Lab 8

- Feedback on Lab 7
- Review
  - Lists
  - Files
  - Modules

LAB 7 FEEDBACK

Review Caesar Cipher

- Consider the following (partial) solutions

```
for char in message:
    asciiVal = ord(char)
    if asciiVal == 32:
        …
    else:
        …
```

```
for char in message:
    if char == " ":
        …
    else:
        …
```

Which solution do you prefer?

Review Caesar Cipher

- Consider the following (partial) solutions

```
for char in message:
    asciiVal = ord(char)
    if asciiVal == 32:
        …
    else:
        …
```

I know what " " means.
I don't immediately know what 32 means.
Lesson: prefer words over numbers.

```
for char in message:
    if char == " ":
        …
    else:
        …
```

Comment Example

```python
def encryptLetter(letter, key):
   """
   Encrypts a single letter by the given key.
   Parameters:
    - letter: a single, lowercase character string
    - key: an integer (between -25 and 25, inclusive)
   Returns the encrypted character as a str
   """
```

- Focus on the *interface* – how to call function and what it does/returns
- Does not say *who* called function, where parameters came from, or where returned to
  - *Any* code can call the function and pass in input from anywhere (e.g., hardcoded, from user input, test function, …)
- Does not say variable name returned

Comment Example 2

```
def encryptLetter(letter, key):
   """Encrypts a single letter.
   PRE: Input parameters are a single, lowercase
   character string (letter) and an integer key
   (between -25 and 25, inclusive)
   POST: returns the encrypted character as a str"""
```

- Focus on the *interface* – how to call function and what it does/returns
- Does not say *who* called function, where parameters came from, or where returned to
  - *Any* code can call the function and pass in input from anywhere (e.g., hardcoded, from user input, test function, …)
- Does not say variable name returned
- **Format doesn't matter as much as containing required content**

Review

- What are things we can do with lists?
- How do we work with files?
  - What are things we can do with files?
- What is your algorithm for finding the average temperature in a file?
  - (Problem from handout)
- From a while back: What is a module?
  - What are the benefits of modules?
  - How do we create a module?
  - How do we use functions defined in a module?

Review: List Operations

Similar to operations for strings

Concatenation	`<seq> + <seq>`
Repetition	`<seq> * <int-expr>`
Indexing	`<seq>[<int-expr>]`
Length	`len(<seq>)`
Slicing	`<seq>[:]`
Iteration	`for <var> in <seq>:`
Membership	`<expr> in <seq>`

Review: List Methods

Method Name	Functionality
`<list>.append(`*x*`)`	Add element *x* to the end
`<list>.sort()`	Sort the list
`<list>.reverse()`	Reverse the list
`<list>.index(`*x*`)`	Returns the index of the first occurrence of *x*, Error if *x* is not in the list
`<list>.insert(`*i*`, `*x*`)`	Insert *x* into list at index *i*
`<list>.count(`*x*`)`	Returns the number of occurrences of *x* in list
`<list>.remove(`*x*`)`	Deletes the first occurrence of *x* in list
`<list>.pop(`*i*`)`	Deletes the *i* th element of the list and returns its value

Note: methods do ***not return*** a ***copy*** of the list …

Review: Iterating through a List

- Read as
  - For every element in the list ...

An item in the list

list object

```
for item in list:
        print(item)
```

Iterates through *items* in list

- Output equivalent to

```
for x in range(len(list)):
        print(list[x])
```

Iterates through *positions* in list

daysOfWeek.py

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

In the Python Interpreter

```
>>> filename = "data/famous_pairs.txt"
>>> myfile = open(filename, "r")
>>> contents = myfile.read()
>>> contents
'Romeo & Juliet\nPeanut Butter & Jelly\nOrville & Wilbur
Wright\nMeriwether Lewis & William Clark\nSonny & Cher\nWhifield
Diffie & Martin Hellman\nBarbie & Ken\n'
>>> print(contents)
Romeo & Juliet
Peanut Butter & Jelly
Orville & Wilbur Wright
Meriwether Lewis & William Clark
Sonny & Cher
Whifield Diffie & Martin Hellman
Barbie & Ken

>>>
```

In the Python Interpreter

```
>>> filename = "data/famous_pairs.txt"
>>> myfile = open(filename, "r")
>>> myline = myfile.readline()
>>> myline
'Romeo & Juliet\n'
>>> print(myline)
Romeo & Juliet

>>> contents = myfile.read()
>>> contents
'Peanut Butter & Jelly\nOrville & Wilbur Wright\nMeriwether Lewis &
William Clark\nSonny & Cher\nWhifield Diffie & Martin
Hellman\nBarbie & Ken\n'
>>>
```

Nuance: Clarify what the `read()` method does

Review: Writing to a File

- Create a file object in **write** mode:
  - `myFile = open("demo.txt", "w")`
- Call write method on file object:
  - `myFile.write("Write string to file")`
  - `myFile.write("Also this string")`
- Close the file:
  - `myFile.close()`

What will demo.txt contain after executing program?
After executing the program a second time?

Review: Writing to a File

- Create a file object in **write** mode:
  - `myFile = open("demo.txt", "w")`
- Call write method on file object:
  - `myFile.write("Write string to file")`
  - `myFile.write("Also this string")`
- Close the file:
  - `myFile.close()`

Good template for working with files:

1. Open file
2. Process file
3. Close file

Review: Modules

- Modules group together related functions and constants
- Unlike functions, no special keyword to define a module
  - A module is named by its filename
- You've used modules in the past
  - graphics.py
  - test.py
  - game.py

Just a Python file!

Calling Function in Context

```python
def main():

    # can change this later to get user input for the
    # filename or loop through a bunch of file names or ...
    avgTemp = calculateAvgTemp(DATAFILE)

    print("The average temperature is {:.2f}".format(avgTemp))
```

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 for the location?

```
def calculateAvgTemp( datafileName ):
```

Algorithm for function?

`avgData.py`

Problem: Report of Avg Temperature

- **Given**: data files that contains the daily high temperatures for last year at various locations
  - Data file contains one temperature per line
  - Example: `data/florida.dat`
- **Problem**: Write a report of the locations and the average temperature in the form
  - Average temperature should be displayed to two decimal places

```
<location1> <avgtemp1>
<location2> <avgtemp2>
…
```

`reportAvgData.py`

Problem: Report of Avg Temperature

- Algorithm:
    - Open the file for writing
    - For each location
        - Calculate the average temperature
        - Write out the information to the file
            - Use the format method
            - Include the \n
    - Close the file

Recursive Copy

- Many Unix commands have *command-line options*
  - Example: `ls -l`
    - `-l`: **l**ong form
    - Command run during `turnin` script so you can see the dates and other information on your submitted files.
- `cp` has the `–r` option, which means to *recursively* copy
  - Means: copy the directory and all of its contents (including subdirectories)
  - Example: to copy the lab8 directory and all of its contents into your cs111 directory
    - `cp –r /csci/courses/cs111/handouts/lab8 ~/cs111`

Lab 8 Overview

- Lists
- Modules
- Reading Files
- Writing Files
- Functions, Lists

Focus is on the current week, but we are using tools we learned in the last ~8 weeks.
Remember (or review) all that you can do.