Modules and Decomposition

UW CSE 190p

Summer 2012

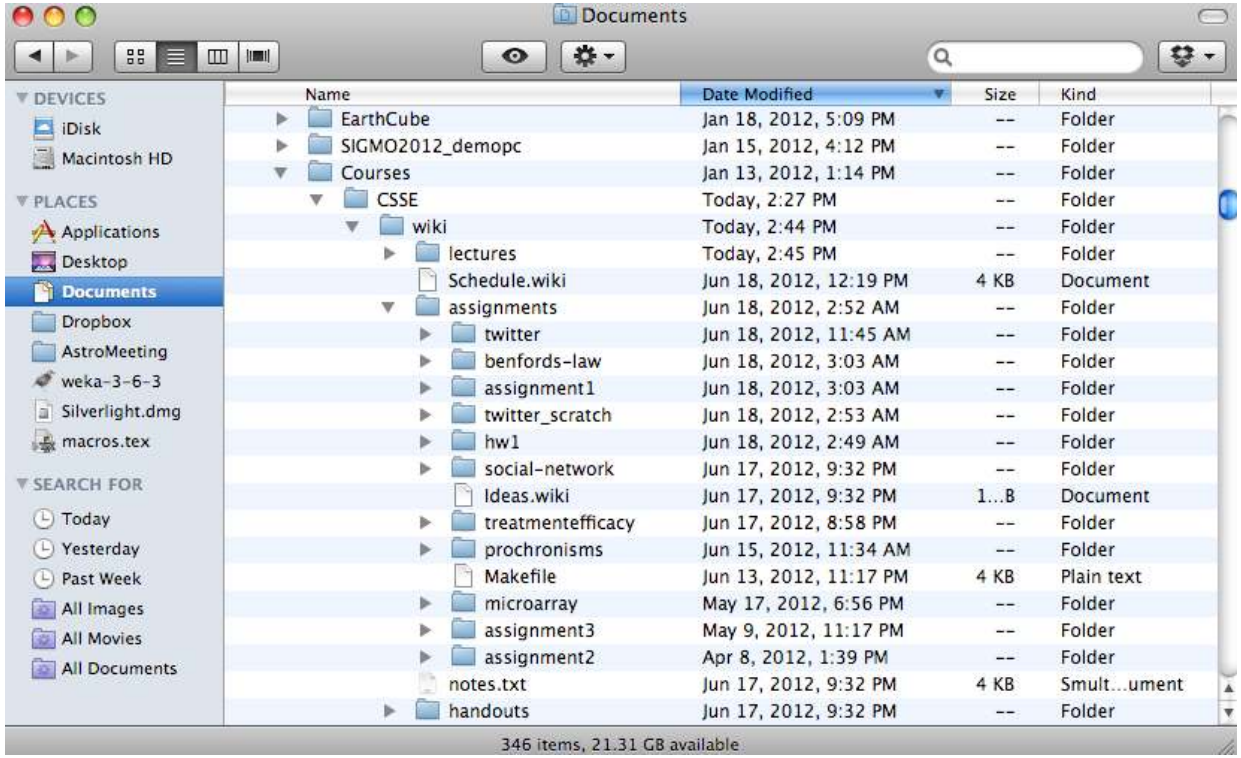
download examples from the calendar

http://c2.com/cgi/wiki?PythonPhilosophy
by Tim Peters

1. Beautiful is better than ugly.
2. Explicit is better than implicit.
3. Simple is better than complex.
4. Complex is better than complicated.
5. Flat is better than nested.
6. Sparse is better than dense.
7. Readability counts.
8. Special cases aren't special enough to break the rules.
    1. Although practicality beats purity.
9. Errors should never pass silently.
    1. Unless explicitly silenced.
10. In the face of ambiguity, refuse the temptation to guess.
11. There should be one-- and preferably only one --obvious way to do it.
    1. Although that way may not be obvious at first unless you're Dutch.
12. Now is better than never.
    1. Although never is often better than **right** now.
13. If the implementation is hard to explain, it's a bad idea.
14. If the implementation is easy to explain, it may be a good idea.
15. NameSpaces are one honking great idea -- let's do more of those!

Namespaces

- A container for a set of identifiers used to disambiguate duplicate names
- Example:



Modules in Python

Working definition for our purposes: a set of related functions, stored together in a python file

```
import math
import sys
import networkx as nx
```

module name

alias

Two motivations

1) provides a *namespace*

2) provides a *unit of abstraction*

```
from networkx import Graph, DiGraph

g = Graph()
```

Graph and DiGraph are now available in the global namespace

Writing a Module

```
def search(query):
  “”“return a list of tweets associated with the given query””“
  ...
```

Step 1: Write your definitions in a file named “twitter.py”
Step 2: Use your new module by writing

```
import twitter
```

That’s it!

The dir() function

- You can inspect the functions in a module using the dir function.

```
>>> dir(math)
['__doc__', '__file__', '__name__', '__package__', 'acos', 'acosh', 'asin',
'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e',
'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum',
'gamma', 'hypot', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p',
'modf', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'trunc']
```

This actually works on any object.

The dir() function

```
>>> import tweet
>>> dir(tweet)
['__builtins__', '__doc__', '__file__', '__name__', '__package__', '_makeurl',
'json', 'search']
```

Writing Modules (2)

- What about all the code not inside a function?
  - test cases, etc.
  - It would be nice if we had a way to determine whether we were running as a module or as a script!

```
print __name__
```

Modules: Public and Private Functions

Some of your functions may not be intended for public use.

Possible example: The read_scores function in homework 5

In Python, and unlike many languages, there's no guaranteed way to protect them.

There is a convention you can use:

```
def _read_scores(filename):
    """return the words and scores contained in filename"""
```

A leading underscore indicates "this function is intended for internal use only," but has no real effect.

What happens on import

```
import twitter
import twitter
```

1. (line 1) Python checks to see if twitter is already imported as a module. It isn't.

2. (line 1) Python looks for a file named twitter.py in the *search path.*

   Aside: The search path can be accessed and modified using the sys module

3. (line 1) If the file is found, the code in the file is executed as usual. The variable twitter is assigned a *module object.*

4. (line 2) Python checks to see if if twitter is already imported as a module. It is, so <u>nothing happens</u>.

Python does not re-read the file, even if it has changed!