Objectives

- Reading from files
  - Numbers!
- Writing to files

Review

- What is the major [implementation] difference between strings and lists?
  - What are the implications of that difference?

Review: Lists vs. Strings

- Strings are **immutable**
  - Can't be mutated?
  - Err, can't be modified/changed
- Lists are **mutable**
  - Can be changed
  - Changes how we call/use methods

Implications:
- Think of `list` variables as ***pointing*** to the list
- Assigning a list to another variable does ***not* make a copy** of the list
- list methods ***modify*** the list on which the method was called
  → Don't return a copy of the object, modified
- When you pass a list into a function, you ***can modify*** the list

Review

- Why should we care about files?
- How do we create a file object?
- How can we read from a file? (3 ways)

Review: 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*
  - `<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" )`

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

Common File Methods

Method Name	Functionality
`read()`	Read all the content from the file, returned as a string object
`readline()`	Read next 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

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

Writing to a File

- Create a file object in **write** mode:
  - `myFile = open("myfile.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` or use `format` method

Problem: Temperature Data

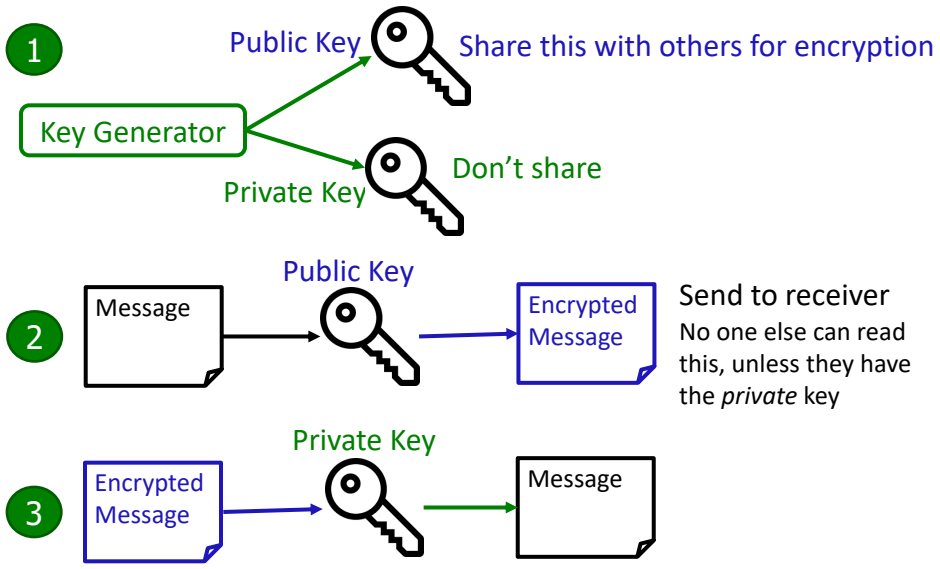
- **Given**: data file that contains the daily high temperatures for last year at one location
  - Data file contains one temperature per line
  - Example: `data/florida.dat`
- **Problem**: What is the average high temperature (to 2 decimal places) for the location?

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

Broader Issue: Cryptography

August Hayden Jake James Mike	Alice Andrew Danny Jenna Nate	Bobby Callie Cat Giovanni Laurie	Charlotte Danielle Karel Matt Natalie	Dan Ellis Kassi Melissa

Public Key Cryptography



Broader Issues Discussion

- What is cryptography?
  - Why is cryptography useful?
- Debate is often summarized as "privacy vs security"
  - What does this mean? How does it relate to the recent case (Apple vs FBI) referred to in the article?
- Why are computer scientists involved in politics?
  - How does that affect software development?
  - Has this class informed your politics?
- Who is Alan Turing?

Duffie and Hellman

- Technique still used in protocols (like SSL today)

Problem: Cleaning Up Data

- **Given**: a CSV file containing students' names and their class according to the Registrar
- **Problem**: This file has TMI
  - Just want the last name and the class year
  - Instead of Ugr:Sophomore, say "Sophomore"
- **Solution:**
  - Read through file "data/years.csv", clean up data
  - Write the cleaned up data to a new file called "data/roster.txt"
    - 1st iteration: lastname class
    - 2nd iteration: nice tables of data

`cleanRoster.py`

Problem: Create a Summary Report

- **Given**: a file containing students names and their years (first years, 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`

Senator Franken vs Facebook lawyer Stretch

- "People are buying ads on your platform with roubles. They're political ads. You put billions of data points together all the time. **That's what I hear that these platforms do: they're the most sophisticated things invented by man, ever.** Google has all knowledge that man has ever developed. You can't put together roubles with a political ad and go hmm, those two data points spell out something bad?"

Your Supervisor

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

M :3

Review: Functions

Where does program start "doing stuff"?

```
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()
```

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

March 8, 2019 Sprenkle - CSCI111 24

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

No longer *primitive*

- Sequence of operations
- Conditionals
  - Handle special cases
- Repetition/Loops
- Subroutines
  - Call, reuse similar techniques