Scientific Computing: Lecture 2

- Slicing and modifying sequences (lists, tuples, strings)
- Conditionals
- Loops
- More on Functions
- Practice Exercises

CLASS NOTES

- HW#00 due next Thursday (2/4).
- Instructions for turning in HW
- Read Chap. 1 by Wednesday
- Download class code and start altering it.

Turning in HW

- Most assignments will consist of DOCUMENTED code and output generated by that code (plots, etc..).
- When your code is complete, save with the following filename template: YourLastName_AssignmentNumber_FileContent.xx
  - For example: `Gladden_HW01_code.py` and `Gladden_HW01_plot1.png`
- Also save your .py file as a .pdf file.
- Upload the files into the appropriate folder in the Box share (NOTE: I switched to Box because it has better permission control).
- The TA for the course is Khagendra Adhikari and his email is kadhikar@go.olemiss.edu. Khagendra will be grading HW assignments.

Slicing sequences

- A ‘sequence’ is a collection of objects like a list or tuple.
- Often one needs part of a sequence. This is called ‘slicing’ in python.
- Important: counting in python starts at 0 (not 1)!
- `range` is a convenient function to build a sequence of integers: `range(start,end,step)` where `step` is assumed 1 if not specified, and `end` is NOT included!
- Strings are considered a list of characters, so slices work for them too.

```
mylist=range(10,100,2) # note list ends at 98
mylist[1] #returns 12 (NOT 10)
mylist[0:3] #returns [10,12,14]
mylist[3:5] #returns [16,18]
mylist[-1] #returns 98, (-) means count from end
mylist[:5] #returns [10,12,14,16,18]
filename = ‘pressure_T298.dat’
extension = filename[-3:] #returns ‘dat’
temp = float(filename[-7:-4]) #returns 298 as a float
```

Combining sequences

- Lists and tuples can be combined with a ‘+’ operator: `[1,2,3] + [4,5,6]`
  returns `[1,2,3,4,5,6]`
- Works with strings too: `‘Hello ’+‘World’` returns `‘Hello World’`
- Lists can be modified with these **methods**
  Say **`mylist=[“Billy”, ”Sue”, ”Jimmy”]`**
  - `mylist.append(item)` # sticks item on the end
  - `mylist.insert(index,item)` #inserts item at location index
  - `mylist[2]=‘Tommy’` #replaces the 3rd item with ‘Tommy’
  - `mylist.pop()` # returns the last item and <u>removes it from list</u>
  - `mylist.sort()` # sorts the <u>original</u> list (numerical or alphabetical).
  - `mylist.remove(item)` # remove the item from the list
  - `mylist.index(item)` # return (first) index of the item

Basic Input and output (I/O)

- Input from users:

  `item = raw_input('Enter a number: ')`

  raw_input always returns a string.   OR

  `n=input('Enter a number:')`

  input expects a python expression!
- `print` puts output to the screen

```
name='Sarah'
age=27
string = "My name is %s and I am %i years old" % (name,age)
print string
```

- %s: string, %i: integer, %g: float (more later).

`Print` in Python 3

- print has been changed from a statement to a function in Python 3: `print “Hello World”` -> `print(“Hello World”)`
- New options with (optional) keyword arguments:
  - `print(“Something”,end=“ “)` – end tells python how to end the line. Default is a newline. This would enter a space and the next print call would come on the same line.
  - `print(“Something”,file=openfile)` – this would write the string to an open file object (more on this later) rather the screen.
  - `print(“Something”,”something else”, sep=“-”)` – this would separate each string with sep. This would output: `Something-something else`
- Python 2.7 (which we are using) recognizes the print function, but not the keyword arguments listed above.

Conditionals in Python

- Conditionals tests whether a condition is True or False

- Use "Boolean" operators:
==, <, >, <=, >=, !=, is, is not, in

- What is "True" in Python: True, 1, any non-empty sequence, any previously defined variable

- What is "False" : False, 0, any empty sequence

```
1>0
ilist=range(0,100,2)
newlist=ilist
2 in ilist
3 in ilist
newlist is ilist
```

Conditionals in Python

- Use ‘if’ or ‘if not’ statements to make decisions:

```
num = 500
if num >= 1000:
    print “You have a big number!”
```

- Use “else if” or “elif” statements to catch alternatives

```
num = 500
if num >= 1000:
    print “You have a big number!”
elif num >=100 and num <= 1000:
   print “You have a medium number”
else:
   print “You have a small number”
```

- Note you can string conditions together with “and” or “or”.

  “and” -> **all** conditions must be True

  “or” -> **any** of the conditions must be True

“for” Loops in Python

- Loops perform repetitive tasks.

- Most common types are “for” and “while” loops.

- Useful function in loops is “range()” – generates a list of numbers.

- Examples:

```
alist=[1,2,3,4]
for i in range(len(alist)):
   alist[i] = alist[i]**2
#OR use “list comprehension”
[ i**2 for i in alist ]
```

```
elements = [‘Au’, ‘H’, ‘He’, ‘C’]
weights = [35, 1, 2, 12]
for element in elements:
   print element
for element,weight in zip(elements,weights):
   print ‘Weight for %s is %i’ % (element,weight)
```

“While” Loops in Python

- “while” loops tests and repeats until a condition is True

- while True > do something >
change what is tested > repeat

- Examples:

```
time=0
y=0
while time <= 20 and abs(y) <= 500. :
    y=-9.8*time**2
    print 'At time = %g, the ball fell %g m' % (time,y)
    time += 1    #short for time = time +1.0
```

- Real code….

Continuous “While” Loops

- Sometimes convenient to keep looping forever until some arbitrary condition happens

- Use “while 1, break” method (remember ‘1’ is always True)

- while True > do something >
change what is tested > repeat

```
while 1:
   num = eval ( raw_input(‘Type a number > 0 (0 to quit) : ‘))
   if num ==0:
       print ‘Quitting this game …’
       break
   else:
      print “Cube of %3.2f is %4.2f. “ % (num, num**3)
```