# Basics on Lists

Programs work with "collections of data"

E.g. Program needs to work with student records

Student record: name, gpa, credit hours, address, club memberships

Most languages use ARRAYS (e.g., C, Java, C++). An array is basically a list. It has a certain size (e.g., "int names[10]" is an array/list of 10 integers in C).

So a Python list can do what a C array can do, AND MORE!

C/Java: each element of the array has to be of the SAME type (i.e., homogeneous)

C/Java: arrays have to be of a fixed size (you define size beforehand, or you ask for more space as needed while the program runs).

Python: lists are dynamic (i.e., no need to state the size beforehand) and lists are heterogeneous (i.e., the elements of the list need not be of the same type; so one item can be an integer, and the next item can be a float).

What we need to learn:

1. How to use lists with "stuff" in them

2. Which functions we can use to "do stuff" with lists

3. To be comfortable programming with lists; they are VERY useful

4. Lists are an example of "Python Sequences". But sometimes, data is stored in a way that is non-sequential. Python DICTIONARIES are useful for this.
# Quick example of why we need to use a list

''' Remember earlier (Lecture #?) we wrote code to work with a sequence of numbers and compute

--- mean, and
--- variance

In that lecture, you learned a clever trick: how to compute mean and variance "on the fly"
that means without storing the numbers anywhere; you used a number as soon as you
read it in or as soon as you computed it).

Now the mean is easy to compute on-the-fly

But to compute the variance, the usual formula needs the mean, because each number
must be subtracted from the mean and the result squared

We then add up all the squared terms and divide by (n-1) to get "unbiased variance".

But in that lecture, we used a trick to get the variance "recursively"
(i.e., the new variance, based on the new data item, used the old variance based on the previous data items and then updated it)

If you did not use that trick (and most text-books do not even talk about it), then you'd have
to compute the mean on the first pass, and store all the numbers on the first pass. Then in the second pass you would find the difference of each number from the mean and then square it etc etc.

...
The text-book example gets the numbers via user-input. In our example, we'll just fill a list with random numbers and pretend that a user input these numbers somehow.

```python
import sys
from math import sqrt
from random import randint

def getlist():
    """ getlist generates a list of random integers ""
    n = int(input("How many numbers in list? "))
    x = []
    for j in range(n):
        x.append(randint(1,500))  # some random integer between 1 and 500
    return(x)

def xbar(nlist):
    """ xbar computes the average of a list of numbers ""
    sum = 0.0
    size = len(nlist)
    for j in range(size):
        sum = sum + nlist[j]
    if (size != 0):
        return(sum/size)
    else:
        print("Error: size of nlist is 0
")```
sys.exit(0)

def s_squared(xbar,nlist):
    """ s_squared computes the variance of a list of numbers, using xbar """
    sum = 0.0
    size = len(nlist)
    for j in range(size):
        sum = sum + (nlist[j] - xbar)*(nlist[j] - xbar)
    if (size != 0):
        return(sum/size-1)
    else:
        print("Error: size of nlist is 0\n")
        sys.exit(0)

def even(n):
    #return True if even; else return False
    if (n%2 == 0):
        return(True)
    else:
        return(False)

def median(nlist):
    """ median returns the median of a list"""
    size = len(nlist)
    if (size == 0):
        print("Error: size of nlist is 0\n")
        sys.exit(0)
    if (even(size)):
        return( (nlist[size//2] + nlist[(size//2) -1])/2)  # even: average of two middle elements of sorted list
    else:
        return(nlist[(size-1)//2 ])
        # odd: middle element of sorted list

def xrange(nlist):
    #we'll use the math lib functions (min and max)
    # and report range as (max - min)
    size = len(nlist)
    if (size == 0):
        print("Error: size of nlist is 0\n")
```python
sys.exit(0)

return(max(nlist)-min(nlist))

def write(nlist):  #write out one list
    for j in range(len(nlist)):
        print(nlist[j])

def write2(alist,blist):  #write out both lists
    for j in range(len(alist)):
        print(alist[j],blist[j])

#HW: The "mode" is the number that repeats most often in the list. If there
#     are no repeat
#     numbers then the list has no mode. Write a simple function to
#     find the mode.

def main():

    mylist = getlist()

    write(mylist)

    m = xbar(mylist)  # get mean
    v = s_squared(m, mylist)  # get variance: notice is the
    2nd pass and you have to feed it m
    sd = sqrt(v)  # get standard deviation

    # We already wrote algorithms to find min and max; there are also library
    # we could use for this. But let's take the easy way now and sort the
    # ascending order. Then we can just read off min and max

    s_mylist = mylist.copy()  #s_mylist is just a copy we make
    s_mylist.sort()  #and then we use list method
    "sort"

    write2(mylist, s_mylist)

    med = median(s_mylist)

    print("Statistics\n")
```
```python
print("Mean:  \{0:0.2f\}\.format(m)\)
print("Variance:  \{0:0.2f\}\.format(v)\)
print(" Std. dev: \{0:0.2f\} ".format(sd))
print(" ")
print(" Min: ",s_mystlist[0],"    Max: ",s_mystlist[len(mystlist)-1])
print("Median:  \{0:0.2f\}\.format(med)\)
print("Range: \{0:02f\}\.format(xrange(mystlist)))\)

main()"

#___________________ save this file as
volunteers.txt_____________________________
#
# make sure last line has a "\n" at end of line followed by no data on next
line, or the read
# data portion of the program will find unexpected data

#The file contains

""" name         height (in feet)         weight (in pounds) """

#of volunteer players who want to join a basketball team

#3.py

#_______________ save only the data below_____________________________

Duck, Daffy       3  5
Fudd, Elmer       3.1 10
Bunny, Bugs       3  5.5
Scrooge, Uncle    4  10
Baggins, Bilbo    4.1 12
Baggins, Frodo    3.9 11
Stooge, Larry     5.5 130
Stooge, Moe       5.5 128
Stooge, Curly     5.5 190
Coyote, Wiley     3.2 15
Runner, Road      2.6  2
Yosemite, Sam     2.8  20
Duck, Huey        2  4.1
Duck, Dewey       2.1  4.2
Duck, Louie       1.9  3.9
Cat, Sylvester    2.83 10
Woodpecker, Wdy   1.5  2
Devil, Tasmanian  1  1
Mouse, Mickey     2.1  2.3
```
Mouse, Minnie  1.9  1.95
Pooh, Winnie      4.39 4.39

# 4.py

''' read in a list of basketball volunteers and select players; use Python
class, objects, sort  '''

''' order the players: descending height list, descending weight list,
descending ht*wt list  '''

''' the top 10 players in each list could make up the team; top 5 actives
and next 5 reserves  '''

class Bplayer:
    def __init__(self, name, height, weight):
        self.name = name
        self.ht = float(height)  # comes in as string, so make it a number
        self.wt = float(weight)

    def getName(self):
        return self.name

    def getHeight(self):
        return self.ht

    def getWeight(self):
        return self.wt

    def getScore(self):
        # score is a product of height and weight
        return (self.ht)*(self.wt)

def makeBplayer(VolunteerString):
    # used by main to make player objects
    name, ht, wt = VolunteerString.split()  # data on line is separated
    # by whitespace (i.e., blanks)

    return Bplayer(name, ht, wt)  # IMPORTANT: It makes and returns
    # a player object!

# some helper functions that we can give to the "sort" function to use as
# the "key" field (i.e., the
# field on which to do the sorting).

# you'll see that we don't have to use these functions if we don't want to.
We can use the methods
# we already have: getHeight, getWeight, etc .... because the functions call
these methods anyway.

def use_ht(aPlayer):
    return(aPlayer.getHeight())

def use_wt(aPlayer):
    return(aPlayer.getWeight())

def use_score(aPlayer):
    return(aPlayer.getScore())

def main():

    filename = "volunteers.txt"
    infile = open(filename, "r")

    playerlist = [ ]

    print("Reading in the volunteer list ................. \n")

    for line in infile:
        p = makeBplayer(line)  # p is now a basketball player object

        print(p.name, p.ht, p.wt, p.getScore())

        playerlist.append(p)

    infile.close()  # now playerlist has all the volunteer objects

    print("_____________________________________________
")

# A basketball team has 5 actives and 5 reserves. How to choose our team? We
# choose based on the 10 tallest, 10 heaviest or 10 with highest score

    htlist = playerlist.copy()

    wtlist = playerlist.copy()

    scorelist = playerlist.copy()
htlist.sort(key = use_ht, reverse = True)  # important: we are sorting based on height, but we
# are passing the "use_ht"
# function via its name to
Notice that we did not say
would have called the
do that. We only want to
do the sort function.

# notice how we sort to get
descending order

print ("Name\t\t\tHeight")
for p in htlist:
    print("{0}\t\t\t{1}".format(p.getName(),p.getHeight()))
    print("________________________________________________________")

# Above we used a helper function use_ht() to sort based on height. But we
did not need to do
# that. Why? Because we already have the function p.getHeight(), when p is a
# Bplayer object.

#So let's try that now to order them in terms of descending weight

wtlist.sort(key=Bplayer.getWeight, reverse = True)  # feed the sort
# function the
# name of the
method directly

print ("Name\t\t\tWeight")
for p in wtlist:
    print("{0}\t\t\t{1}".format(p.getName(),p.getWeight()))
    print("________________________________________________________")

# order based on score = height*weight

scorelist.sort(key=Bplayer.getScore, reverse = True)  # feed the method
sort the method directly

```python
print("Name\t\tScore")

for p in scorelist:
    print("{0}\t\t{1}".format(p.getName(), p.getScore()))
```

main()