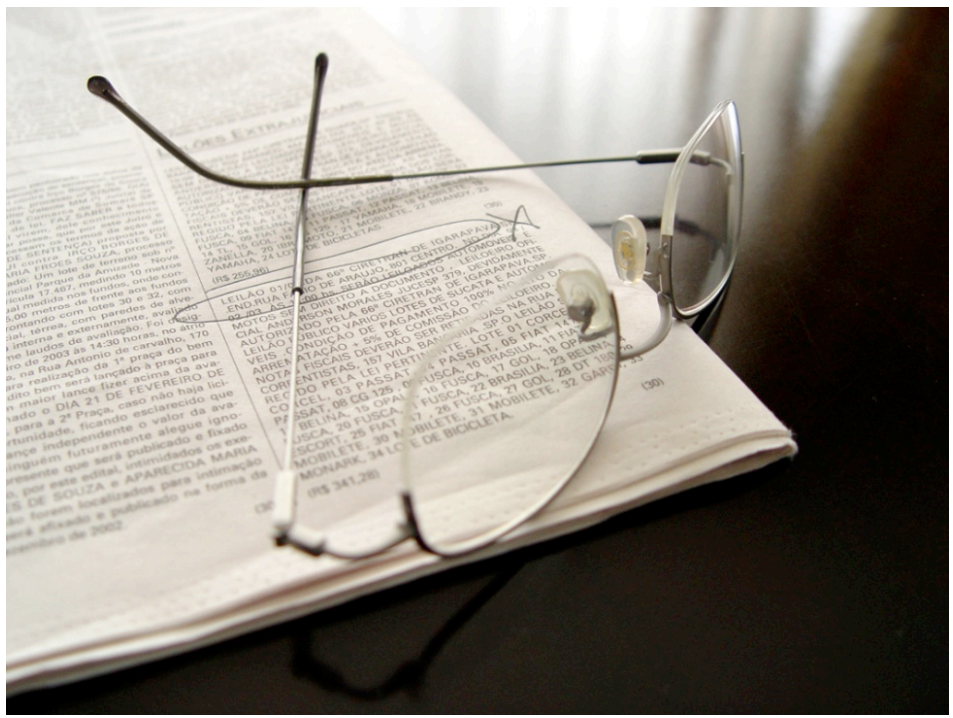
Basic numeric operations

- NumPy / SciPy provide heavy lifting
  - Think a free Python implementation of MATLAB
  - Matrix data types, vector processing, sophisticated computation libraries
  - http://numpy.scipy.org/
- Basic mathematical modules are included by default
  - "import math" gives you square root, common constants, etc

Syntax	Result
1 + 1	2
1 - 2.0	-1.0
2 * 12	24
5 / 2	2
5.0 / 2	2.5
5 % 2	1
2 ** 8.1	274.37
2 * 5e100	9.99e+100
2.0 * 10e1	200.0
math.pi	3.141592...
math.sqrt(85)	9.2195...

Text processing

- Text processing is fundamental to scripting languages
- Historically a syntactic burden in C and cumbersome in Java
- Python excels in this area (so does Perl)



Strings

- Strings are immutable objects which store a sequence of characters (plain or Unicode)
  - May be used to represent arbitrary sequences of binary bytes
- Quoting: single or double allowed so long as you're consistent, triple quoting for multiline

String literals

- Unicode support
  - Useful for multilingual text and special characters
  - Very strong support in Python 3000
- Raw string encoding
  - r“C:\Josh\Document’s\”
- All the typical escape characters
  - “\n” – new line
  - “\t” – tab

Syntax	Result
‘Marge’	‘Marge’
“Homer”	‘Homer’
“Lisa’s Music”	“Lisa’s Music”
“Homer’s\tdonut”	“Homer’s donut”
u’bart\u0026lisa’	u’bart&lisa’
r”C:\Simpson\Bart\”	“C:\\Simpson\\Bart”

Basic string operations

Syntax	Result
s1 = “Josh” s2 = “a marathon”	Declares a new variable
s1 + “ is training for “ + s2	‘Josh is training for a marathon’ (concatenation)
len(“abc”)	3 (length)
s1[0]	‘J’ (give me the first character in s1)
s1 * 3	“JoshJoshJosh” (repeat)

Common theme with Python: basic operations are abundant and behave as you would expect – the complexity of syntax corresponds with the complexity of the operation

Basic string methods

- Built-in methods are efficient
- Typically you can guess method names once you understand the pattern
- .startswith(), .endswith(), etc
- Always see the documentation, many methods exist, avoid reinventing the wheel

Syntax	Result
foo = “The Simpsons”	Declares a new variable
foo.lower()	‘the simpsons’
foo.find(“S”)	4
foo.find(“z”)	-1
foo.split(“ “)	[“The”, “Simpsons”]
foo.islower()	False
“The” in foo	True
“ whitespace “.strip()	“whitespace”

Lists

- A list is an ordered, mutable sequence of items
- Lists may be composed of arbitrary objects of different types
- Arbitrary nesting is allowed

Basic construction and indexing

- Lists are:
  - Ordered collections of arbitrary objects
  - Accessed by offset
  - Variable length and heterogeneous
  - Mutable (unlike strings, lists may be changed in place)
  - Nestable

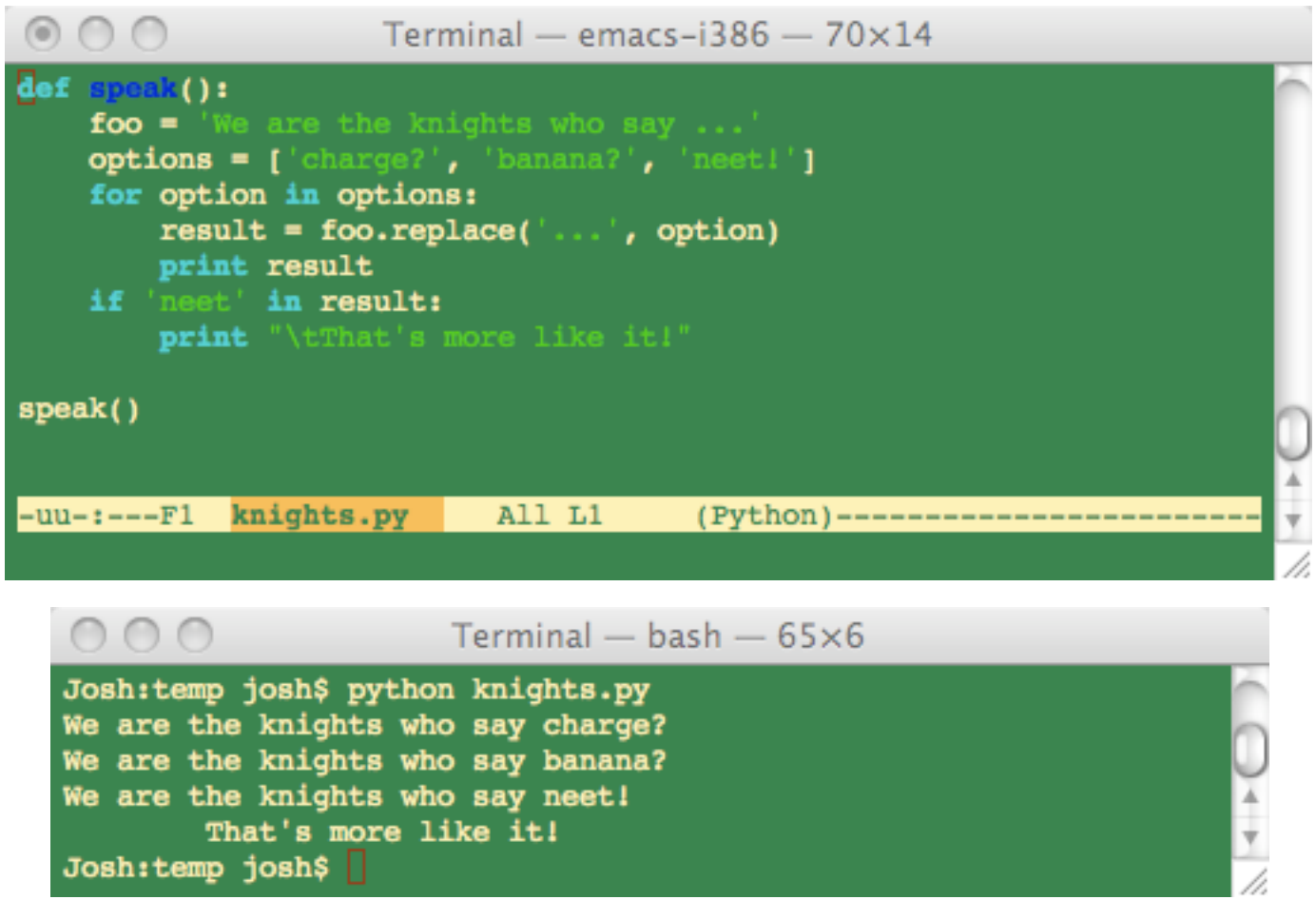
Foo = []	(an empty list)
Foo = [0,1,"bar"]	(a list containing two integers and a string)
Foo[2]	"bar" (the second element in foo)
Foo = [0,1,["a", "nested", "list"], 3]	(a nested list)
Foo[2]	["a", "nested", "list"] (the third element of Foo)
Foo[2][1]	"nested" (the second element of the third element of foo)

Basic list methods

- Appending to or popping from the list modifies the original copy
- Notice that sorting is a built-in method
  - A common feature of scripting languages
  - Later we'll discuss user defined sort criteria
  - Highly optimized, handles many special common cases
  - A partially sorted list, a list constructed from two sorted lists, a reversed list, etc
- Slicing and indexing:
  - Foo = ["a", "b", "c", "d"]
  - Foo[-2]
    - Counts from the right, returns 'c'

Foo = []	(empty list)
Foo.append("a")	["a"]
Foo.append("c")	["a", "c"]
Foo.append("b")	["a", "c", "b"]
Foo.append("x")	["a", "c", "b", "x"]
Foo.pop()	"x" (removes and returns the last element from Foo)
Foo.sort()	["a", "b", "c"]
Foo.index("c")	2
Foo[1:]	["b", "c"] (slicing: give me all the elements beginning from index 1)

Basic list membership, string manipulation



Basic nested lists

```
def main():
    matrix = [["upper left",2,3],[4,"center",6],[7,8,"lower right"]]
    print matrix[1][1]
    print "The upper row“ + matrix[0]

main()

→
center
The upper row: ['upper left', 2, 3]
```

Dictionaries

- Dictionaries are mappings: arbitrary collections of objects indexed by (almost) arbitrary keys
- Underlying implementation is a hashmap
- One of the more optimized types in Python, used properly many operations are constant time
- Items in a dictionary are key / value pair
- dict() function for creation
  - donut_supply = dict(homer=12, marge=6)

Dictionaries

- Dictionaries are unordered collections
- Unlike lists, which are accessed by index, dictionaries are accessed by **key**
- Like lists, dictionaries are mutable and heterogeneous
- Methods .keys(), .values(), .items() vs. iterkeys(), etc

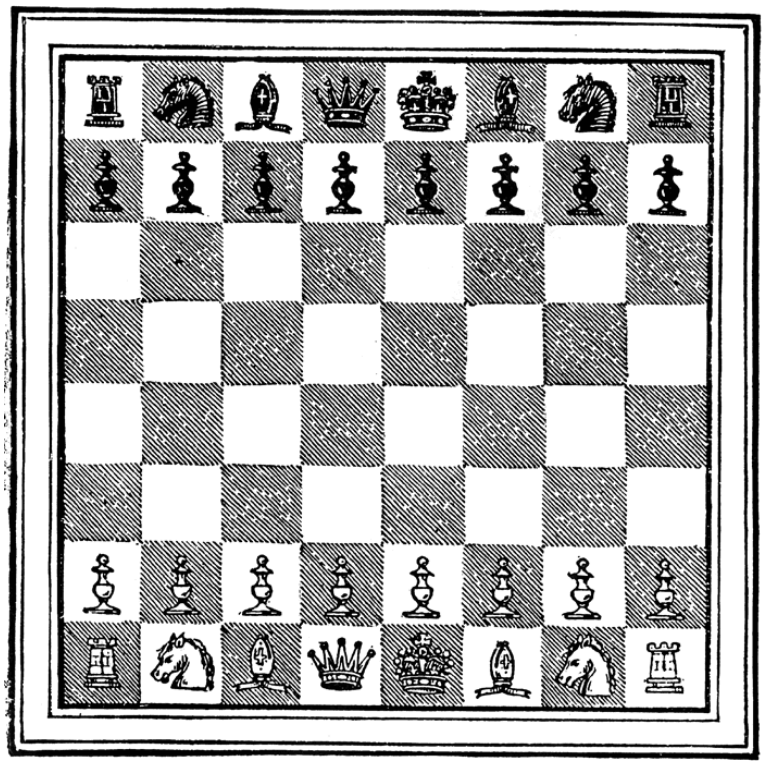
Syntax	Result
foo = {}	Empty dictionary
foo['coffee'] = "good"	Single item dictionary
foo['decaf'] = "bad"	Foo now contains two items
foo['coffee']	"good"
decaf' in foo	True
foo.keys()	['coffee', 'decaf']
foo['tea']	KeyError!
foo['tasty'] = ['cookies', 'ice-cream']	Note that dictionary keys may reference arbitrary objects

Discussion:

Why can you not sort a dictionary in place?
What would you do if you had to sort one?

Dictionary basics: sparse data structures

```
def main():
    board = {}
    # a chess board
    # say we only want to
    keep track of the
    # position of the kings
    board[(0, 4)] =
    "LIGHT_KING"
```



Discussion:

Can you compare the memory usage of this representation against a matrix using lists? (Chess boards are 8x8)

- Ignore the size of the dictionary and list support code (becomes insignificant at large sizes)

Basic dictionaries as records

```
import random
homer[‘donutSupply’] = 5
homer['hairsRemaining‘] = 4
homer['noises'] = [‘Munch’, ‘Crunch’, ‘MMM Donut’, ‘Aghggh’]
while homer['donutSupply'] > 0:
    homer['donutSupply'] -= 1
    print random.choice(homer['noises'])

→
MMM Donut
Aghggh
Aghggh
Munch
Crunch
```

Tuples in Brief

- Tuples use many of the same operations as lists and dictionaries
- Tuples are:
  - Ordered collections
  - Accessed by offset
  - Static and immutable
  - Fixed length
  - Heterogeneous
  - Nestable
- Why tuples?
  - Program integrity
  - Like constants, they're guaranteed to be consistent

Syntax	Result
()	An empty tuple
foo = ('homer', 'marge', 42)	A three item tuple
foo[1]	'Marge]
len(Foo)	3
foo[-1]	42
foo[:-1]	['homer', 'marge'] (give me everything up until the last element)

Control flow basics

Comparisons and Booleans

- Parenthesis are optional
- The end of line is the end of statement
- The end of indentation is the end of a block
- Why indentation syntax?
  - Enforces consistency and readability

C++:

```
if (x > y){
    cout << “x is larger”;
}else {
    cout << “x is smaller”;
}
```

Python:

```
if x > y:
    print “x is larger”
else:
    print “x is smaller”
```

Comparisons and Booleans Continued

```
C, C++, Java:
If (x)
   if (y)
        statement1;
else
    statement2;
```

- The dangling else problem:
  - Which statement does the else belong to?
- This problem doesn't occur in Python
- Note: although you may be smart enough not to write code this way, others on your project may not be

Comparisons and Booleans Continued

Syntax	Result
foo = True	A new boolean variable
5 > 5	False
5 is 5, 5 == 4	True
5 is not 5	False
‘x’ is “x”	True
5 >= 4	True
(True and True)	True
True and (5 > 4)	True
True and (“a” is “b”)	False
True or False	True

Python:

```
If x:
    if y:
        statement1
    else:
        statement2
```

Control flow: for statements

For loops are valid over any iteratble (strings, lists, tuples)

```
for x in range(5):
    print x
->
0
1
2
3
4
```

```
Simpsons = [“Homer”, “Marge”,
                “Lisa”, “Bart”]
for person in Simpsons:
    print person
->
Homer
Marge
Lisa
Bart
```

Control flow: nested for statements

- Loops (like all statements) can be nested
- For loops are valid over any iterable sequence

```
i = 0
for x in range(3):
    for y in range(3):
        i = i + 1
        #or i += 1
print i
```

Discussion:

What will this print? Take a moment and work it out.

Control flow: while statements

General form:

While <test>: #repeat while true

- <statements>

Else: #optional

- <statements>

```
a = 0
b = 10
while a < b:
    a += 1
    print a
```

Control flow: break and continue

- break:
  - exits the loop immediately
- continue:
  - returns to the beginning of the loop

```
a = 0
b = 100
while a < b:
    a += 1
    if a % 2 == 0:
        continue
    b = b / 2
    print a
     if b < 10:
        break
```

Discussion:

What will this print? Take a moment and work it out.

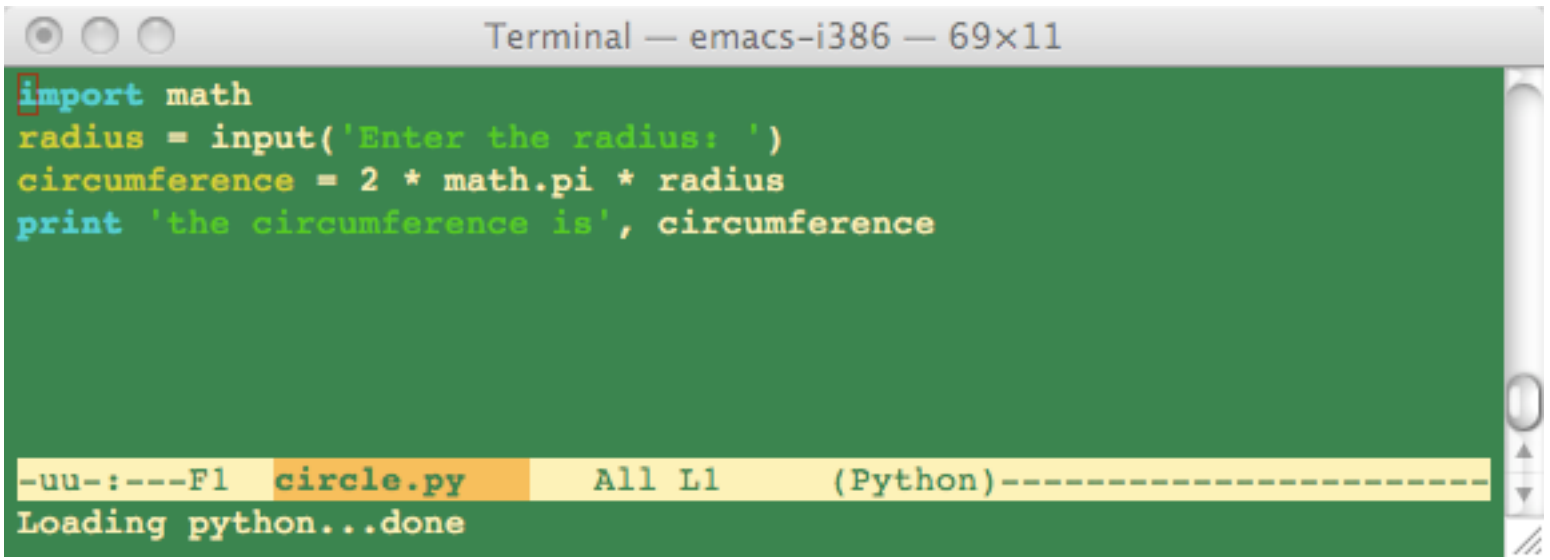
Preview: list comprehension

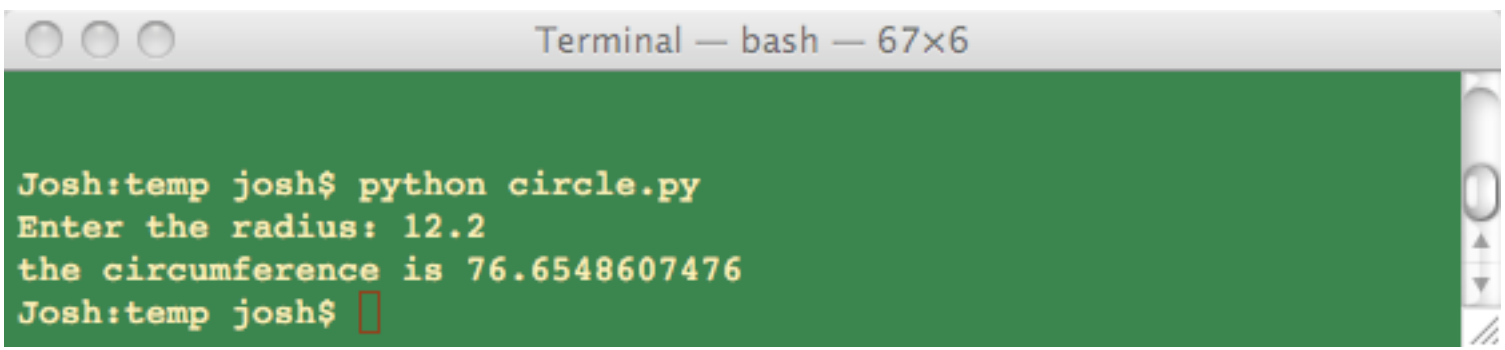
- List comprehension will likely be new to you if you're familiar with C or Java
  - Derived from set theory
- A shortcut to construct a new list object
  - Also a performance boost due to Pythons internals
- We'll see these used more later

```
foo = [1, 2, 3]
bar = [x + 10 for x in
   foo]
→
bar is now [11, 12, 13]

#Identical to:

bar = []
for x in foo:
   bar.append(x + 10)
```

User input from the console basics

Reading from the console





Type casting

- What happens if you try to square a number the user enters?
- Correction
  - print int(userSays) ** 2
- Casts work the other way too
  - x = "hi" + 5 + " times“
  - x = "hi" + str(5) + " times“
- You may cast between complicated structures as well – but be careful you understand the expected behavior in advance

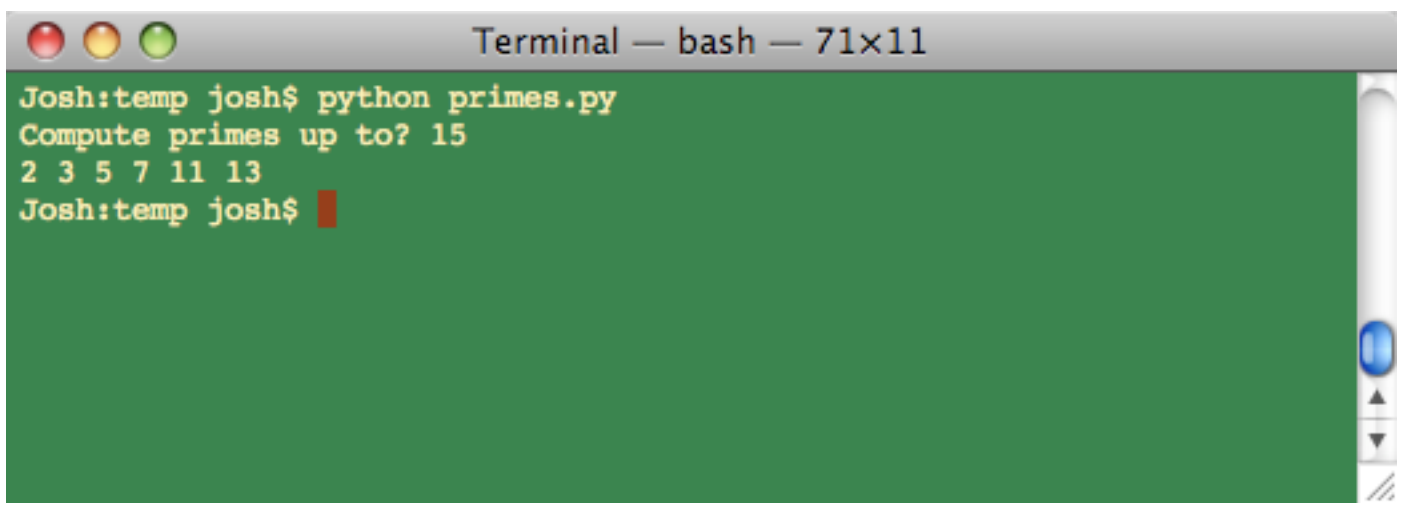
```
while True:
    x = input(‘Square: ')
    if (x == 'quit'):
        break
    else:
        print x ** 2

→
5
TypeError: unsupported operand for ‘str’
```

Assignment 1

Due by the next lecture

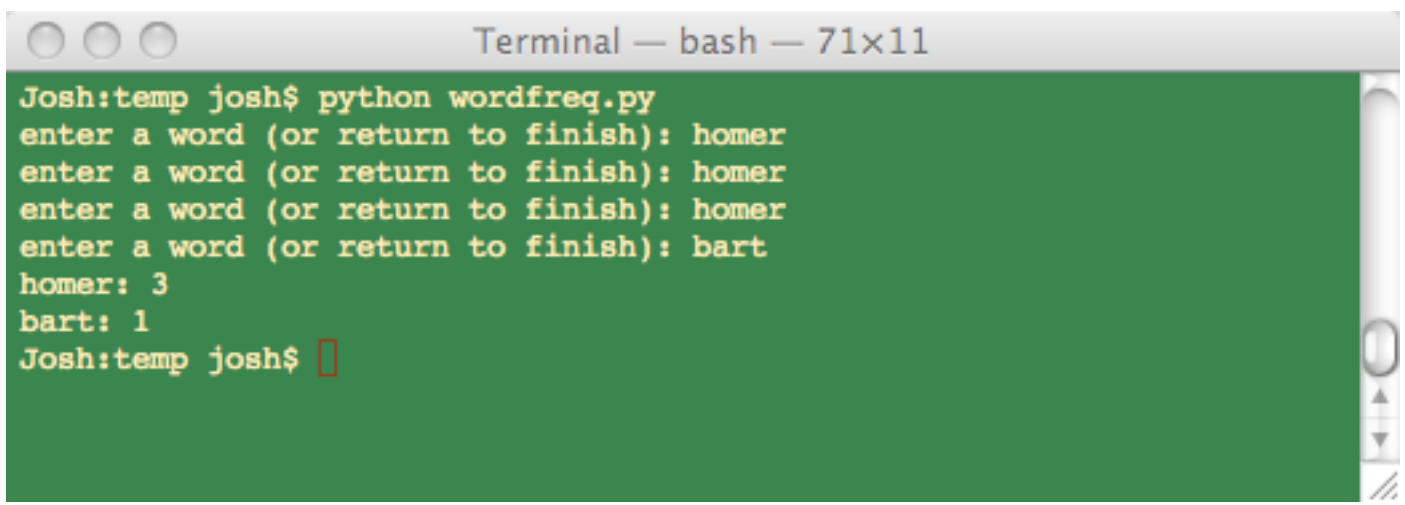
Exercise 1 of 2
Primes

- Write a script to compute and print the prime numbers below N. Read N from the console by prompting the user.
- Performance is not important, you'll be graded on correctness only.



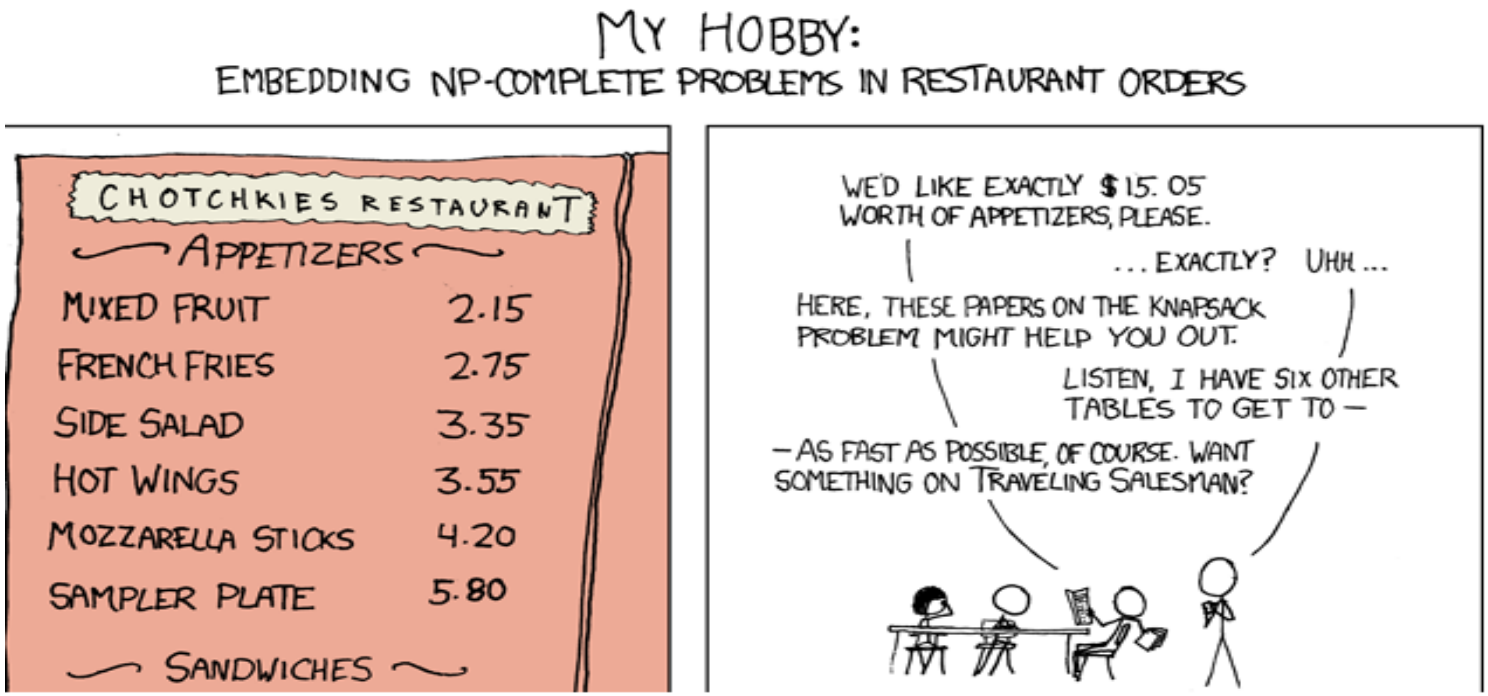
Exercise 2 of 2
Word frequency

- Write a word frequency counter. Your program should prompt the user to enter a single word per line (or return to finish). Report the frequency of each word, sorted in ascending order.
- Hint: Use a dictionary. When the user enters a word, check if the dictionary has that key.



Challenge problem

- A reasonable submission excuses you from the rest of the weeks homework. For the ambitious only, implement a solution to the following comic (a variant of the knapsack problem).



Source http://xkcd.com/287

See you next week

- Please remember to check the course website
  - www.cs.columbia.edu/~joshua/teaching/cs3101/cvn
- Questions: joshua@cs.columbia.edu