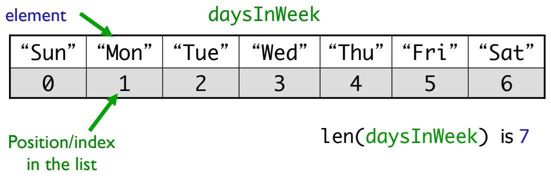
Objectives

- Introduction to Files

Review

- What is a list?
- How do we specify a list?
- How are lists and strings similar?
- How are they different?
  - What are the implications of those differences?

Lists: A Sequence of Data Elements



daysInWeek

“Sun”	“Mon”	“Tue”	“Wed”	“Thu”	“Fri”	“Sat”
0	1	2	3	4	5	6

`len(daysInWeek)` is 7

- Elements in lists can be *any* data type
- Operations similar to strings

str Method Flashback

- `string.split([sep])`
  - Returns a list of the words in the string `string`, using `sep` as the delimiter string
  - If `sep` is not specified or is `None`, any whitespace (space, new line, tab, etc.) is a separator
  - Example:

```
phrase = “Hello, Computational Thinkers!”
x = phrase.split()
```

What is x? Its data type? What does x contain?

str Method Flashback

- `string.join(iterable)`
  - Return a string which is the concatenation of the *strings* in the iterable. The separator between elements is S.
  - Example:

```
x = ["1","2","3"]
phrase = " ".join(x)
```

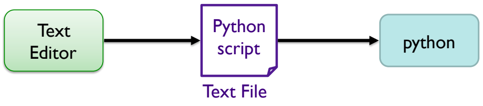
What is x? Its data type?
What is phrase’s data type?
What does phrase contain?

Sources of Input to Program

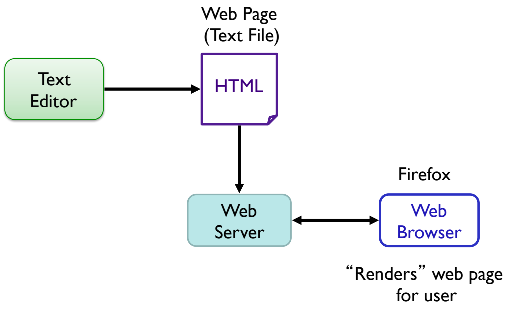
- User input
  - Slow if need to enter a lot of data
  - Error-prone
    - User enters the wrong value!
  - What if want to run again after program gets modified?

Sources of Input to Program

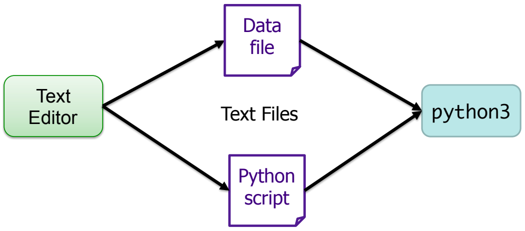
- Text files
  - Enter data once into a file, save it, and reuse it
  - Good for large amounts of data
  - Programs can use files to *communicate*
  - Need to be able to *read from* and *write to* files



Example Use of Files



Example Use of Text File as Input



Updated Wheel of Fortune

- Uses a file of puzzles
  - Puzzles no longer appear directly in program
  - Can modify puzzle file to get different puzzles

```
displayedPuzzle = ""
for char in puzzle:
   if char.isalpha():
      displayedPuzzle += "_"
   else:
      displayedPuzzle += char
```

Files

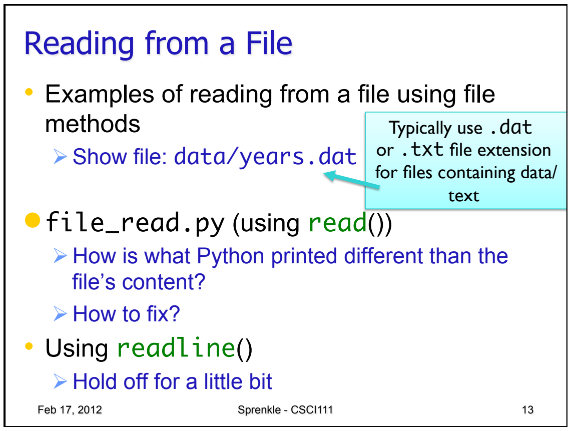
- Conceptually, a file is a **sequence** of data stored in memory
- To use a file in a Python script, create an object of type `file`
  - `file` is a *data type*

Built-in function "constructs" a `file` object

  - `<varname> = open(<filename>,<mode>)`
    - `<filename>`: `string`
    - `<mode>`: `string`, "r" for read, "w" for write, "a" for append (and others)
  - Ex: `dataFile = open( "years.dat", "r" )`

Common File Methods

Method Name	Functionality
`read()`	Read the entire content from the file, returned as a string object
`readline()`	Read one line from file, returned as a string object (which includes the "\n"). If it returns "", then you've reached the end of the file
`write(string)`	Write a string to the file
`close()`	Close the file. Must close the file after done reading from/writing to a file

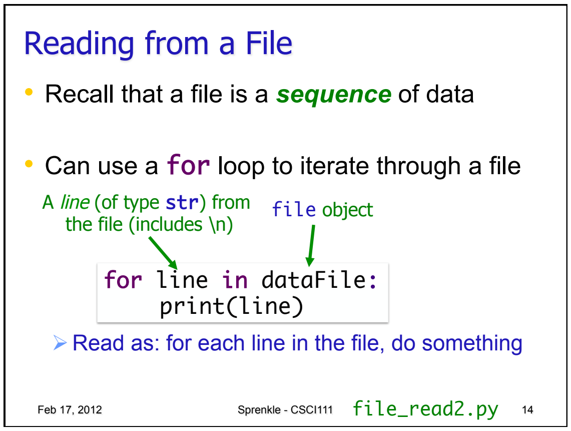


Reading from a File

- Examples of reading from a file using file methods
  - Show file: `data/years.dat`

Typically use `.dat` or `.txt` file extension for files containing data/text

- `file_read.py` (using `read()`)
  - How is what Python printed different than the file's content?
  - How to fix?
- Using `readline()`
  - Hold off for a little bit



Reading from a File

- Recall that a file is a ***sequence*** of data

- Can use a **for** loop to iterate through a file

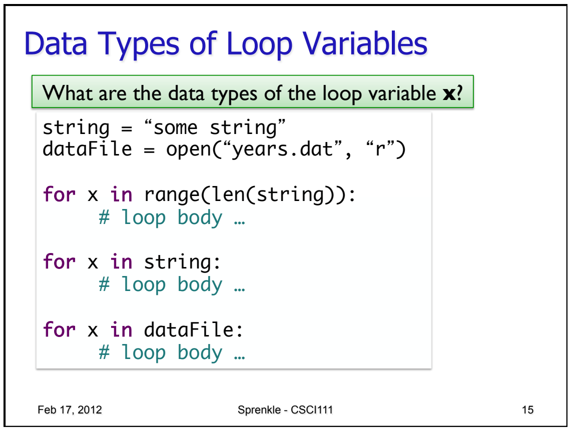
A *line* (of type **str**) from the file (includes \n)

`file` object

```
for line in dataFile:
    print(line)
```

  - Read as: for each line in the file, do something

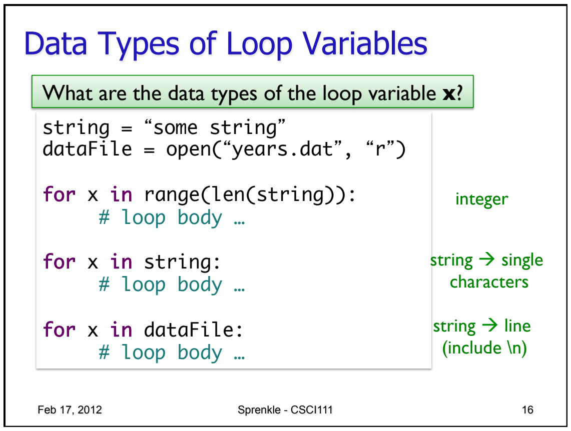
`file_read2.py`



Data Types of Loop Variables

What are the data types of the loop variable **x**?

```
string = "some string"
dataFile = open("years.dat", "r")

for x in range(len(string)):
     # loop body …

for x in string:
     # loop body …

for x in dataFile:
     # loop body …
```



Data Types of Loop Variables

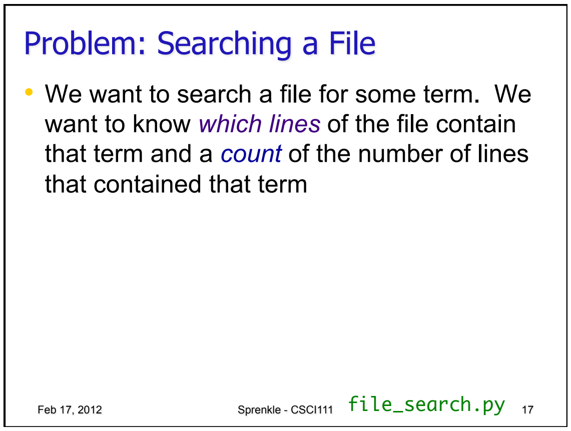
What are the data types of the loop variable **x**?

```
string = "some string"
dataFile = open("years.dat", "r")

for x in range(len(string)):
     # loop body …

for x in string:
     # loop body …

for x in dataFile:
     # loop body …
```

integer

string → single characters

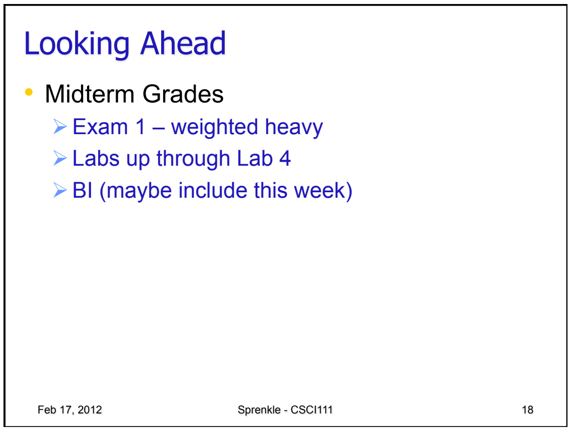
string → line (include \n)



Problem: Searching a File

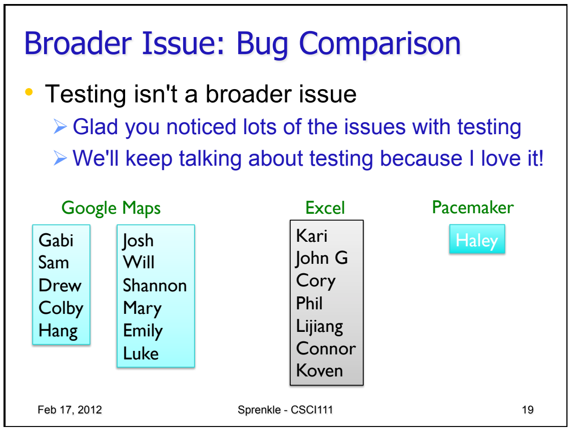
- We want to search a file for some term. We want to know *which lines* of the file contain that term and a *count* of the number of lines that contained that term

`file_search.py`



Looking Ahead

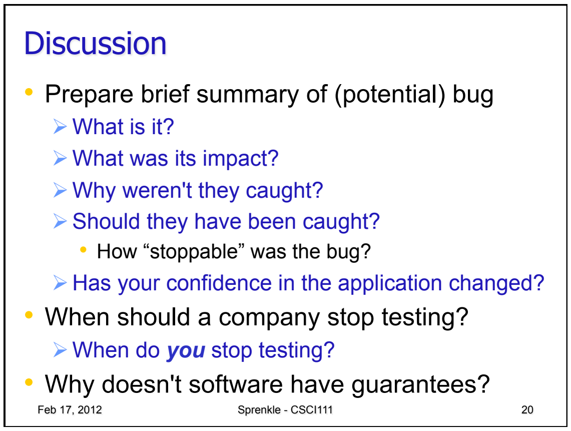
- Midterm Grades
  - Exam 1 – weighted heavy
  - Labs up through Lab 4
  - BI (maybe include this week)



Broader Issue: Bug Comparison

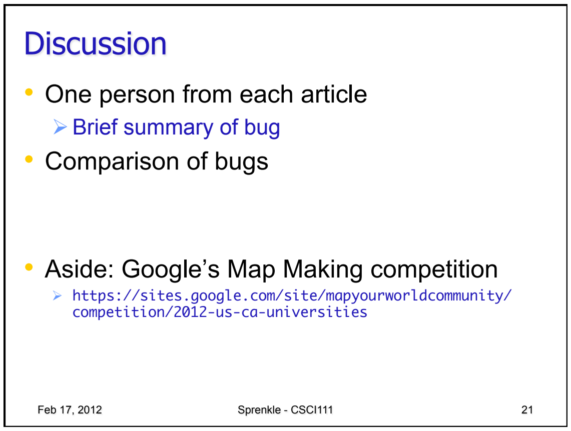
- Testing isn't a broader issue
  - Glad you noticed lots of the issues with testing
  - We'll keep talking about testing because I love it!

Google Maps		Excel	Pacemaker
Gabi	Josh	Kari	Haley
Sam	Will	John G	
Drew	Shannon	Cory	
Colby	Mary	Phil	
Hang	Emily	Lijiang	
	Luke	Connor	
		Koven	



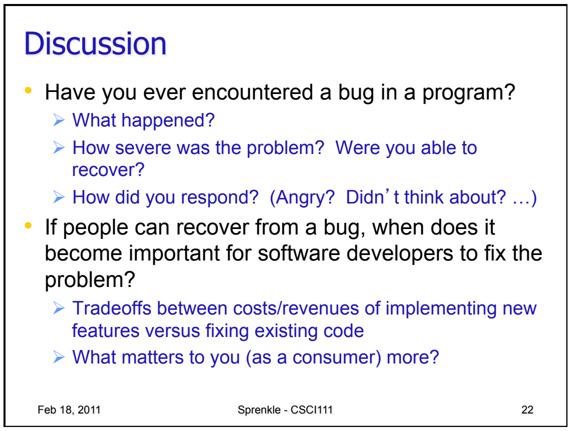
Discussion

- Prepare brief summary of (potential) bug
  - What is it?
  - What was its impact?
  - Why weren't they caught?
  - Should they have been caught?
    - How “stoppable” was the bug?
  - Has your confidence in the application changed?
- When should a company stop testing?
  - When do ***you*** stop testing?
- Why doesn't software have guarantees?



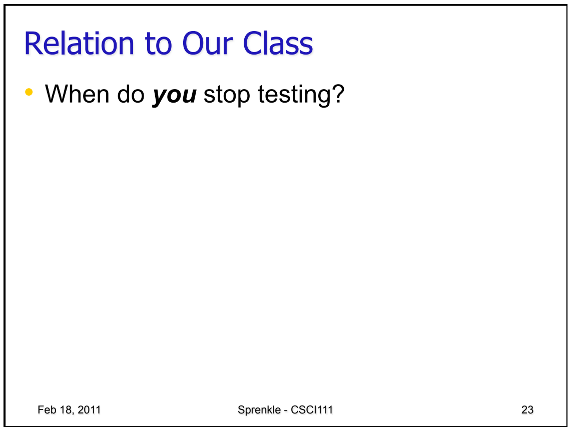
Discussion

- One person from each article
  - Brief summary of bug
- Comparison of bugs
- Aside: Google’s Map Making competition
  - `https://sites.google.com/site/mapyourworldcommunity/competition/2012-us-ca-universities`



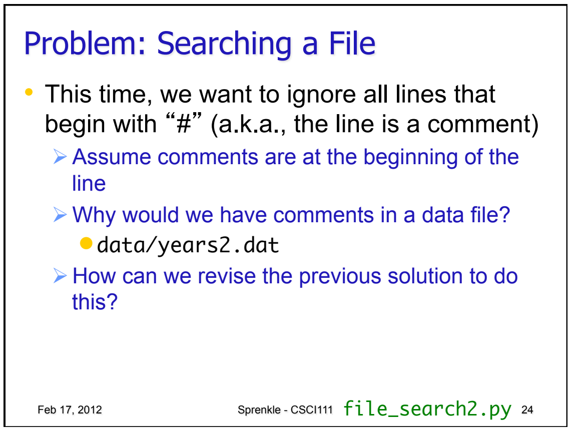
Discussion

- Have you ever encountered a bug in a program?
  - What happened?
  - How severe was the problem? Were you able to recover?
  - How did you respond? (Angry? Didn’t think about? …)
- If people can recover from a bug, when does it become important for software developers to fix the problem?
  - Tradeoffs between costs/revenues of implementing new features versus fixing existing code
  - What matters to you (as a consumer) more?



Relation to Our Class

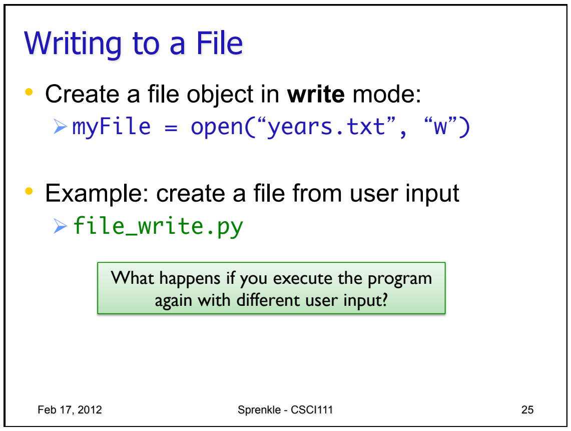
- When do ***you*** stop testing?



Problem: Searching a File

- This time, we want to ignore all lines that begin with “#” (a.k.a., the line is a comment)
  - Assume comments are at the beginning of the line
  - Why would we have comments in a data file?
    - `data/years2.dat`
  - How can we revise the previous solution to do this?

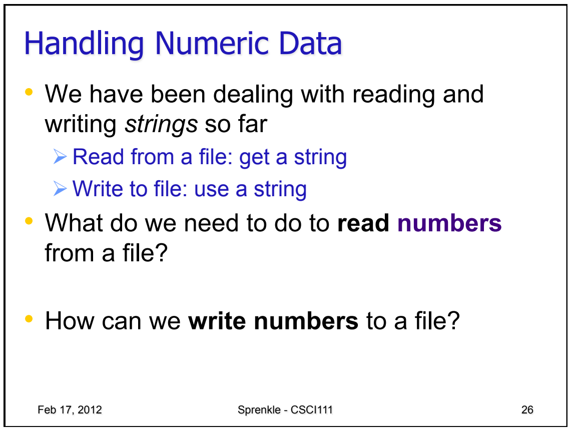
`file_search2.py`



Writing to a File

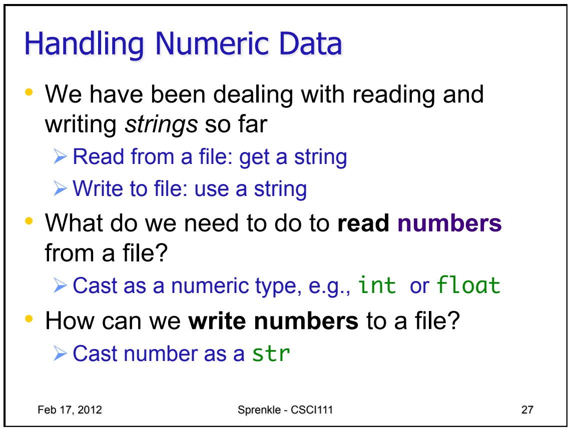
- Create a file object in **write** mode:
  - `myFile = open("years.txt", "w")`
- Example: create a file from user input
  - `file_write.py`

What happens if you execute the program again with different user input?



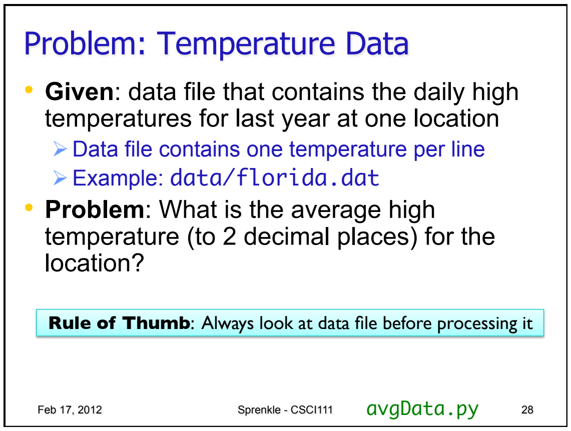
Handling Numeric Data

- We have been dealing with reading and writing *strings* so far
  - Read from a file: get a string
  - Write to file: use a string
- What do we need to do to **read numbers** from a file?
- How can we **write numbers** to a file?



Handling Numeric Data

- We have been dealing with reading and writing *strings* so far
  - Read from a file: get a string
  - Write to file: use a string
- What do we need to do to **read numbers** from a file?
  - Cast as a numeric type, e.g., `int` or `float`
- How can we **write numbers** to a file?
  - Cast number as a `str`

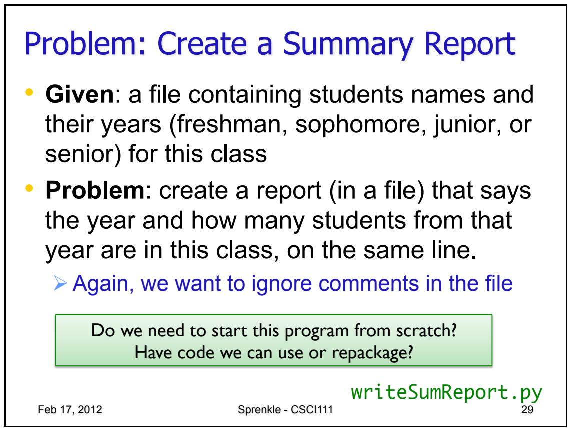


Problem: Temperature Data

- **Given**: data file that contains the daily high temperatures for last year at one location
  - Data file contains one temperature per line
  - Example: `data/florida.dat`
- **Problem**: What is the average high temperature (to 2 decimal places) for the location?

Rule of Thumb: Always look at data file before processing it

`avgData.py`

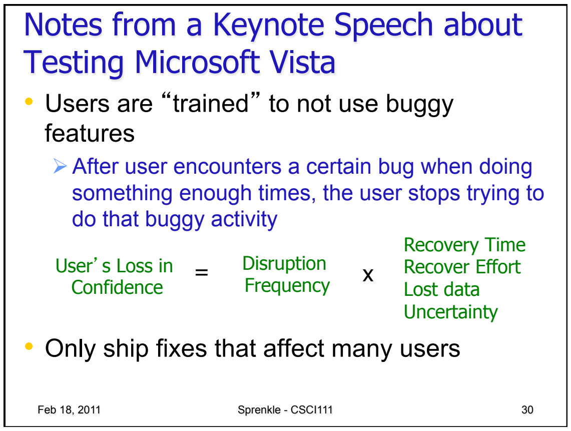


Problem: Create a Summary Report

- **Given**: a file containing students names and their years (freshman, sophomore, junior, or senior) for this class
- **Problem**: create a report (in a file) that says the year and how many students from that year are in this class, on the same line.
  - Again, we want to ignore comments in the file

Do we need to start this program from scratch?
Have code we can use or repackage?

`writeSumReport.py`



Notes from a Keynote Speech about Testing Microsoft Vista

- Users are "trained" to not use buggy features
  - After user encounters a certain bug when doing something enough times, the user stops trying to do that buggy activity

User's Loss in Confidence = Disruption Frequency x Recovery Time
Recover Effort
Lost data
Uncertainty

- Only ship fixes that affect many users

Status from Official Excel Blog

Happy Ending

- Post on 9/25/07
  - We've come up with a fix for this issue and are in the final phases of a broad test pass in order to ensure that the fix works and doesn't introduce any additional issues - especially any other calculation issues. This fix then needs to make its way through our official build lab and onto a download site - which we expect to happen very soon.
- Post on 10/9/07
  - As of today, fixes for this issue in Excel 2007 and Excel Services 2007 are available for download …
  - We are in the process of adding this fix to Microsoft Update so that it will get automatically pushed to users running Excel 2007 or Excel Services 2007. Additionally, the fix will also be contained in the first service pack of Office 2007 when it is released (the release date for SP1 of Office 2007 has not been finalized).

Broader Issue: Volunteer Computing

- What is the goal of the project/problem they are solving?
- Why are computer scientists involved with this problem/its solution?
- What is their solution to the problem?
  - What was their insight to the solution?
- What are some of the results of their solution?
- What are some issues they have had to solve?
- What are other problems that are being solved in similar ways?
- What other problems should we use volunteer computing to solve?
- How does involving the public in science change people's perception of science, if at all?
- How does this article relate to this class?

Discussion

- Problem: huge computational problems, huge data sets; limited computing resources
  - Supercomputers are expensive
- Insight: lots of computers that are often idle
  - Leverage these cheap resources to create a distributed super computer
- Can break up a huge problem into small pieces that can be solved separately
  - Merge solved pieces back together

Problems to Solve

- How to break up the problem, how to merge
  - Need *correct*, *efficient* solutions
- How do we distribute the problems?
- Lots of different OSs, types of machines
  - Process in platform-independent way
- How do we know we're getting the **right** answer?
- What if a volunteer gives unreliable results?
- How can we identify malicious behavior?
- How do we store all the results?

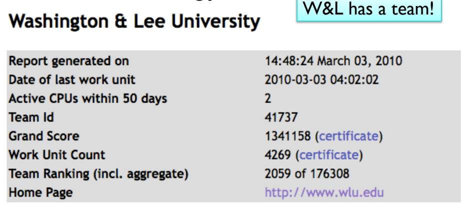
Computer science problems motivated by other domains!

Kismet: Folding@HOME

Why I chose this article…

- `http://folding.stanford.edu`
- Accurately simulate folding of proteins
- Results help understand diseases and fundamental biology

W&L has a team!



Washington & Lee University

Report generated on	14:48:24 March 03, 2010
Date of last work unit	2010-03-03 04:02:02
Active CPUs within 50 days	2
Team Id	41737
Grand Score	1341158 (certificate)
Work Unit Count	4269 (certificate)
Team Ranking (incl. aggregate)	2059 of 176308
Home Page	http://www.wlu.edu

Review: Functions

```
CONSTANT = 12

def main():
    first = input("Enter the first number: ")
    second = input("Enter the second number: ")
    computedVal = myFunction(first, second)
    print "The answer is", computedVal

def myFunction(x, y):
    result = x*x + y*y + CONSTANT
    return result

main()
```

Where does program start "doing stuff"?

What variables can function "see" here? What vars can't it see?

Review: Why Functions?

- Organize code
- Easier to read
- Easier to change
- Easier to reuse

Parts of an Algorithm

- Primitive operations
  - What data you have, what you can do to the data
- Naming
  - Identify things we're using
- Sequence of operations
- Conditionals
  - Handle special cases
- Repetition/Loops
- Subroutines
  - Call, reuse similar techniques

- Which of these have we covered?
- How do we implement them in Python?

Parts of an Algorithm

where most of the rest of the semester focuses

- Primitive operations
  - What **data** you have, what you **can do** to the data
- Naming
  - Identify things we're using
- Sequence of operations
- Conditionals
  - Handle special cases
- Repetition/Loops
- Subroutines
  - Call, reuse similar techniques

No longer *primitive*