

Lists

Lists

Look at lists and how to process multiple variables:

1. Basics of Lists
2. Creating Lists
3. Accessing Lists
4. Manipulating Lists
5. More on Lists

Basics of lists I

Up to now, we have been using variables that have a single value e.g. float, complex, int

Lists allow you to:

- Group a collection of items (elements) in an ordered way
- Access and process list elements in a simple way
- Pass list of data to functions
- Read and write bulk data to/from files

This is the basic way that you handle “bulk” data.

Basics of lists II

- Example: create a list of floats and access elements

```
vals = [1.0, 4.5, 7.0, 8.3, 9.3]  
print("Fourth element is : " + str(vals[3]))
```

- Creates a list of floats, with elements called `vals[0]`, `vals[1]`, `vals[2]`, `vals[3]`, `vals[4]`
- Note that the numbering (list index) starts at 0
- Can use the elements of the list as variables e.g.

```
y = vals[0]*vals[4] # Use elements as variables  
vals[1] = y + 4.0 # Update the values of elements
```

Creating lists

- Create a list using values

```
vals = [1.0, 4.5, 7.0]
```

- Create a list using variables

```
x = 10.0  
y = 13.0  
z = 16.0  
v = [x,y,z]
```

- Create a list where all the elements have the same value in 'bulk'
 - often used to initialise list elements

```
v = [0.0]*25
```

- Create an empty (blank) list then append values to it
 - often done using a loop

```
x = 10.0  
y = 13.0  
z = 16.0  
v = []          # A blank list  
v.append(x)  
v.append(y)  
v.append(z)
```

- Creates 25 floats all set to 0.0, numbered 0 -> 24

Accessing lists

- Access list elements using `[i]` syntax, where `i` is an `int`. For example:

```
mydata = [1.0, 4.5, 7.0, 8.3, 9.3]
total = 0.0
for i in range(len(mydata)):
    total = total + mydata[i]**2

print("Total sum squared " + str(total))
```

- This is the conventional way to access list elements, and is similar to other languages (C, C++, Java)

Accessing lists: the python way

- As accessing list elements is very common. there is a faster Python way

```
mydata = [1.0, 4.5, 7.0, 8.3, 9.3]
total = 0.0
for x in mydata:
    total += x**2

print("Total sum squared " + str(total))
```

- This loops through the list `mydata` with `x` taking the value of each element of the list in turn
- This is much more efficient than using `for i in range`, but it is less obvious what the code is doing

Manipulating lists

- Append element to the end

```
vals = [1.0, 4.5, 7.0]  
a = 6.0  
vals.append(a)
```

- Extend `vals` by adding another list to the end

```
u = [7.5, -0.8, 10.0]  
vals.extend(u)
```

- Get length of list (the number of elements)

```
l = len(vals)
```

- Get 'slice' of list (copy of subset of elements)

– slice returned as a new list

```
w = vals[3:7]
```

- Note that `vals[start:end]` contains `vals[start]` to `vals[end-1]`
- So in this example, list `w` contains 4 elements, `vals[3]` to `vals[6]`

More on lists

- These examples used lists of floats, but lists are much more general. Lists can be:
 - list of int, complex, boolean, str etc.
 - list of lists
 - list of objects (see optional section on objects and OOP at end of this course)
- A list can even contain different type of objects in the same list...
- ...but this is a really dangerous feature since it is up to YOU (the programmer) to keep track of what type each element is. Just don't do it!
- In more advanced programming, it is better to use the NumPy library to deal with long lists of floats
 - see e.g. Computer Simulation or Computer Modelling courses.