CSc 110
2D Lists

Benjamin Dicken

2D Lists!

- Putting lists within lists
- Can actually have more than 2 levels

Review

Should be familiar with this kind of list usage:

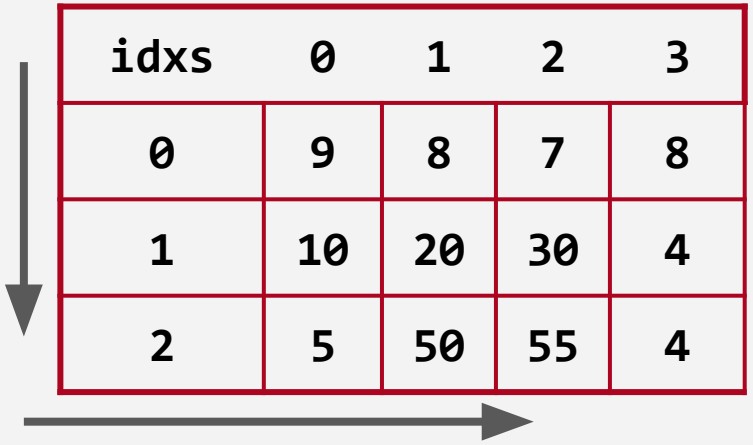
```
items = [5, 10, 20, 6, 7, 8]
items[0] = 10
items[1] = 3
items[4] = 99
print(str(items))
```

index	0	1	2	3	4	5
value	10	3	20	6	99	8

2D list

- The ***“first dimension”*** is the outer list
- The ***“second dimension”*** are the inner lists
- When we draw pictures of 2D lists, the first dimension is the vertical axis, the second is the horizontal
- Notice how the list can be formatted to reflect the 2D diagram

```
items = [[9,  8,  7,  8],
         [10, 20, 30, 4],
         [5,  50, 55, 4]]
```



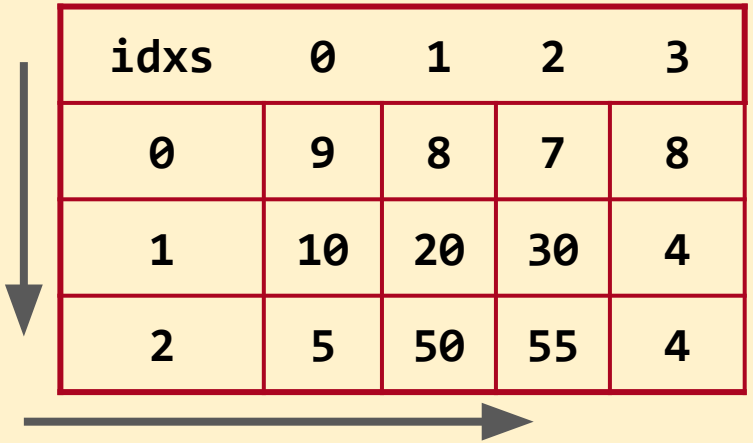
idxs	0	1	2	3
0	9	8	7	8
1	10	20	30	4
2	5	50	55	4

2D list

What will each of the below list accesses evaluate to?

```
val_a = items[0][0]
val_b = items[2][3]
val_c = items[1][2]
print(val_a, val_b, val_c)
```

```
items = [[9,  8,  7,  8],
         [10, 20, 30, 4],
         [5,  50, 55, 4]]
```



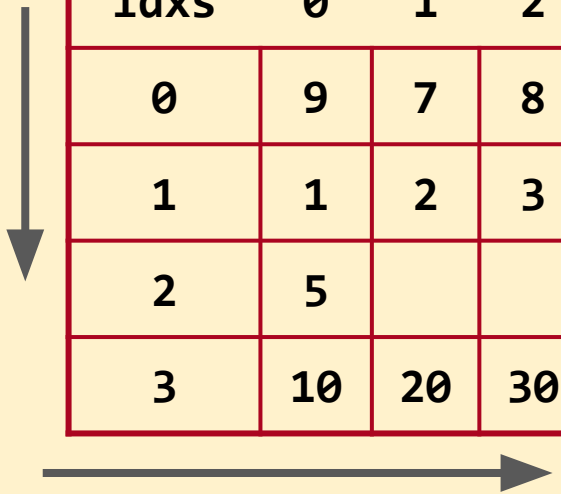
idxs	0	1	2	3
0	9	8	7	8
1	10	20	30	4
2	5	50	55	4

Which lookups are valid?

```
val_a = items[1][5]  # ?
val_b = items[2][5]  # ?
val_c = items[3][2]  # ?
```

```
items = [[9, 7, 8],
         [1, 2, 3, 4, 5, 6],
         [5],
         [10, 20, 30]]
```

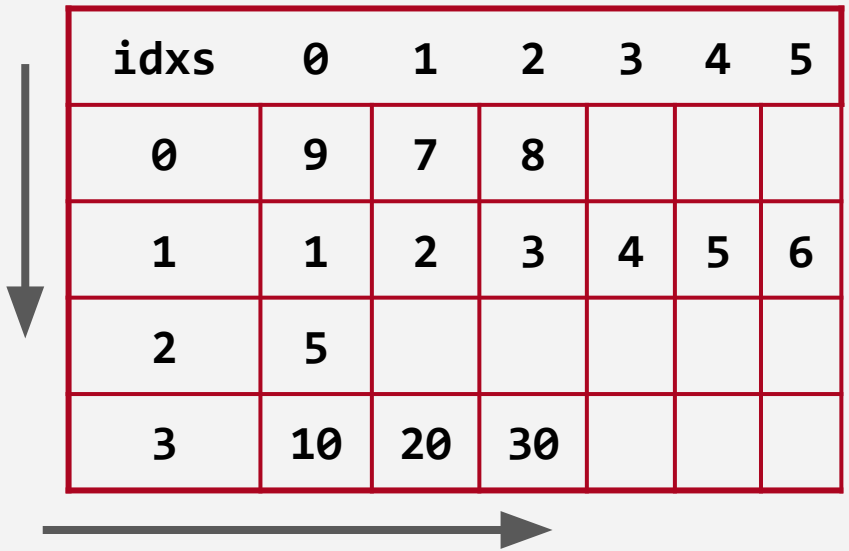
idxs	0	1	2	3	4	5
0	9	7	8			
1	1	2	3	4	5	6
2	5					
3	10	20	30			



Which lookups are valid?

```
items = [[9, 7, 8],
         [1, 2, 3, 4, 5, 6],
         [5],
         [10, 20, 30]]
```

```
val_a = items[1][5]   # OK
val_b = items[2][5]   # NOT OK
val_c = items[3][2]   # OK
```



idxs	0	1	2	3	4	5
0	9	7	8			
1	1	2	3	4	5	6
2	5					
3	10	20	30			

What will it print?

```
names = [["Ric", "Janet", "Joe"],
         ["Ike", "Alan", "Ko"],
         ["Cam", "Jan", "Jane"]]

l1 = names[1][0][2]
l2 = names[2][1][2]
l3 = names[0][1][4]
print(l1 + l2 + l3)
```

2D list

```
names = [["Ric", "Janet", "Joe"],
         ["Ike", "Alan", "Ko"],
         ["Cam", "Jan", "Jane"]]
```

- 2D lists of strings are kind-of like a 3D list
- The first 2 dimensions are the lists, the last is the characters of the string

```
names          ==  [["Ric", "Janet", "Joe"], ["Ike", "Alan", "Ko"],
                    ["Cam", "Jan", "Jane"]]
names[2]       ==  ["Cam", "Jan", "Jane"]
names[2][1]    ==  "Jan"
names[2][1][0] ==  "J"
```

Traversing a 2d List

- As with regular lists, it is often useful to iterate (loop) through all of the elements in a 2D list
- This can be achieved in a few ways
  - Using the indexes
  - Using the actual values
- There are advantages to each one

What will this print?

```
names = [["Ric", "Janet", "Joe"],
         ["Ike", "Alan", "Ko"],
         ["Cam", "Jan", "Jane"]]

for i in names:
    print(i)
```

What will this print?

```
names = [["Ric", "Janet", "Joe"],
         ["Ike", "Alan", "Ko"],
         ["Cam", "Jan", "Jane"]]

for i in names:
    print(i)
```

```
['Ric', 'Janet', 'Joe']
['Ike', 'Alan', 'Ko']
['Cam', 'Jan', 'Jane']
```

What will this print?

```
names = [["Ric", "Janet", "Joe"],
         ["Ike", "Alan", "Ko"],
         ["Cam", "Jan", "Jane"]]

for i in names:
    for j in names:
        print(i)
```

What will this print?

```
names = [["Ric", "Janet", "Joe"],
         ["Ike", "Alan", "Ko"],
         ["Cam", "Jan", "Jane"]]

for i in names:
    for j in names:
        print(i)
```

```
['Ric', 'Janet', 'Joe']
['Ric', 'Janet', 'Joe']
['Ric', 'Janet', 'Joe']
['Ike', 'Alan', 'Ko']
['Ike', 'Alan', 'Ko']
['Ike', 'Alan', 'Ko']
['Cam', 'Jan', 'Jane']
['Cam', 'Jan', 'Jane']
['Cam', 'Jan', 'Jane']
```

What will this print?

```
names = [["Ric", "Janet", "Joe"],
         ["Ike", "Alan", "Ko"],
         ["Cam", "Jan", "Jane"]]

for i in names:
    for j in names:
        print(j)
```

What will this print?

```
names = [["Ric", "Janet", "Joe"],
         ["Ike", "Alan", "Ko"],
         ["Cam", "Jan", "Jane"]]

for i in names:
    for j in names:
        print(j)
```

```
['Ric', 'Janet', 'Joe']
['Ike', 'Alan', 'Ko']
['Cam', 'Jan', 'Jane']
['Ric', 'Janet', 'Joe']
['Ike', 'Alan', 'Ko']
['Cam', 'Jan', 'Jane']
['Ric', 'Janet', 'Joe']
['Ike', 'Alan', 'Ko']
['Cam', 'Jan', 'Jane']
```

What will this print?

```
names = [["Ric", "Janet", "Joe"],
         ["Ike", "Alan", "Ko"],
         ["Cam", "Jan", "Jane"]]

for i in names:
    for j in i:
        print(j)
```

What will this print?

```
names = [["Ric", "Janet", "Joe"],
         ["Ike", "Alan", "Ko"],
         ["Cam", "Jan", "Jane"]]

for i in names:
    for j in i:
        print(j)
```

```
Ric
Janet
Joe
Ike
Alan
Ko
Cam
Jan
Jane
```

Activity

What would you change?

```
names = [["Ric", "Janet", "Joe"],
         ["Ike", "Alan", "Ko"],
         ["Cam", "Jan", "Jane"]]

for i in names:
    for j in i:
        print(j)
```

```
Ric
Janet
Joe
Ike
Alan
Ko
Cam
Jan
Jane
```

```
Ric    Janet  Joe
Ike    Alan   Ko
Cam    Jan    Jane
```

What would you change?

```
names = [["Ric", "Janet", "Joe"],
         ["Ike", "Alan", "Ko"],
         ["Cam", "Jan", "Jane"]]

for i in names:
    for j in i:
        print(j + '\t', end='')
    print()
```

```
Ric
Janet
Joe
Ike
Alan
Ko
Cam
Jan
Jane
```

```
Ric     Janet   Joe
Ike     Alan    Ko
Cam     Jan     Jane
```

What will this print?

```
names = [["Ric", "Janet", "Joe"],
         ["Ike", "Alan", "Ko"],
         ["Cam", "Jan", "Jane"]]

for i in range(0, len(names)):
    for j in range(0, len(names[i])):
        print(names[i][j], end="\t")
    print()
```

What will this print?

```
names = [["Ric", "Janet", "Joe"],
         ["Ike", "Alan", "Ko"],
         ["Cam", "Jan", "Jane"]]

for i in range(0, len(names)):
    for j in range(0, len(names[i])):
        print(names[-j-1][i], end="\t")
    print()
```

Implement the function

- Implement a function named **filter_employees** that has two parameters, a 2D list where each row represents an employee, and a number
- Each row is: name, dept, ID, salary
- Should print names of employees with salaries greater than the second parameter

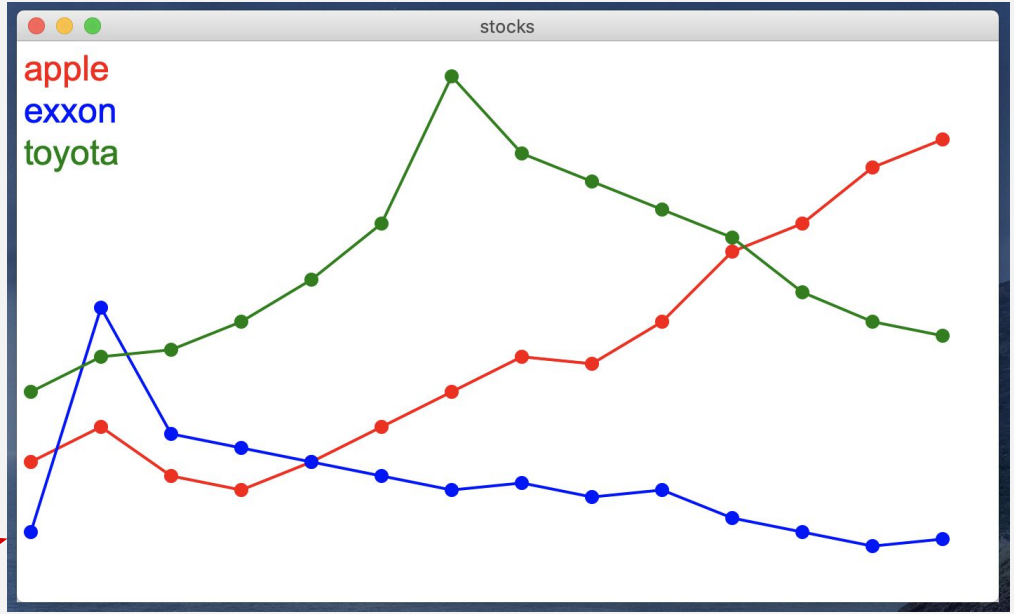
```
employees = [ ['Joe Smith', 'Engineer', 102, 100000],
              ['Jane Doe', 'Sales', 450, 50000],
              ['Janet Paul', 'Legal', 151, 150000],
              ['Cameron Johnson', 'CEO', 50, 500000]
              ['Sally Sanderson', 'Cheif Scientist', 100, 1000000]]
filter_employees(employees, 100000)
```

Implement the function

- Implement a function named **get_highest_average** that has one parameter, a 2D list
- Each row is: list of 7 integers, representing temp highs for each day of a week
- Should return highest average

```
weeks = [ [100, 100, 105, 105, 102, 102, 100],
          [70, 71, 75, 72, 81, 80, 74],
          [78, 72, 74, 71, 70, 70, 70],
          [30, 32, 40, 38, 31, 32, 30] ]
average = get_highest_average(weeks)
print(average) # should print 102
```

Graphical Stock Plotter



stocks.csv

```
Apple,red,100,125,90,80,100,125,150,175,170,200,250,270,310,330
Exxon,blue,50,210,120,110,100,90,80,85,75,80,60,50,40,45
toyota,green,150,175,180,200,230,270,375,320,300,280,260,221,200,190
```

CSV

- The CSV file format is a common way to represent tabular data
  - **C**omma-**S**eparated **V**alues
  - Regular text files, with particular rules about how the data within it is formatted
    - Useful analogy: text representation of a excel spreadsheet

CSV Formatting

- Each line of the file represents a “row”
- Each line has value(s) separated by commas
- Each line should have the same number of columns
- The first line can (optionally) be a titles for each column

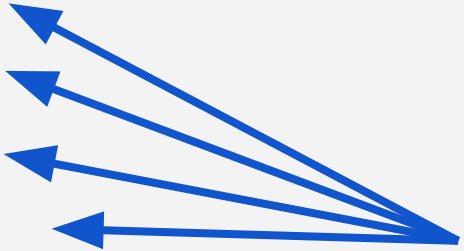
CSV Grades example

```
Name,SID,exam1,exam2,final
Steve,1429,67,84,91
Annie,3211,81,85,85
James,3171,90,75,80
Carlita,7877,85,87,89
```

CSV Grades example (students.csv)

```
Name,SID,exam1,exam2,final
Steve,1429,67,84,91
Annie,3211,81,85,85
James,3171,90,75,80
Carlita,7877,85,87,89
```

Optional row of column titles

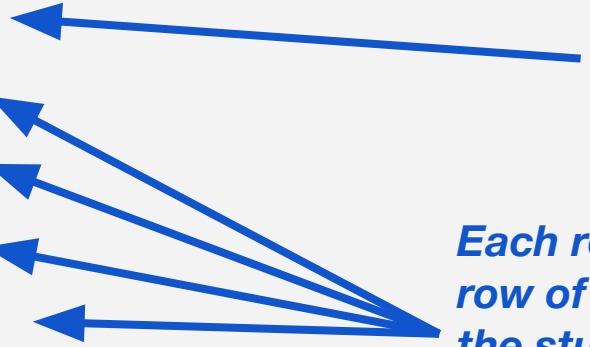


Each row represents a row of data (an entry in the student table

Each row has the same number of columns - to match the titles

CSV Grades example (students.csv)

```
Steve,1429,67,84,91
Annie,3211,81,85,85
James,3171,90,75,80
Carlita,7877,85,87,89
```



Title row is optional

Each row represents a row of data (an entry in the student table

Each row has the same number of columns - to match the titles

Load the contents of csv file into 2D list

```
csv_data = []

# What should go here?
# Load data as so:
# [ [1, 4, 7, 2],
#   . . . .
#   [0, 1, 0, 1] ]
```

numbers.csv

```
1,4,7,2
3,10,20,30
3,4,7,1
1,1,1,2
1,5,5,1
0,1,0,1
```

Load the contents of csv file into list

```
csv_data = []

csv_file = open('numbers.csv', 'r')

# What should go here?
```

numbers.csv

```
1,4,7,2
3,10,20,30
3,4,7,1
1,1,1,2
1,5,5,1
0,1,0,1
```

Load the contents of csv file into list

```
csv_data = []

csv_file = open('numbers.csv', 'r')
for line in csv_file:
    line = line.strip('\n')
    values = line.split(',')
    # What should go here?
```

numbers.csv

```
1,4,7,2
3,10,20,30
3,4,7,1
1,1,1,2
1,5,5,1
0,1,0,1
```

Load the contents of csv file into list

```
csv_data = []

csv_file = open('numbers.csv', 'r')
for line in csv_file:
    line = line.strip('\n')
    values = line.split(',')
    csv_data.append(values)
```

numbers.csv

```
1,4,7,2
3,10,20,30
3,4,7,1
1,1,1,2
1,5,5,1
0,1,0,1
```

Load the contents of csv file into list

```
csv_data = []

csv_file = open('numbers.csv', 'r')
for line in csv_file:
    line = line.strip('\n')
    values = line.split(',')
    csv_data.append(values)
```

What type are the values in the 2D list?

numbers.csv

```
1,4,7,2
3,10,20,30
3,4,7,1
1,1,1,2
1,5,5,1
0,1,0,1
```

Load the contents of csv file into list

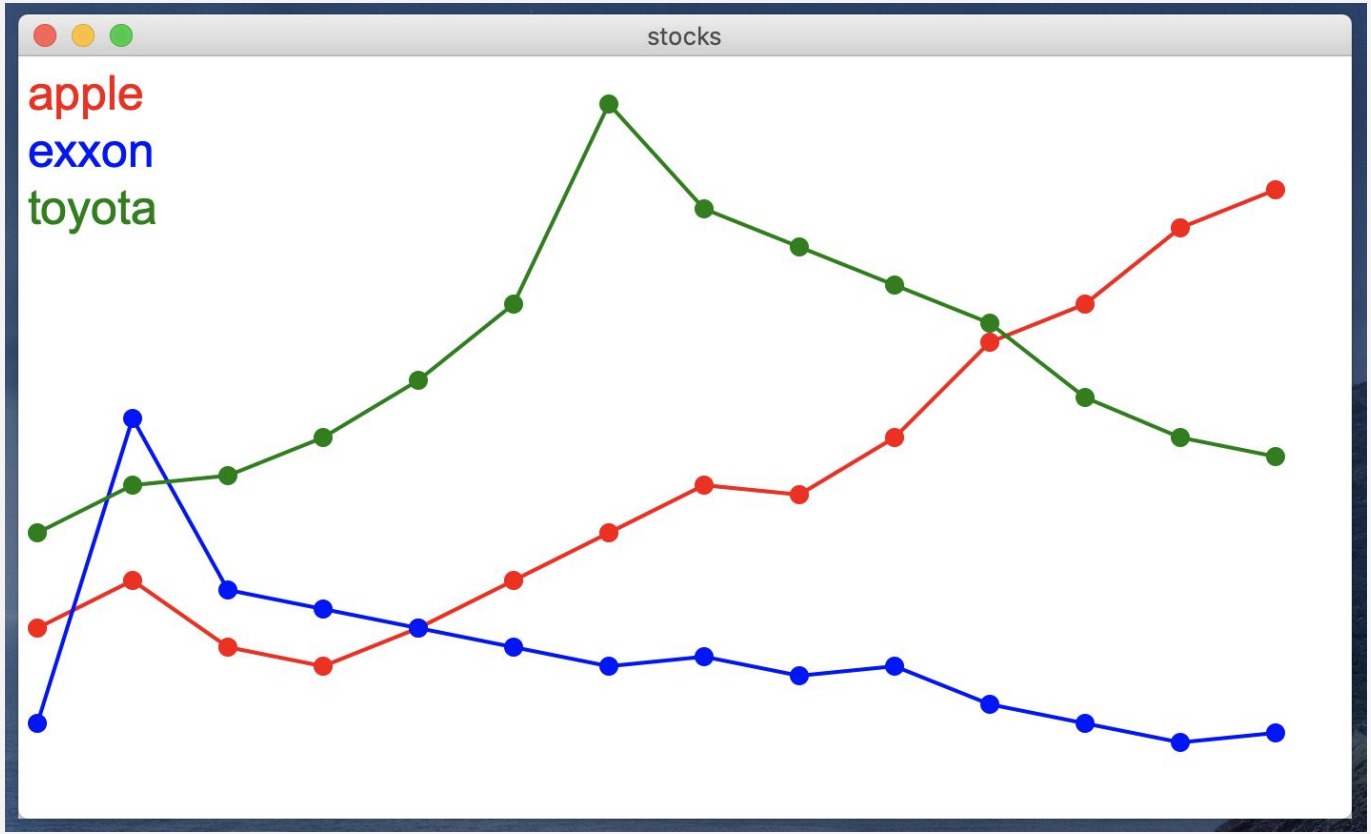
```
csv_data = []

csv_file = open('numbers.csv', 'r')
for line in csv_file:
    line = line.strip('\n')
    values = line.split(',')
    num_vals = []
    for i in values:
        num_vals.append( int(i) )
    csv_data.append(num_vals)
```

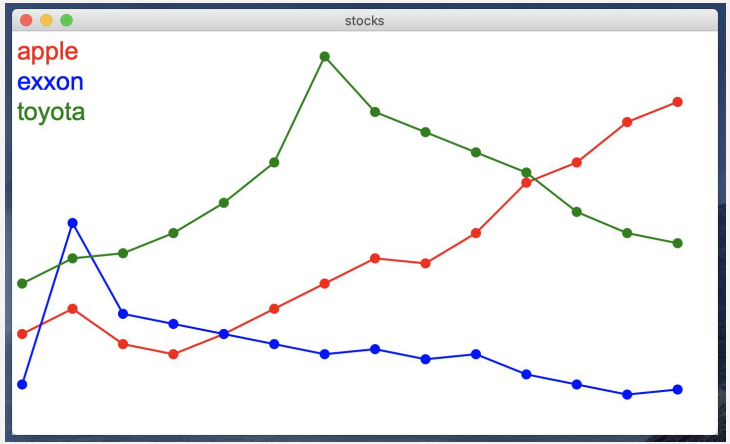
numbers.csv

```
1,4,7,2
3,10,20,30
3,4,7,1
1,1,1,2
1,5,5,1
0,1,0,1
```

Graphical Stock Plotter



Graphical Stock Plotter

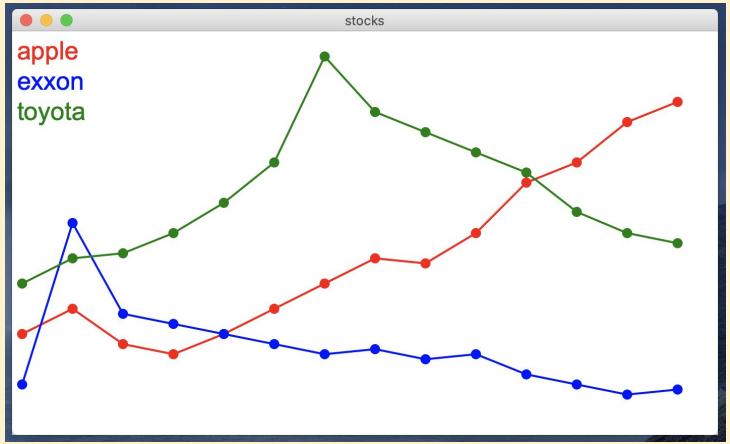


stocks.csv

```
Apple,red,100,125,90,80,100,125,150,175,170,200,250,270,310,330
Exxon,blue,50,210,120,110,100,90,80,85,75,80,60,50,40,45
toyota,green,150,175,180,200,230,270,375,320,300,280,260,221,200,190
```

Graphical Stock Plotter

* ***Download plot.py***
* ***Implement main***



stocks.csv

```
Apple,red,100,125,90,80,100,125,150,175,170,200,250,270,310,330
Exxon,blue,50,210,120,110,100,90,80,85,75,80,60,50,40,45
toyota,green,150,175,180,200,230,270,375,320,300,280,260,221,200,190
```

Implement the Function

```
def get_stock_data(file_name):
    '''
    It should open up a CSV file and load the information
    Into a 2D list. The first two columns should be
    added as strs, the others as integers.
    The function should return the 2D list after loaded.
    '''
```

stocks.csv

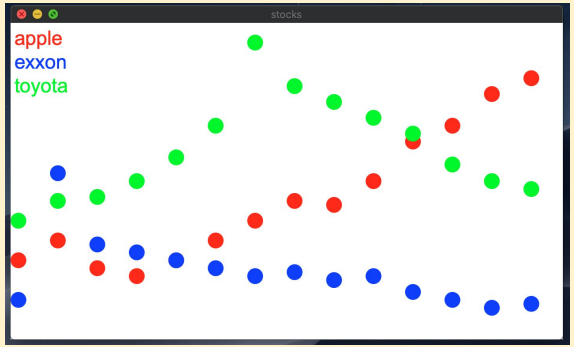
```
Apple,red,100,125,90,80,100,125,150,175,170,200,250,270,310,330
Exxon,blue,50,210,120,110,100,90,80,85,75,80,60,50,40,45
toyota,green,150,175,180,200,230,270,375,320,300,280,260,221,200,190
```

get_stock_data

```
def get_stock_data(file_name):
    f = open(file_name, 'r')
    data = []
    for line in f:
        row = []
        sp = line.split(',')
        row.append(sp[0])
        row.append(sp[1])
        for element in sp[2:]:
            row.append(int(element))
        data.append(row)
    return data
```

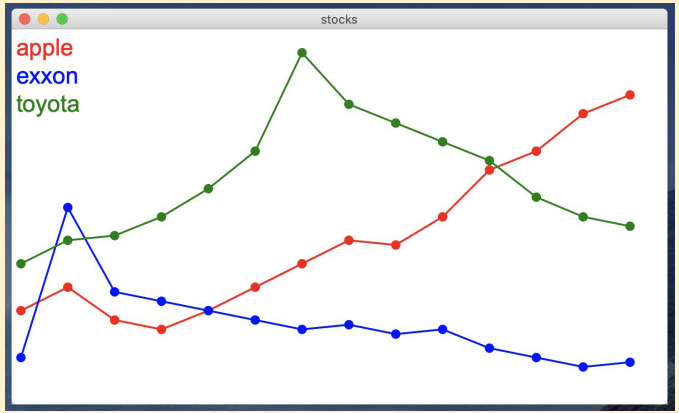
```
def main():
    file_name = input('Enter name of file: ')
    stock_data = get_stock_data(file_name)
    gui = graphics(700, 400, 'stocks')
    i = 0
    while i < len(stock_data):
        label = stock_data[i][0]
        color = stock_data[i][1]
        gui.text(5, 5+30*i, label, color, 25)
        # Finish the code
    gui.primary.mainloop()
```

Finish the main function



```
def main():
    file_name = input('Enter name of file: ')
    stock_data = get_stock_data(file_name)
    gui = graphics(700, 400, 'stocks')
    i = 0
    while i < len(stock_data):
        label = stock_data[i][0]
        color = stock_data[i][1]
        gui.text(5, 5+30*i, label, color, 25)
        j= 0
        while j < len(stock_data[i])-2:
            x = 10 + (j*50)
            y = -(stock_data[i][j+2]-400)
            gui.ellipse(x, y, 20, 20, color)
            j += 1
        i += 1
    gui.primary.mainloop()
```

Change to make dots smaller and add lines



main

```
def main():
    file_name = input('Enter name of file: ')
    stock_data = get_stock_data(file_name)
    gui = graphics(700, 400, 'stocks')
    i = 0
    while i < len(stock_data):
        label = stock_data[i][0]
        color = stock_data[i][1]
        gui.text(5, 5+30*i, label, color, 25)
        j= 0
        prev_x = -1
        prev_y = -1
        while j < len(stock_data[i])-2:
            x = 10 + (j*50)
            y = -(stock_data[i][j+2]-400)
            if prev_x >= 0:
                gui.line(prev_x, prev_y, x, y, color, 2)
            gui.ellipse(x, y, 10, 10, color)
            prev_x = x
            prev_y = y
            j += 1
        i += 1
    gui.primary.mainloop()
```

```
from graphics import graphics

def get_stock_data(file_name):
    f = open(file_name, 'r')
    data = []
    for line in f:
        row = []
        sp = line.split(',')
        row.append(sp[0])
        row.append(sp[1])
        for element in sp[2:]:
            row.append(int(element))
        data.append(row)
    return data
```

```
def main():
    file_name = input('Enter name of file: ')
    stock_data = get_stock_data(file_name)
    gui = graphics(700, 400, 'stocks')
    i = 0
    while i < len(stock_data):
        label = stock_data[i][0]
        color = stock_data[i][1]
        gui.text(5, 5+30*i, label, color, 25)
        j= 0
        prev_x = -1
        prev_y = -1
        while j < len(stock_data[i])-2:
            x = 10 + (j*50)
            y = -(stock_data[i][j+2]-400)
            if prev_x >= 0:
                gui.line(prev_x, prev_y, x, y, color, 2)
            gui.ellipse(x, y, 10, 10, color)
            prev_x = x
            prev_y = y
            j += 1
        i += 1
    gui.primary.mainloop()

main()
```