Lecture 3

continuing w/ programming
basics of Python (Chapter 2)

need a way to represent
a sequence of numbers.

Python has two ways of doing
this: lists + arrays

begin w/ list
It consists of a list of
quantities of a given type
(elements do not have to be
of the same type)
(but usually we'll have them
be of the same type)

Write a list as

```
[3, 0, 0, -7, 24]
```

Another possibility

```
[1, 2.5, 3+4.6j]
```

(int, float, complex)

can set a variable to a list

```
r= [1, 2.5, 3+4.6j]
```

can print:

```
print(r)
```

can also do

```
x=1.0
y=1.5
z=-2.2
r=[x, y, z]
```

remember that Python evaluates RHS + then assigns to LHS (so if you change x later, then r won't change)

can also calculate elements of a list from mathematical expressions as

r= [2*x, x+y, z/sqrt(x**2+ y**2)]

you can access the elements of a list via

r[0], r[1], r[2], etc.

counting starts @ zero.

Bring up 1-list-sqrt.py

can change elements of a list via

```
r= [1.0, 1.5, -2.2]
r[1] = 3.5
print(r)
```

Built-in functions to operate on lists:

sum, max, min, len

Bring up 2-list-avg.py

How to apply the same function to every element of a list?

use the "meta function" map

Suppose we would like to take the logarithm of every number in the list. Then we would run

```
map(log, r)
```

creates an object called "iterator" in computer memory.
can convert this to another list.

Bring up 3-log-map.py

What if we want to add an item to a list?
E.g., add 6.1 to list r, then

```
r.append(6.1)
```

simple example of Python's object-oriented programming features

can also have mathematical functions in append function, as in

```
r= [1.0, 1.5, -2.2]
x = 0.8
r.append (2*x+1)
 print(r)
```

can also create an empty list w/

```
r= [ ]
```

& then can add later

```
r.append (1.0)
r.append (1.5)
r.append (-2.2)
print (r)
```

need to create a list
before adding to it, or you
get an error.

can remove an item from the
end of list w/

r.pop()

could also remove an element @
given location w/in list w/

r.pop(n)

but generally avoid for large
lists due to slow operation.

Alternative to list is an array. Differences w/ list:

1) # of elements in array is fixed. Cannot add or remove

2) elements must all be of same type. cannot mix & cannot change later

Advantages of array over list:

3) can be 2-dimensional

4) they behave like vectors or matrizes. can add & subtract them

5) much faster than lists.

In physics, we often will use a fixed array of elements, all of same type. We will work w/ arrays far more often than lists.

Array functions are part of package numPy

Bring up 4-create-array.py

could also do

zeros (10, int) or

zeros (10, complex)

similar function in numpy called ones

It takes time to set the initial values of elements of array, but if we don't need zeros to begin w/, just use

```
from numpy import empty
  a = empty(4, float)
```

just initializes to whatever is there in memory

can take a list and convert to an array w/

```
r = [1.0, 1.5, -2.2]
a = array (r, float)
```

can also convert array to list w/

```
r = list(a)
```

can create a 2D array from a list of lists w/

```
a= array( [[1,2,3],[4,5,6]], int)
print(a)
```

to access elements of 2D array, use

```
a[0,1]    or    a[1,0]
```

can also set them w/

```
a[0,1]= 1        a[1,0] = -1
```

Reading an array from a file

special function in numpy called loadtxt

Bring up 5-load-array.py

can also read in a 2-D array of numbers. file w/

```
1234
3456
5678
```

could be read in w/ loadtxt

---

can do arithmetic w/ individual elements of array as

```
a[0] = a[1]+1
```

But you can also do arithmetic w/ whole array

Bring up 6-arith-array.py

can also add arrays of the same size w/ a+b

Multiplying two vectors does not give inner product but instead gives Hadamard-Schur product (elementwise)

Bring up 7-mult-array.py

can also do these kinds of calculations w/ matrices. Suppose we would like to compute

$$\begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} 4 & -2 \\ -3 & 1 \end{bmatrix} + 2 \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} -3 & 5 \\ 0 & 2 \end{bmatrix}$$

can also multiply matrices & vectors together w/ dot use dot(a,v) for

$$\begin{bmatrix} a \end{bmatrix}\begin{bmatrix} v \end{bmatrix}$$

&

dot(v,a) for

$$\begin{bmatrix} v \end{bmatrix}\begin{bmatrix} a \end{bmatrix}$$

can apply sum, max, min, len to arrays & can use map as well to apply to all elements of array. I.e., suppose we want sqrt of elements. Then do

```
b = array (map(sqrt,a), float)
```

can get size & "shape" of array w/ commands

Bring up 9-array-size-shape.py

getting average of a set of values

mean square . . . – –

calculating geometric mean

$$\bar{x} = \left[\prod_{i=1}^{n} x_i\right]^{1/n}$$

But this is equal to

$$\exp\left(\log\left(\bar{x}\right)\right) =$$

$$= \exp\left[\,1/n \;\log \prod_{i=1}^{n} x_i\right]$$

$$= \exp\left\{\,1/n \sum_{i=1}^{n} \log x_i\right\}$$

(only works for positive #s)

Bring up 10-means-arith-geo.py

could also use the log function from the numpy package

Bring up 11-alt-geo.py

Word of caution about working w/ arrays

given program:

```
from numpy import array
a = array([1,1], int)
b = a
a[0] = 2
print(a)
print(b)
```

output is

```
[2 1]
[2 1]
```

Why?

You might think it would be

$$\begin{bmatrix} 2 & 1 \end{bmatrix}$$
$$\begin{bmatrix} 1 & 1 \end{bmatrix}$$

but what Python does w/ b=a is to

set b to be another name for a

It does not copy contents of a into b. (similar to pointers in C)

to copy just do

```
from numpy import copy
b=copy(a)
```

Slicing out elements from a list or array. ~~want~~ to select certain elements.

Bring up 12-slicing.py

variants of this, including

```
r[2:]
r[:5]
```

Slicing works w/ arrays or 2-D arrays

`a[2:4, 3:6]`

gives you subblocks of a 2D array a

`a[2,:]` gives the whole of row 2 of array a

For loops

Most common kind of loop that we will use

Bring up 13-simple-for.py

program 1st makes list
for each value in list, executes
& goes to next value in list

Indentation is important here

can use "break" & "continue"
in a for loop to break out
or continue ~~to~~ next iteration

for long lists w/ obvious values to loop through, you can use the range function.

Bring up 14-range-for.py