

Introduction to Programming for Scientists

Prof. Steven Ludtke
N410.07, sludtke@bcm.edu

Lecture 2:
Conditions, Loops & Variables

Reminder

Class material at:

<http://blake.bcm.edu/IP15>

If you missed the first lecture, it is archived on the site above.

Python

Data storage

'simple' types - numbers, strings, ...

compound types - lists, dictionaries, sets, ...

Operate on data

statements - `a=b*10`, `print b*5+3`, ...

functions - `sin(a)`, `len(x)`, ...

methods (functions on an object) - `"abc".count("b")`

Program Flow

`for ... in ...`

`if, else`

`while ()`

Interact with the outside world

User interactions - `raw_input()`

Disk and other device access - file i/o

Type conversion

Type	Creation	Conversion
integer		int()
long integer	L suffix	long()
floating point	decimal point	float()
boolean	True or False	bool()
complex	x+yj	complex()
string	" ", ' ', """ """, ''' '''	str()
list	[a,b,...]	list()
tuple	(a,b,...)	tuple()
set	{a,b,...}	set()
dict	{a:b,c:d,...}	dict()

Python

Data storage

'simple' types - numbers, strings, ...

compound types - lists, dictionaries, sets, ...

Operate on data

statements - `a=b*10`, `print b*5+3`, ...

functions - `sin(a)`, `len(x)`, ...

methods (functions on an object) - `"abc".count("b")`

Program Flow

for ... in ...

if, else

while ()

Interact with the outside world

User interactions - `raw_input()`

Disk and other device access - file i/o

Functions vs. Methods

Functions : `sin(x)`, `cos(y)`, `len(s)`

normally return a value

Not type-specific

Methods : `st.upper()`, `lst.append(5)`, `lst.sort()`

functions applied to a specific “object”

don't always return anything

methods are type-specific

Some Built-in functions

input & raw_input

range, xrange - makes a list (or iterator) covering a range

len

max,min

reversed, sorted

print (actually a statement in Python2)

Methods of Strings

Remember strings are immutable !

upper, lower, title, capitalize

count, find, rfind, index

replace

split

join

in (not really a method)

Lists

```
[item1,item2,item3,...] # items can be anything
(item1,item2,item3,...) # A tuple is an immutable
list
a=[0,1,2,3,4,5,6]      # A list of 7 numbers
a[n]                   # nth element in list
a[n:m]                 # sublist elements n to m-1
a[-n]                  # nth item from the end
a[3] -> 3
a[1:4] -> [1,2,3]
a[-2] -> 5
a[2:-2] -> [2,3,4]
a[2]="x" -> [0,1,"x",3,4,5,6]
```

List Methods

append, extend

del, remove

count

index

reverse, sort

Sets

Sets have no order and elements are unique

```
set([1,2,3,4,5])
```

methods:

add, remove, discard, clear

issubset, issuperset

union, intersection, difference

Dictionaries

keys must be immutable, values can be any type

```
{ k1:v1, k2:v2, k3:v3, ... }
```

Example:

```
a={ 1:2,2:3,"a":"b",2.0:3.2,(1,2):"really?" }
```

```
a[1] -> 2
```

```
a[(1,2)] -> "really?"
```

```
a[2] -> 3.2
```

Methods:

- keys, values, items

- has_key

- set_default

Programs you can Run

Do NOT use a word-processor like Word, Pages, etc. Use a 'text editor'. The built-in editor 'idle' is a good choice for beginners.

Just type 'python program.py' -or- :

use a '.py' extension for your programs

for unix/mac, put:

```
#!/usr/bin/env python
```

on the first line of the file, and type:

```
chmod a+x file.py
```

NOTE: on windows, as soon as the program exits, the window showing the output will close. If you put a `raw_input()` at the end of your program, it will wait until you press enter before closing the window so you can see the output.

Writing Actual Programs

How would you display a table of x vs $\sin(x)$ for x from 0 to 2π in steps of $\pi/4$?

Suboptimal, but functional

```
from math import *  
  
print 0, sin(0)  
print pi/4, sin(pi/4)  
print pi/2, sin(pi/2)  
print 3*pi/4, sin(3*pi/4)  
print pi, sin(pi)  
print 5*pi/4, sin(5*pi/4)  
print 3*pi/2, sin(3*pi/2)  
print 7*pi/4, sin(7*pi/4)  
print 2*pi, sin(2*pi)
```

Umm... slightly better ?

```
from math import *  
x=[0,pi/4,2*pi/4,3*pi/4,4*pi/4,5*pi/4,6*pi/4,7*pi/4,8*pi/4]  
  
print x[0],sin(x[0])  
print x[1],sin(x[1])  
print x[2],sin(x[2])  
print x[3],sin(x[3])  
print x[4],sin(x[4])  
print x[5],sin(x[5])  
print x[6],sin(x[6])  
print x[7],sin(x[7])  
print x[8],sin(x[8])
```


Now What ?

How would you display a table of x vs $\sin(x)$ for x from 0 to 2π in steps of $\pi/64$?

Python

Data storage

'simple' types - numbers, strings, ...

compound types - lists, dictionaries, sets, ...

Operate on data

statements - `a=b*10`, `print b*5+3`, ...

functions - `sin(a)`, `len(x)`, ...

methods (functions on an object) - `"abc".count("b")`

Program Flow

for ... in ...

while ()

if, else

Interact with the outside world

User interactions - `raw_input()`

Disk and other device access - file i/o

Program Flow

for i in list:

if condition :

 Boolean operators

 >, <, <=, >=, ==, !=, and, or, not, in

elif condition :

else :

while condition :

for Loops

Execute 'code' for each item in list, assigning the element to 'var' in each cycle:

```
for var in list:
```

```
    code
```

Example:

```
a=[1,2,3,4,5]
```

```
for i in a:
```

```
    print i,i*2
```

Blocks of Code

```
j=0
```

```
for i in x:
```

```
    j=j+i
```

```
    print i,j
```

```
for i in x: print i
```

Umm... slightly better ?

```
from math import *  
x=[0,pi/4,2*pi/4,3*pi/4,4*pi/4,5*pi/4,6*pi/4,7*pi/4,8*pi/4]  
  
print x[0],sin(x[0])  
print x[1],sin(x[1])  
print x[2],sin(x[2])  
print x[3],sin(x[3])  
print x[4],sin(x[4])  
print x[5],sin(x[5])  
print x[6],sin(x[6])  
print x[7],sin(x[7])  
print x[8],sin(x[8])
```

Better !

```
from math import *  
x=[0,pi/4,2*pi/4,3*pi/4,4*pi/4,5*pi/4,6*pi/4,7*pi/4,8*pi/4]  
  
for i in x:  
    print i,sin(i)
```

More improvement...

```
from math import *  
x=range(9)
```

```
for i in x:  
    j=i*pi/4  
    print j,sin(j)
```


Would this work ?

```
from math import *  
x=range(9)  
for i in x: i=i*pi/4  
  
for i in x:  
    print i,sin(i)
```

Try this instead

```
from math import *  
x=range(9)  
for i in range(len(x)): x[i]*=pi/4  
  
for i in x:  
    print i,sin(i)
```

Works, but not very satisfying...

List Generators

A for loop inside a list definition !

```
[x... for x in y]
```

example:

```
a=[0,1,2,3,4,5,6,7]
```

```
a=[i**2 for i in a]
```

```
print a
```

```
[0,1,4,9,16,25,36,49]
```

Much Better !

```
from math import *  
  
for i in [i*pi/4 for i in range(9)]:  
    print i, sin(i)
```

Writing Actual Programs

How would you display a table of x vs $\sin(x)$ for x from 0 to 4π in steps of $\pi/8$, but only include values where $\sin(x) > 0$?

Start with this

```
from math import *  
x=[i*pi/8 for i in range(33)]
```

```
for i in x:  
    print i,sin(i)
```

But what about the $\sin(x) > 0$ requirement ?

The if statement

Boolean operators

>, <, <=, >=, ==, !=, and, or, not, in

if condition :

elif condition :

else :

Tack in our if

```
from math import *  
x=[i*pi/8 for i in range(33)]  
  
for i in x:  
    if sin(i)>0 :  
        print i,sin(i)
```


Comments

Anything after '#' on a line is a comment

Tack in our if

```
from math import *
# This generates our x-values
x=[i*pi/8 for i in range(33)]

for i in x:
    if sin(i)>0 :
        print i,sin(i)
# loop over x-values
# only print if sin(x)>0
```

1-line solution

```
from math import *
```

```
print "".join(["{:1.2f}\t{:1.2f}\n".format(x*pi/8,sin(x*pi/8)) for x in range(33)])
```

Homework #1

For all homework assignments, you are free to consult others on concepts, but the final code you turn in should be your own. If you are just learning programming for the first time, I would suggest that you try to spend at least 30 min thinking about each problem before seeking assistance. Even then, the first few assignments may be very frustrating and time consuming, but if you don't practice the fundamentals now, you may be in real trouble later in the class. The only way to learn programming is by doing it. There are many possible solutions to each of these problems. If you need help, you can contact the TA or email me at any time, or find me any time my office door is open (mornings are usually better). We will go over the solutions at the beginning of class each Friday, so the homework must be emailed to the TA before then !

To hand in your homework: For each problem, create a ".py" file containing the program that solves it. This should not be a Word doc, or a PDF, but a text file you could execute directly. Use comments to document your programs!

Submit homework by email with the subject "Homework N" to koire@bcm.edu with a cc to sludtke@bcm.edu.

Practice #1

The homework (next page) is due next Friday. Before the Lab next Monday, it would be a good idea to get a little practice with data types, conversion and manipulation. These small exercises should not be turned in, but I strongly encourage you try and do them yourself over the weekend to gain a little experience. Set aside an hour and try to solve each one for ~10 minutes. After trying yourself, download the solutions and make sure you understand them all:

- 1) Create a list of numbers from 5 to 15 inclusive stepping by 0.5.
- 2) Start with the string "this is a short test string" and create a new string with the letters sorted alphabetically
- 3) Create a string containing only the unique letters in "abracadabra"
- 4) Start with `s="1 2 4 8 16"`. Convert the string to a list of integers and take the log base 2 of each number.

Homework #1

Ask the user to enter a 1-letter DNA sequence, for example
"CTGGGCCACACTGGAAGAACTGTGTTGGGCCACA"

Count the number of each nucleotide present in the entered sequence
(and print the count)

Print the reverse complement of the entered sequence

Write a program to identify the winner of a rock,paper,scissors game. Ask the user what player 1 picked (rock, paper or scissors), then ask what player 2 picked. Finally, print the winner (player 1, 2 or tie)