# CSC 110: Week 7, Lecture 2

Dr. Bodine

## Last time…

- Lists/arrays

- Operations and Methods
  - Concatentation, Length, Indexing, Slicing, …
  - Append, sort, delete vs pop, …

- Examples of using lists
  - Our grade program from assignment 3

## This time…

- Continue looking at our grade program from assignment 3
- Complex data structures
  - Collections of records
- Dictionaries

## Using our lists to do some maths…

- Consider our list of student scores from assignment 3
  - Remember our file that had a list of <First Name> <Last Name> <score>
  - We output grades based on the score

- Now what if I wanted to know how high above or below the average each person scored?
  - Modify our program to compute the average and then print for each student how their score compared to the average
  - <First Name> <Last Name> scored <insert number> <above/below> average

## How would we do this?

- First we have to go through the file to get the scores and store them in a list
- After we have the list we can get the average
- Once we have the average we can loop back through the scores and calculate the difference between the score and the average
  - We also need to check if the score is above or below average to be able to print the correct statement!

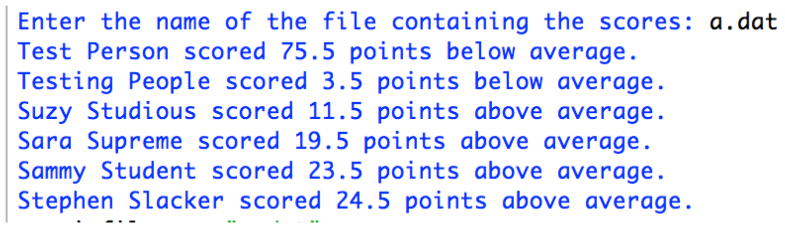
```python
def main():
    #get filename from user and open file
    infilename = input("Enter the name of the file containing the scores: ")
    infile =open(infilename,"r")

    #open outputfile
    outfile = open("gradesif.txt","w")

    scores=[]
    names=[]

    #read in lines and get scores
    for line in infile:
        entry = line.split()
        entry[2] = eval(entry[2]) #turn score into a number
        # append to our lists
        names.append(entry[0]+" "+entry[1])
        scores.append(entry[2])

    average = sum(scores)/len(scores)


    for i in range(len(names)):
        diff = scores[i]-average
        aboveorbelow =''
        if diff>=0:
            aboveorbelow = 'above'
        elif diff<0:
            aboveorbelow = 'below'
            diff =abs(diff)

        print(names[i],"scored",diff, "points", aboveorbelow, "average.")
```

```
Enter the name of the file containing the scores: a.dat
Test Person scored 75.5 points below average
Testing People scored 24.5 points above average
Suzy Studious scored 23.5 points above average
Sara Supreme scored 11.5 points above average
Sammy Student scored 19.5 points above average
Stephen Slacker scored 3.5 points below average
```

## Sorting

- What if we wanted to sort them first?  Want to print low to high…
- Well, we can sort scores and see what happens....

```python
def main2():
    #get filename from user and open file
    infilename = input("Enter the name of the file containing the scores: ")
    infile =open(infilename,"r")

    #open outputfile
    outfile = open("gradesif.txt","w")

    scores=[]
    names=[]

    #read in lines and get scores
    for line in infile:
        entry = line.split()
        entry[2] = eval(entry[2]) #turn score into a number
        # append to our lists
        names.append(entry[0]+" "+entry[1])
        scores.append(entry[2])

    average = sum(scores)/len(scores)
    scores.sort()

    for i in range(len(names)):
        diff = scores[i]-average
        aboveorbelow =''
        if diff>=0:
            aboveorbelow = 'above'
        elif diff<0:
            aboveorbelow = 'below'
            diff =abs(diff)

        print(names[i],"scored",diff, "points", aboveorbelow, "average.")
```



```
Enter the name of the file containing the scores: a.dat
Test Person scored 75.5 points below average.
Testing People scored 3.5 points below average.
Suzy Studious scored 11.5 points above average.
Sara Supreme scored 19.5 points above average.
Sammy Student scored 23.5 points above average.
Stephen Slacker scored 24.5 points above average.
```

Uh oh! Our records got out of order!!!!

Sort changed the order of the scores list but not the order of the names list!

## List inside lists…

- In python we can store lists inside of lists
    - [[<Name1>, <score1>],[<Name2, <score2>],[<Name3>,<score3>]]

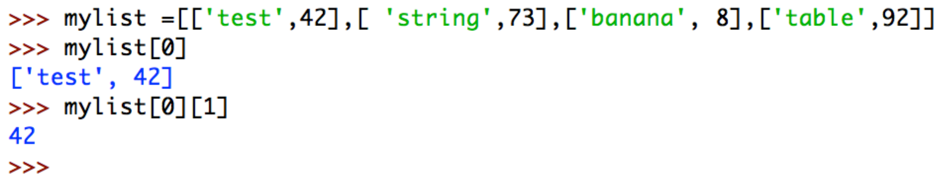
    - But we still need to know which index position contains which item

- Remember, this is not possible in other languages.
    - Need to keep track of data manually, more complicated sorting

# How would we make/use list of lists

- Start with a test list of lists:
  - mylist =[[‘test’,42],[ ‘string’,73],[ ‘banana’, 8],[‘table’,92]]

- How can I access the data inside the inner lists?
  - Indexing!  Let’s take a look at how this works…



```
>>> mylist =[['test',42],[ 'string',73],['banana', 8],['table',92]]
>>> mylist[0]
['test', 42]
>>> mylist[0][1]
42
>>>
```

  - How would we get ‘string’ out?  How would we get 92 out?

## Adding this to our program

- Start out the same: open file, initialize empty list, loop over lines
  - Have infile
  - Have entries = []
- Inside loop, we split and append the resulting list to the list.
  - Result is a list stored as the first element in the list
  - entries.append(line.split())
- Then we can access the score by knowing which element of the inner list the score is (2 in this case)

```
def main():
    #get filename from user and open file
    infilename = input("Enter the name of the file containing the scores: ")
    infile =open(infilename,"r")

    #open outputfile
    outfile = open("gradesif.txt","w")

    scores=[]
    entries=[]

    #read in lines and get scores
    for line in infile:
        entry = line.split()
        entry[2] = eval(entry[2])
        entries.append(entry)
   #turn score into a number from a string
        scores.append(entry[2])

    average = sum(scores)/len(scores)

    for i in entries:
        diff = i[2]-average
        if diff>0:
            print(i[0],i[1],"scored",diff, "points above average")
        elif diff<0:
            print(i[0],i[1],"scored",abs(diff), "points below average")
        elif diff ==0:
            print(i[0],i[1],"scored the same as the average")
```

## Let’s take a closer look at this code…

- What type were the elements in entries? What type were the elements in scores? Why does it matter?
- What happens if I try to index entries[i]? What happens if I try to index entries[i][j]?
- Why does i[index] work given our list of lists structure?

## Tuples in python

- Tuples are sequences
    - Say, what?
- Like lists except they are immutable
    - Items cannot be changed
- Uses ( ) instead of [ ]

## Remember lists…

- In Python lists are versatile sequences
  - Heterogenous, dynamic
  - Can store lists inside of lists!

- Used to store sequential data
  - Indexed by natural numbers

What if instead I want to index by something other than a number?

Example: Non-sequential data

## Dictionaries / Mappings

- Uses key-value pair to look up data
  - In contrast to indexing with natural numbers
  - For example, can say I want to get name instead of get the value in position 0
- Mapping is the a collection that allows us to look up information associated with an arbitrary key
  - May be called dictionaries, hashes, associative array

## Python dictionaries

- Mutable collections
  - From keys to values

- Keys must be immutable type
- Values can be any type

- Implemented with curly brackets:
  - Example: grades={“Test Student”:0, “Suzy Studious”:99}

## Dictionary operations and methods

- Assignment: <dict>[<key>]=<newvalue>
  - Used to add to dictionary, not just for reassigning

- Can access values like we did with indices: <dict>[<key>]

- <dict>.keys()
- <dict>.values()
- <dict>.items()
- <dict>.get(<key>,<default>)
- del <dict[<key>]
- <dict>.clear()
- For <var> in <dict>:
- <key> in <dict>

## Notes about key-value pairs

- Look up information based on key-value pairs
- Keys have to be unique!
  - Think of functions in mathematics
  - Multiple keys can have the same value but only one value per key.
  - This is very important in databases
- Values can be repeated by more than one key and can be reassigned

## Example with dictionaries

- Let’s use the example of students exam scores
- If we want to store their names associated with their scores we choose something unique for the key: FirstName+LastName
- Rewrite our program to make a dictionary with the scores

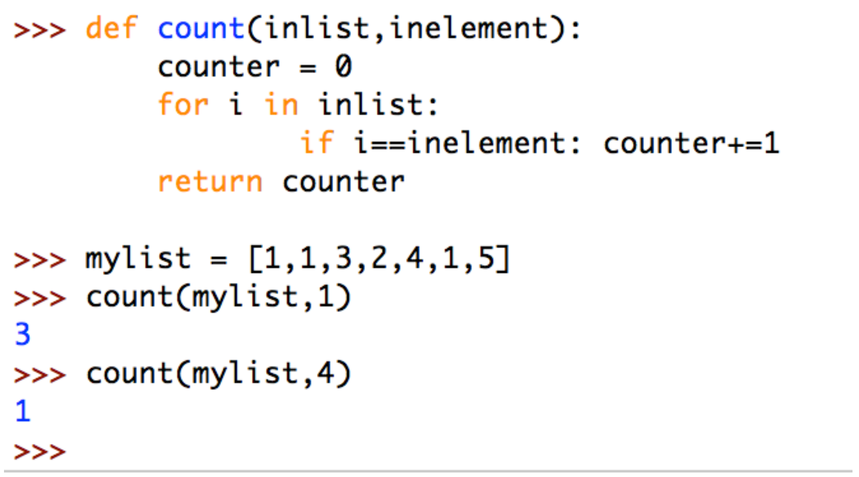
```python
def main():
    #get filename from user and open file
    infilename = input("Enter the name of the file containing the scores: ")
    infile =open(infilename,"r")

    #open outputfile
    outfile = open("gradesif.txt","w")

    scores={}
    mysum = 0
    number = 0

    #read in lines and get scores
    for line in infile:
        entry = line.split()
        name = entry[0]+ " " +entry[1]
        entry[2] = eval(entry[2]) #turn score into a number
        # add to our dictionary
        scores[name]=entry[2]
        mysum += entry[2]
        number+=1

    average = mysum/number

    for name in scores:
        diff = scores.get(name)-average
        aboveorbelow=''
        if diff>=0:
            aboveorbelow='above'
        elif diff<0:
            aboveorbelow='below'
            diff=abs(diff)

        print(name,"scored",diff, "points", aboveorbelow, "average")
```

```
Enter the name of the file containing the scores: a.dat
Test Person scored 75.5 points below average
Testing People scored 24.5 points above average
Suzy Studious scored 23.5 points above average
Sara Supreme scored 11.5 points above average
Sammy Student scored 19.5 points above average
Stephen Slacker scored 3.5 points below average
```

## Challenge problems

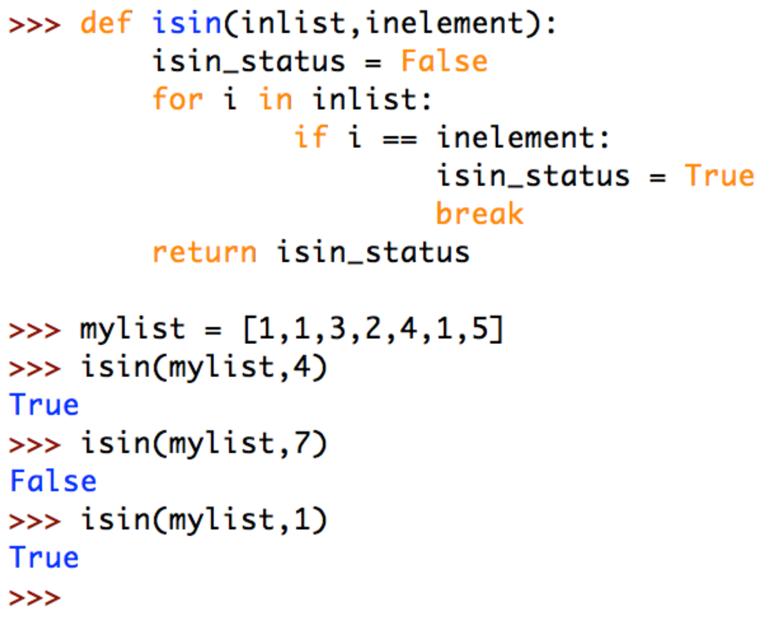
- Write an algorithm for the following operations in python and test by writing them up in a suitable function. (Do <u>not</u> use the standard methods to implement the functions)
- count(inlist,inelement) (should work like inlist.count(inelement))
- isin(inlist,inelement) (should work like inelement in inlist)
- index(inlist,inelement) (should work like inlist.index(inelement))

## count(mylist,x)



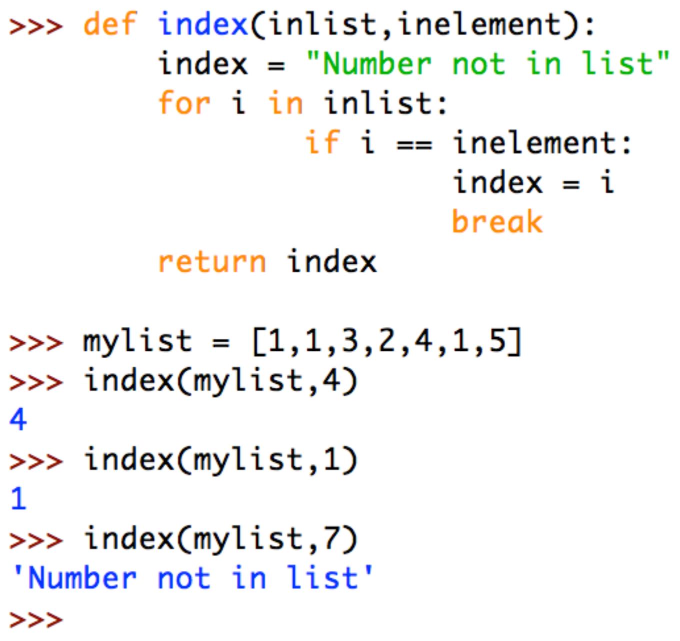
```
>>> def count(inlist,inelement):
	counter = 0
	for i in inlist:
		if i==inelement: counter+=1
	return counter

>>> mylist = [1,1,3,2,4,1,5]
>>> count(mylist,1)
3
>>> count(mylist,4)
1
>>>
```

## isin(mylist,x)



```
>>> def isin(inlist,inelement):
        isin_status = False
        for i in inlist:
                if i == inelement:
                        isin_status = True
                        break
        return isin_status

>>> mylist = [1,1,3,2,4,1,5]
>>> isin(mylist,4)
True
>>> isin(mylist,7)
False
>>> isin(mylist,1)
True
>>>
```

## index(inlist,inelement)



```
>>> def index(inlist,inelement):
        index = "Number not in list"
        for i in inlist:
                if i == inelement:
                        index = i
                        break
        return index

>>> mylist = [1,1,3,2,4,1,5]
>>> index(mylist,4)
4
>>> index(mylist,1)
1
>>> index(mylist,7)
'Number not in list'
>>>
```

## Another Challenge

- Write and test a function removeDuplicates(somelist) that removes duplicate values from a list.

## Next week..

- Classes!  Read chapter 10 of the book
- Understanding objects and methods.
- Defining classes in python
- Encapsulation