Data Abstraction

UW CSE 140

Winter 2013

What is a program?

- What is a program?
  - A sequence of instructions to achieve some particular purpose
- What is a library?
  - A collection of routines that are helpful in multiple programs
- What is a data structure?
  - A representation of data, and
  - Routines to manipulate the data
    - Create, query, modify

Why break a program into parts?

- Easier to understand each part
  - Abstraction: When using a part, understand only its specification (documentation string); ignore its implementation
- Easier to test each part
- Reuse parts

Breaking a program into parts: the parts, and how to express them

Organizing the program & algorithm:

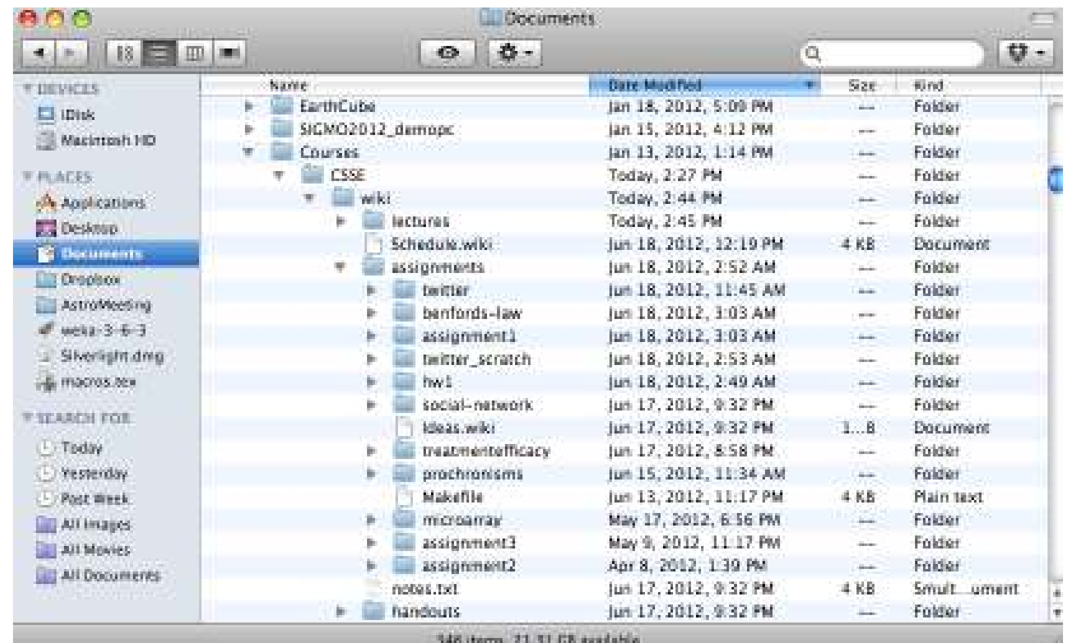
- Function (procedure)
- Library (collection of useful functions)
- Data structure (representation + methods)

Organizing the code (related but not the same!):

- Files
- Modules
- Namespaces

Namespace

- Disambiguates duplicate variable names
- Examples:
  - **`math.sin`**
  - File system directories



Review:
Accessing variables in a namespace

```
import math
... math.sin ...
```

```
import networkx as nx
```

module name — alias

```
g = nx.Graph()
```

```
from networkx import Graph, DiGraph
```

Graph and DiGraph are now available in the global namespace

```
g = Graph()
```

Recall the design exercise

- We created a module or library: a set of related functions
- The functions operated on the same data structure
  - a dictionary associating words with a frequency count
  - a list of tuples of measurements
- Each module contained:
  - A function to create the data structure
  - Functions to query the data structure
  - We could have added functions to modify the data structure

Two types of abstraction

Abstraction: Ignoring/hiding some aspects of a thing

- In programming, ignore everything except the specification or interface
- The program designer decides which details to hide and to expose

Procedural abstraction:

- Define a procedure/function specification
- Hide implementation details

Data abstraction:

- Define what the datatype represents
- Define how to create, query, and modify
- Hide implementation details of representation and of operations
  - Also called “encapsulation” or “information hiding”

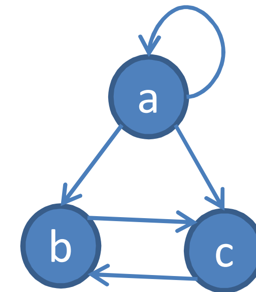
Data abstraction

- Describing field measurements:
  - "A dictionary mapping strings to lists, where the strings are sites and each list has the same length and its elements corresponds to the fields in the data file."
  - "FieldMeasurements"
- Which do you prefer? Why?

(This must appear in the documentation string of every function related to field measurements!)

Representing a graph

- A graph consists of:
  - nodes/vertices
  - edges among the nodes



- Representations:
  - Set of edge pairs
    - (a, a), (a, b), (a, c), (b, c), (c, b)
  - For each node, a list of neighbors
    - { a: [a, b, c], b: [c], c: [b] }
  - Matrix with boolean for each entry

	a	b	c
a	✓	✓	✓
b			✓
c		✓	

Text analysis module

(group of related functions)

representation = dictionary

```
# program to compute top 10:
wordcounts = read_words(filename)
result = topk(wordcounts, 10)
```

```
def read_words(filename):
    """Return a dictionary mapping each word in filename to its
frequency"""
    words = open(filename).read().split()
    wordcounts = {}
    for w in words:
      wordcounts.setdefault(w, 0) # set wordcounts[w] to 0 if not set
      wordcounts[w] += 1
    return wordcounts

def wordcount(wordcounts, word):
    """Given a frequency dictionary, return the count of the given
word"""
    return wordcounts[word]

def topk(wordcounts, k=10):
    """Given a frequency dictionary, return the top k most frequent
words, in order"""
    scores_with_words = [(c,w) for (w,c) in wordcounts.items()]
    scores_with_words.sort()
    return scores_with_words[0:k]

def totalwords(wordcounts):
    """Return the total number of words in the file"""
    return sum([s for (w,s) in wordcounts])
```

Problems with the implementation

```
# program to compute top 10:
wordcounts = read_words(filename)
result = topk(wordcounts, 10)
```

- The wordcount dictionary is exposed to the client: the user might corrupt or misuse it.
- If we change our implementation (say, to use a list), it may break the client program.

We prefer to

- Hide the implementation details from the client
- Collect the data and functions together into one unit

Datatypes and classes

- A class defines a data type
- A class creates a namespace for:
  - Variables to hold the data
  - Functions to create, query, and modify
    - Each function defined in the class is called a method
      - Takes “**`self`**” (a value of the class type) as the first argument
- A class defines a datatype
  - An object is a value of that type
  - Compare to **`int`** vs. 22

Recall the text analysis implementation

```
# program to compute top 10:
wordcounts = read_words(filename)
result = topk(wordcounts, 10)
```

```
def read_words(filename):
  """Populate a WordCounts object from the given file"""
  words = open(filename).read().split()
  wordcounts = {}
  for w in words:
    wordcounts.setdefault(w, 0)
    wordcounts[w] += 1

def wordcount(wordcounts, word):
  """Return the count of the given word"""
  return wordcounts[word]

def topk(wordcounts, k=10):
  """Return a list of the top k most frequent words in order"""
  scores_with_words = [(c,w) for (w,c) in wordcounts.items()]
  scores_with_words.sort()
  return scores_with_words[0:k]

def totalwords(wordcounts):
  """Return the total number of words in the file"""
  return sum([s for (w,s) in wordcounts])
```

Text analysis, as a class

The type of **wc** is **WordCounts**

```
# program to compute top 10:
wc = WordCounts()
wc.read_words(filename)
result = wc.topk(10)
```

topk takes 2 arguments

```
class WordCounts:
  """Represents the words in a file."""
  # Internal representation:
  # variable wordcounts is a dictionary from each word to its frequency

  def read_words(self, filename):
    """Populate a WordCounts object from the given fi
    words = open(filename).read().split()
    self.wordcounts = {}
    for w in words:
      self.wordcounts.setdefault(w, 0)
      self.wordcounts[w] += 1

  def wordcount(self, word):
    """Return the count of the given word"""
    return self.wordcounts[word]

  def topk(self, k=10):
    """Return a list of the top k most frequent words in order"""
    scores_with_words = [(c,w) for (w,c) in self.wordcounts.items()]
    scores_with_words.sort()
    return scores_with_words[0:k]

  def totalwords(self):
                  otal number of words in the file"""
                  for (w,s) in self.wordcounts])
```

Defines a class (a datatype) named WordCounts

The type of **self** is **WordCounts**

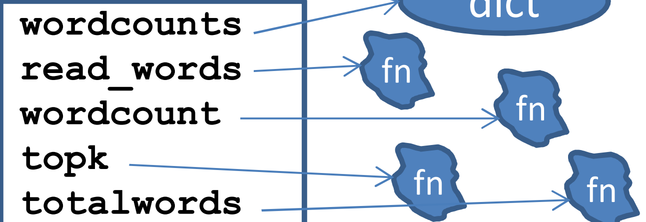
Modifies a WordCounts object

read_words does not return a value; it mutates **self**

Queries a WordCounts object

Each function in a class is called a *method*.
Its first argument is of the type of the class.

The namespace of a WordCounts object:



```
# program to compute top 10:

wc = WordCounts()
wc.read_words(filename)

result = wc.topk(10)

result = WordCounts.topk(wc, 10)
```

Weird constructor: it does no work

You have to call a mutator immediately afterward

A value of type WordCounts

A namespace, like a module

A function that takes two arguments

Two equivalent calls

Class with constructor

```
# program to compute top 10:
wc = WordCounts(filename)
result = wc.topk(10)
```

```
class WordCounts:
  """Represents the words in a file."""
  # Internal representation:
  # variable wordcounts is a dictionary from words to their frequency

  def __init__(self, filename):
    """Create a WordCounts object from the given file"""
    words = open(filename).read().split()
    self.wordcounts = {}
    for w in words:
      self.wordcounts.setdefault(w, 0)
      self.wordcounts[w] += 1

  def wordcount(self, word):
    """Return the count of the given word"""
    return self.wordcounts[word]

  def topk(self, k=10):
    """Return a list of the top k most frequent words in order"""
    scores_with_words = [(c,w) for (w,c) in self.wordcounts.items()]
    scores_with_words.sort()
    return scores_with_words[0:k]

  def totalwords(self):
    """Return the total number of words in the file"""
    return sum([s for (w,s) in self.wordcounts])
```

Alternate implementation

```
# program to compute top 10:
wc = WordCounts(filename)
result = wc.topk(10)
```

Exact same program!

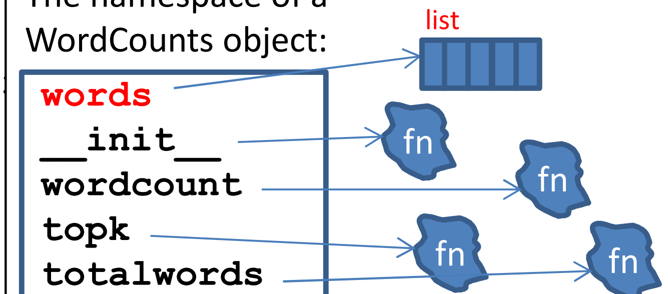
```
class WordCounts:
  """Represents the words in a file."""
  # Internal representation:
  # variable words is a list of the words in the file

  def __init__(self, filename):
    """Create a WordCounts object from the given file"""
    self.words = open(filename).read().split()

  def wordcount(self, word):
    """Return the count of the given word"""
    return self.words.count(word)

  def topk(self, k=10):
    """Return a list of the top k most frequent words in order"""
    scores_with_words = [(wordcount(w),w) for w in set(self.words)]
    scores_with_words.sort()
    return scores_with_words[0:k]

  def totalwords(self):
    """Return the total number of words in the
    return len(self.words)
```



Quantitative analysis

```
# Program to plot
mydict = read_measurements(filename)
result = mydict.Stplot()
```

```
def read_measurements(filename):
  """Return a dictionary mapping column names to data.
Assumes the first line of the file is column names."""
  datafile = open(filename)
  rawcolumns = zip(*[row.split() for row in datafile])
  columns = dict([(col[0], col[1:]) for col in rawcolumn
  return columns

def tofloat(measurements, columnname):
  """Convert each value in the given iterable to a float"""
  return [float(x) for x in measurements[columnname]]

def STplot(measurements):
  """Generate a scatter plot comparing salinity and temperature"""
  xs = tofloat(measurements, "salt")
  ys = tofloat(measurements, "temp")
  plt.plot(xs, ys)
  plt.show()

def minimumO2(measurements):
  """Return the minimum value of the oxygen measurement"""
  return min(tofloat(measurements, "o2"))
```

Quantitative analysis, as a class

```
# Program to plot
mm = Measurements()
mm.read_measurements(filename)
result = mm.Stplot()
```

```
class Measurements:
  """Represents a set of measurements in UWFORMAT."""

  def read_measurements(self, filename):
    """Populate a Measurements object from the given file.
Assumes the first line of the file is column names."""
    datafile = open(filename)
    rawcolumns = zip(*[row.split() for row in datafile])
    self.columns = dict([(col[0], col[1:]) for col in rawcolumn
    return columns

  def tofloat(self, columnname):
    """Convert each value in the given iterable to a float"""
    return [float(x) for x in self.columns[columnname]]

  def STplot(self):
    """Generate a scatter plot comparing salinity and temperature"""
    xs = tofloat(self.columns, "salt")
    ys = tofloat(self.columns, "temp")
    plt.plot(xs, ys)
    plt.show()

  def minimumO2(self):
    """Return the minimum value of the oxygen measurement"""
    return min(tofloat(self.columns, "o2"))
```

Quantitative analysis, with a constructor

```
# Program to plot
mm = Measurements(filename)
result = mm.Stplot()
```

```
class Measurements:
  """Represents a set of measurements in UWFORMAT."""

  def __init__(self, filename):
    """Create a Measurements object from the given file.
Assumes the first line of the file is column names."""
    datafile = open(filename)
    rawcolumns = zip(*[row.split() for row in datafile])
    self.columns = dict([(col[0], col[1:]) for col in rawcolumn


  def tofloat(self, columnname):
    """Convert each value in the given iterable to a float"""
    return [float(x) for x in self.columns[columnname]]

  def STplot(self):
    """Generate a scatter plot comparing salinity and temperature"""
    xs = tofloat(self.columns, "salt")
    ys = tofloat(self.columns, "temp")
    plt.plot(xs, ys)
    plt.show()

  def minimumO2(self):
    """Return the minimum value of the oxygen measurement"""
    return min(tofloat(self.columns, "o2"))
```