TCSS 142 – Introduction to Programming

Autumn 2014
Day 14

Day 14 Overview

- Programming Assignment 1
- Lists
- Files

Programming A 1

- Questions?

A List

- A string is a data structure that allows a collection of elements:
  - it is homogenous – consisting of characters
  - it is sequential – characters are stored as a sequence

- A list is an ordered, sequential collection of zero or more Python data objects
  - it is heterogeneous - may consist of different data type items

0	1	2	3
55	‘dog’	4.57	“CSC 1100”

Basic List Ops

- Just like in a string each data item has its own position and the similar operations are allowed on lists:

– Indexing	[ ]
– Concatenation	+
– Repetition	*
– Membership	in, not in
– Length	len
– Slicing	[:]

List Practice

- Go to http://codingbat.com/python/List-1 and complete 8 of the list exercises listed at that link.

More Methods

- List elements are mutable – string elements are not mutable

- Function `list` converts other sequences to lists.

- `list` in conjunction with function `range` creates a list consisting of a sequence of numbers

- Additional list methods – page 304 and Python docs

Let’s try it

```
>>> myList = list(range(5))
>>> myList
???
>>> yourList = list(range(5, -1, -1)
>>> yourList
???
>>> yourList.append(9)
>>> yourList
???
>>> anotherList = list('green eggs and ham')
>>> anotherList
???
```

Exercise

- Write a function that accepts two strings and returns `True` if the two strings constitute an anagram, `False` otherwise
  - An anagram is a word, phrase, or name formed by rearranging the letters of another, such as *cinema*, formed from *iceman*
  - Hint: use both strings and lists to solve the problem

List Applications: Dispersion

- Dispersion measures how spread out the data values are
  - the range of data = dispersion
  - the range of data = maxValue – minValue

```
>>> alist = [20, 32, 21, 26, 33, 22, 18]
>>> max(alist) - min(alist)
>>> a = max("TCSS 142")
>>> a
???
>>> b = min ("TCSS 142")
>>> b
???
>>> a - b
???
```

Finding Central Tendency

- Central tendency could be found by computing:
  - Mean (average)
  - Median (item in the exact middle of a sequence)
  - Mode (item that occurs most often – we will not calculate mode for now)

Mean

- Mean is simply an average
  - Sum all elements
  - Divide the sum by the number of terms, if num of terms > 0

```
>>> myList = [24, 2, 20, 33, 78]
>>> mean = sum(myList) / len(myList)
>>> mean
???
```

Median

- To find the median, the data needs to be sorted:
  - For the odd number of items, the median is the exact middle value (element at length // 2)

0	1	2	3	4
1	2	5	7	8

  - For the even number of items, the median is the average of the 2 middle values

    ((element at (length//2) + element at (length//2)-1) / 2.0)

0	1	2	3	4	5
1	2	5	7	8	10

Median: Algorithm

1. If a list is not to be modify, first copy it
2. Sort the copy
3. If number of elements odd, divide length of list // 2 and store into variable `pos`

   ```
   median = list [pos]
   ```

4. If number of elements even, divide length of list // 2 and store into variable pos

   ```
   median = (list [pos] + list[pos-1]) / 2.0
   ```

Exercise

- Create a program called `median.py` and write a function that accepts a list of numbers and returns the median value in that list. The list is not to be modified by this function.

Files

- Large data sets reside in files
  - we will use .txt and .csv files

- To open a file for reading

```
filevariable = open('filename', 'r')
```

- To open a file for writing

```
filevariable = open("filename", "w")
```

- When done, one needs to close the files

```
filevariable.close()
```

File Reading Methods

- `read()` reads an entire file and returns its contents as a string or bytes object
- `readline()` reads a line at a time and returns it as a string or bytes object
- `readlines()` reads a file and returns it as a list of strings
- Methods above take an optional argument that denotes the number of bytes to be read
- For reading a file, you can iterate over a file object – works like `readline()`

```
for line in fileObject:
   # do something with line
```

Read/Write Position

- Sequential files are processed top-down, left-write, like any text written in English

- Whenever you read from a file, a read/write position keeps track of how much of the file was already processed up until this point
  - So if you repeat the read statement, the program will start reading where the previous read left off
  - If you want to start from the beginning, look at other methods provided for file objects or simply reopen the file

Reading from Files

- A for loop views a file object as a sequence of lines of text
  - On each pass through the loop, the loop variable is bound to the next line of text in the sequence

- A line of a file is defined as a sequence of characters (a string) up to an including a special character called newline (\n)
  - newline is blank and not visible
  - it is created when enter or return key is entered when typing

Processing All Lines

- Algorithm
  - For each line in a file:
    - Treat it as a string and do something with it

- Write a program `rainfall.py` that reads the file `rainfall.txt` and produces the following output to the screen:

```
Akron  had  25.81  inches of rain.
Albia  had  37.65  inches of rain.
Algona  had  30.69  inches of rain.
Allison  had  33.64  inches of rain.
```

Steps

- Make sure your program file and the data file are in the same directory

- Let's first echo print the contents of `rainfall.txt`

- Then, let's add necessary statements to print the line in the desired format

- What happens when you try to open a file that does not exist?

Writing to Files

- Open a file for writing using `'w'`

- String data are written to a file using the method `write`
  - Write method expects a single argument
  - The argument must be a string
  - If you want to create different lines, you must include the escape character `\n`

- Let's add to our code.

Questions

- What happens if you try to open a file for writing that does not exits?

- What happens if you open a file for writing that does exist and you write to it – what happens to the old file contents?

- What happens if you open a file for writing but the file has been already opened?

Calculations

- What if we want to process the file to produce the following output?

```
Akron had 65.56 cm of rain.
Albia had 95.63 cm of rain.
Algona had 77.95 cm of rain.
Allison had 85.45 cm of rain.
```

Let's adjust our program.

Earthquake Stats

- We will process statistics of earthquake data from one week in October , taken from: http://earthquake.usgs.gov/earthquakes/recenteqsww/Quakes/quakes_all.html

- Download the file `earthquakes.txt` and create a program `earthStats.py` in the same directory as your text file.

earthStats.py

- In `main` prepare the file object for reading
- Create a function that takes an open file object, reads it line by line and returns a list with earthquake magnitudes
- Create a function that takes a list and returns a median of a list without modifying the list
- Create a function that takes a list and prints its contents, its min, max, range, mean, and median to the screen
- Let's write our own implementations of minimum and maximum

While Controlled File Reading

```
rainfile = open("rainfall.txt","r")

aline = rainfile.readline()         # priming read

while aline != "":                  # test
  values = aline.split()
  print(values[0], " had ",values[1]," inches of rain.")
  aline = rainfile.readline()       # update read

rainfile.close()
```

Reading Web Pages

- A Web page is simply a type of file readable by a Web browser.

- We will look at a typical Web page file saved in HTML format (hypertext markup language)

```
<html>
 <head> info about the file </head>
 <body> info to be displayed in a browser</body>
</html>
```

Reading Web Pages

- URL – Uniform Resource Locator

  (protocol: // server / folders / document), e.g.

  http://www.tacoma.uw.edu/institute-technology/institute-technology

  http://faculty.washington.edu/monikaso/index.html

- Python provides `urlib` package and `urlib.request` module to get online data
  - once the URL is open, you can read the webpage as if you were reading data from a plain textfile using `read` and `readline` (returning `bytes` object, not strings)

Reading Web Pages

- Download `readWebpage.py`, examine, and run

- Suppose we want to:
  - count the number of lines in the heading of a Web page (<head></head>)
  - count the number of lines in the body of a Web page (<body></body>)
  - let's write a program `webcounter.py` to implement it

Algorithm

1. Read the lines until you reach <head> (do nothing with them)

2. Read the lines until you reach </head> and count them

3. Read the lines until you reach <body> (do nothing with them)

4. Read the lines until you reach </body> and count them

List Histogram Question

- Given a file of integer exam scores, such as:

```
82
66
79
63
83
```

Write a program that will print a histogram of stars indicating the number of students who earned each unique exam score.

```
85: *****
86: ************
87: ***
88: *
91: ****
```

Lists for tallying

- Use score as an index

- Use list location as a tally

0	0
1	0
2	0
…	…
50	0
51	0
…	…
99	0
100	0

List Histogram Answer

```
gradesTally = [0] * 101
fileobject = open("scores.txt", "r")

for line in fileobject:
    i = int(line)
    gradesTally[i] += 1

for x in range(101):
    print(x, end = ':')
    for z in range (gradesTally[x]):
        print('*', end = '')
    print()
fileobject.close()
```

Exercise

- Count the number of times each letter appears in a file using a list
  - Use characters as indexes (size?)
  - Lowercase vs uppercase
  - Value of A, a, Z, z
  - Filename `charcounter.py`

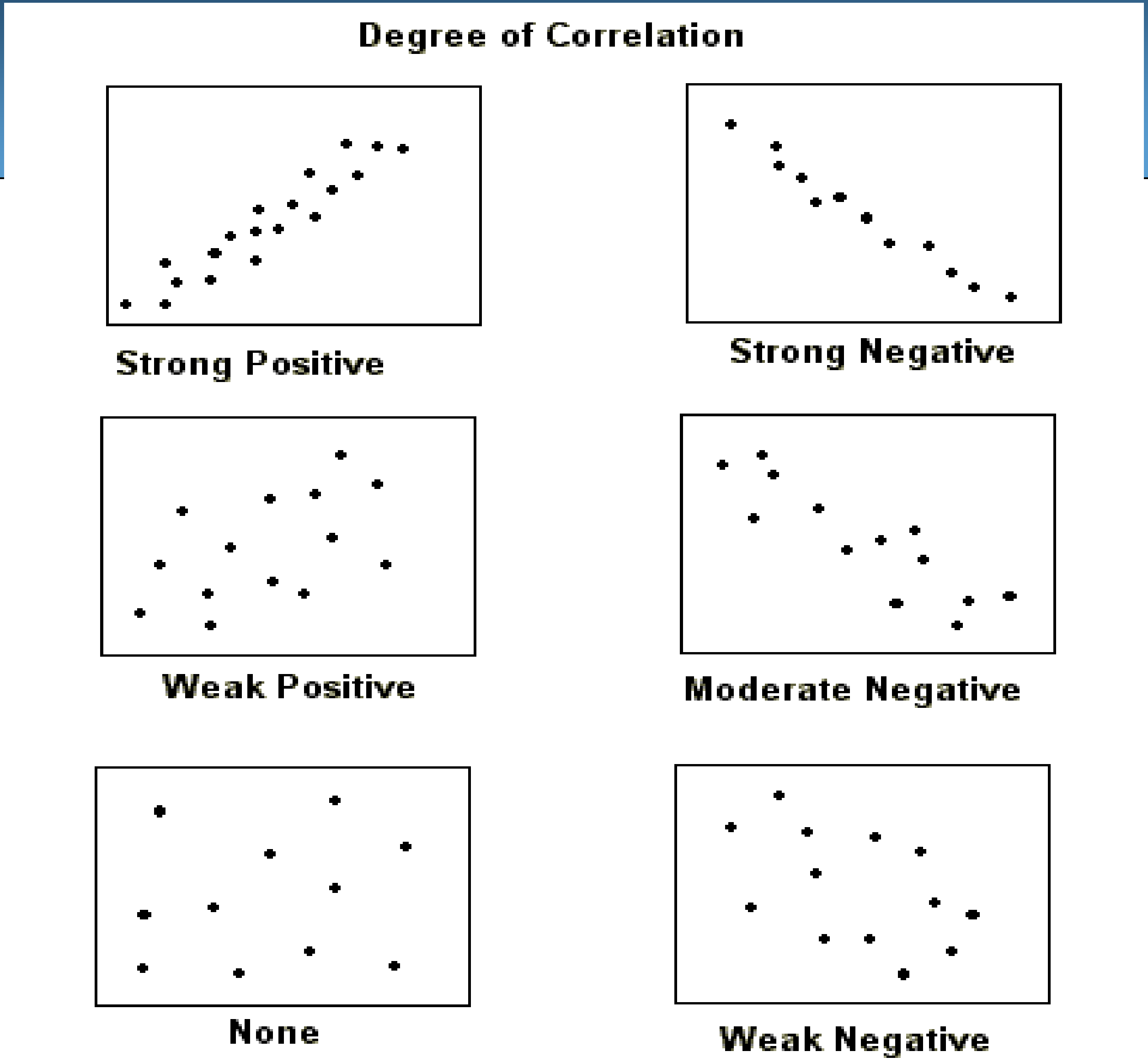
Internet as a Source of Data

- A lot of data nowadays resides in `csv` files.
  - comma separated values
  - just a text file that organizes records into rows and delimits fields with commas instead of blanks
  - download `amazonstock.csv` and open with `Notepad`
  - `Excel` provides support for `csv` files as well – open in `Excel`

- Consider http://ichart.finance.yahoo.com/table.csv?s=AAPL&d=9&e=23&f=2014
  - ? Indicates parameters coming in key-value pairs
  - s gives us the stock name in this case Apple
  - d, e, f, stand for the date

Correlation

- We will look at the correlation between Apple and Amazon stock data
- Correlation measures the strength and direction of the relationship between two variables (but don't assume causal relationship)
  - 1.0 means positive correlation (both vars get larger)
  - 0.0 no correlation
  - -1.0 negative correlation (as one gets larger, the other var gets smaller)



x axis values of one variable
y axis of the other variable

Correlation

- We will use the Pearson correlation coefficient to tell us the correlation between closing stock prices
  - we will use a function that calculates the correlation for us as a black box (like a library function)
  - we will feed two lists of values to it to get back the correlation factor
    - the lists need to be of the same length
  - download `amazonstock.csv` and `stocks.py` from `Canvas`

Preprocessing Data

- To make sure the data is compatible, we will extract only the closing prices for dates that both stocks have in common:
  - this will also ensure that lists are of the same length

- We have two files of uneven length
  - Approach 1: construct appropriate lists as you are reading from a file
  - Approach 2: read each file into a separate list and then adjust lists based on dates

Approach 1

Date	Open	High	Low	Close	Volume	Adj Close
2014-10-01	322.04	325.16	311.31	322.2	3565200	322.2
2014-09-02	339.98	349.38	317.64	322.44	3363900	322.44
2014-08-01	313.69	346.67	304.59	339.04	3045400	339.04
2014-07-01	325.86	364.85	311.86	312.99	4755300	312.99
2014-04-01	338.09	348.3	288	304.13	6779300	304.13

2014-08-01	94.9	102.9	93.28	102.5	46746200	102.06
2014-07-01	93.52	99.44	92.57	95.6	49637900	94.72
2014-06-02	633.96	651.26	89.65	92.93	59839500	92.07
2014-05-01	592	644.17	580.33	633	74996300	89.59
2014-04-01	537.76	599.43	511.33	590.09	82044000	83.06
2014-03-03	523.42	549	522.81	536.74	61552000	75.55
2014-02-03	502.61	551.19	499.3	526.24	82267500	74.07

list1	list2

Approach 1

Date	Open	High	Low	Close	Volume	Adj Close
2014-10-01	322.04	325.16	311.31	322.2	3565200	322.2
2014-09-02	339.98	349.38	317.64	322.44	3363900	322.44
2014-08-01	313.69	346.67	304.59	339.04	3045400	339.04
2014-07-01	325.86	364.85	311.86	312.99	4755300	312.99
2014-04-01	338.09	348.3	288	304.13	6779300	304.13

list1	list2

2014-08-01	94.9	102.9	93.28	102.5	46746200	102.06
2014-07-01	93.52	99.44	92.57	95.6	49637900	94.72
2014-06-02	633.96	651.26	89.65	92.93	59839500	92.07
2014-05-01	592	644.17	580.33	633	74996300	89.59
2014-04-01	537.76	599.43	511.33	590.09	82044000	83.06
2014-03-03	523.42	549	522.81	536.74	61552000	75.55
2014-02-03	502.61	551.19	499.3	526.24	82267500	74.07

Approach 1

Date	Open	High	Low	Close	Volume	Adj Close
2014-10-01	322.04	325.16	311.31	322.2	3565200	322.2
2014-09-02	339.98	349.38	317.64	322.44	3363900	322.44
2014-08-01	313.69	346.67	304.59	339.04	3045400	339.04
2014-07-01	325.86	364.85	311.86	312.99	4755300	312.99
2014-04-01	338.09	348.3	288	304.13	6779300	304.13

2014-08-01	94.9	102.9	93.28	102.5	46746200	102.06
2014-07-01	93.52	99.44	92.57	95.6	49637900	94.72
2014-06-02	633.96	651.26	89.65	92.93	59839500	92.07
2014-05-01	592	644.17	580.33	633	74996300	89.59
2014-04-01	537.76	599.43	511.33	590.09	82044000	83.06
2014-03-03	523.42	549	522.81	536.74	61552000	75.55
2014-02-03	502.61	551.19	499.3	526.24	82267500	74.07

list1	list2
339.04	102.06

Approach 1

Date	Open	High	Low	Close	Volume	Adj Close
2014-10-01	322.04	325.16	311.31	322.2	3565200	322.2
2014-09-02	339.98	349.38	317.64	322.44	3363900	322.44
2014-08-01	313.69	346.67	304.59	339.04	3045400	339.04
2014-07-01	325.86	364.85	311.86	312.99	4755300	312.99
2014-04-01	338.09	348.3	288	304.13	6779300	304.13

list1	list2
339.04	102.06
312.99	94.72

2014-08-01	94.9	102.9	93.28	102.5	46746200	102.06
2014-07-01	93.52	99.44	92.57	95.6	49637900	94.72
2014-06-02	633.96	651.26	89.65	92.93	59839500	92.07
2014-05-01	592	644.17	580.33	633	74996300	89.59
2014-04-01	537.76	599.43	511.33	590.09	82044000	83.06
2014-03-03	523.42	549	522.81	536.74	61552000	75.55
2014-02-03	502.61	551.19	499.3	526.24	82267500	74.07

Approach 1

Date	Open	High	Low	Close	Volume	Adj Close
2014-10-01	322.04	325.16	311.31	322.2	3565200	322.2
2014-09-02	339.98	349.38	317.64	322.44	3363900	322.44
2014-08-01	313.69	346.67	304.59	339.04	3045400	339.04
2014-07-01	325.86	364.85	311.86	312.99	4755300	312.99
2014-04-01	338.09	348.3	288	304.13	6779300	304.13

2014-08-01	94.9	102.9	93.28	102.5	46746200	102.06
2014-07-01	93.52	99.44	92.57	95.6	49637900	94.72
2014-06-02	633.96	651.26	89.65	92.93	59839500	92.07
2014-05-01	592	644.17	580.33	633	74996300	89.59
2014-04-01	537.76	599.43	511.33	590.09	82044000	83.06
2014-03-03	523.42	549	522.81	536.74	61552000	75.55
2014-02-03	502.61	551.19	499.3	526.24	82267500	74.07

list1	list2
339.04	102.06
312.99	94.72

Approach 1

Date	Open	High	Low	Close	Volume	Adj Close
2014-10-01	322.04	325.16	311.31	322.2	3565200	322.2
2014-09-02	339.98	349.38	317.64	322.44	3363900	322.44
2014-08-01	313.69	346.67	304.59	339.04	3045400	339.04
2014-07-01	325.86	364.85	311.86	312.99	4755300	312.99
2014-04-01	338.09	348.3	288	304.13	6779300	304.13

2014-08-01	94.9	102.9	93.28	102.5	46746200	102.06
2014-07-01	93.52	99.44	92.57	95.6	49637900	94.72
2014-06-02	633.96	651.26	89.65	92.93	59839500	92.07
2014-05-01	592	644.17	580.33	633	74996300	89.59
2014-04-01	537.76	599.43	511.33	590.09	82044000	83.06
2014-03-03	523.42	549	522.81	536.74	61552000	75.55
2014-02-03	502.61	551.19	499.3	526.24	82267500	74.07

list1	list2
339.04	102.06
312.99	94.72

Approach 1

Date	Open	High	Low	Close	Volume	Adj Close
2014-10-01	322.04	325.16	311.31	322.2	3565200	322.2
2014-09-02	339.98	349.38	317.64	322.44	3363900	322.44
2014-08-01	313.69	346.67	304.59	339.04	3045400	339.04
2014-07-01	325.86	364.85	311.86	312.99	4755300	312.99
2014-04-01	338.09	348.3	288	304.13	6779300	304.13

2014-08-01	94.9	102.9	93.28	102.5	46746200	102.06
2014-07-01	93.52	99.44	92.57	95.6	49637900	94.72
2014-06-02	633.96	651.26	89.65	92.93	59839500	92.07
2014-05-01	592	644.17	580.33	633	74996300	89.59
2014-04-01	537.76	599.43	511.33	590.09	82044000	83.06
2014-03-03	523.42	549	522.81	536.74	61552000	75.55
2014-02-03	502.61	551.19	499.3	526.24	82267500	74.07

list1	list2
339.04	102.06
312.99	94.72
304.13	83.06

Approach 1 Algorithm

1. Read one line from each file

2. While the read succeeded:
   a. Compare dates
      - If date the same, append closing prices to relevant lists and read one line from each file
      - If date1 > date2, read line from file1
      - If date1 < date2, read line from file2

3. Send both lists to correlation function

Comparing Dates

- Dates are read as strings – how are strings compared?

- Open the csv file in `Notepad` – how are dates formatted – would string comparison yield proper results?
  - If yes, use it
  - If not, convert a string to a date first

Strings as Dates

- Python contains a module datetime that has a function date
  - datetime.date(year, month, day)
  - where year, month, day need to be integers
- We need to parse a string so that we can create a date out of it
  - split on '-'
  - convert each subcomponent to int and send to date function

```python
import datetime

string1 = input("Enter a string: ")
string2 = input("Enter a string: ")

val1 = string1.split("/")
val2 = string2.split("/")

date1 = datetime.date(int(val1[2]), int(val1[0]), int(val1[1]))
date2 = datetime.date(int(val2[2]), int(val2[0]), int(val2[1]))

if string1 > string2:
    print(string1)
elif string1 < string2:
    print(string2)

if date1 > date2:
    print(date1)
elif date1 < date2:
    print(date2)
```

Putting It All Together

- Let’s put all these ideas together to solve the problem

Approach 2 Algorithm

1. Read data from file 1 into list 1

2. Read data from file 2 into list 2

3. Compare both lists position by position

    a. if dates the same, update both positions

    b. if date 1 > date 2, delete date 1 record from list 1, update list 1 position

    c. if date 1 < date 2, delete date 2 record from list 2, update list 2 position

Last Slide ☺

- Class ends at 17:10