Python Programming 1
variables, loops, and input/output

Biol4230 Thurs, Feb 1, 2018
Bill Pearson wrp@virginia.edu 4-2818 Pinn 6-057

- A quick introduction to Python
  - Running python
  - Variable types: `scalars, arrays[]=[0,1,2], tuples[]=(1,'pi',3.12), hashes[]={key:value}`
  - Flow control: `if () then: else:, for:, while:`
  - Input/output and `print; fileinput, open()`
  - Useful python functions:
    `.split(), .join(), .strip('\n')`
- Programming – a problem solving approach

To learn more:

- Practical Computing: Part III – ch. 7 – 10
- Learn Python the Hard Way: learnpythonthehardway.org/book/
- Think Python (collab) www.greenteapress.com/thinkpython/thinkpython.pdf
- Exercises due noon Monday, Feb. 5 (save in biol4230/hwk3)
  1. Write a program to generate 10 random numbers between 0 and 100 (0 <=x<100), calculate the mean (average). Print both the random numbers and the mean to a file.
     a. Make a program that calculates the average of 100 random "real" numbers between 0 and 100 (0 <= x < 100)
     b. write a program that generates 101 random numbers, stores them in an array, and calculates the median (hint, use the sort function to sort the array, then report the value of the middle).
  2. write a program to read a file of Uniprot accession strings and download the sequences in FASTA format to "stdout"
  3. Repeat steps 8 – 10 of last week's bash script homework using python programs (see last slide) to isolate the range of E()-values of interest

Introduction to python

- Variable types:
  - (in bash scripting, variables are $n, and not typed)
  - floats, ints, strings
  - arrays=[], tuples=() of floats, ints, strings
  - dict's (hashes) `fields = dict(zip(names, data))`
- Control structures:
  - `for x in list :` (bash do; done)
  - `if (true) : ; elif (true): ; else :`
    `# ; only for multiple statements on one line`
  - statement blocks are indented (no "done", "}", or "fi")
- Functions are often applied to variables
  - `array.sort()`
  - `string.strip("\n"), string.split("\t")`
- Some functions are "imported"
  - `import fileinput`
    `for line in fileinput.input()`
  - `from urllib import urlopen`
    `print urlopen(url_string).read()`

Running Python

Running a script:

$ **python myscript.py**

Spontaneous Python:

$ **python**

>>>print "Here we are."

Here we are.

>>><ctrl-D>

Executable scripts:

$ **chmod +x myscript.py**

$ **myscript.py**

Literals: strings and numbers

```
$ python
Python 2.7.11 |Anaconda 2.4.0 (64-bit)| (default, Dec  6 2015, 18:08:32)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-1)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
Anaconda is brought to you by Continuum Analytics.
Please check out: http://continuum.io/thanks and https://anaconda.org
>>> print 2+2
4
>>> print "2+2=",2+2
2+2= 4
>>> print "2+2='; print 2+2
  File "<stdin>", line 1
    print "2+2='; print 2+2
                          ^
SyntaxError: EOL while scanning string literal
>>> print "2+2="; print 2+2
2+2=
4
>>> print "2+2=",; print 2+2
2+2= 4
```

Practical Computing, Ch. 8

Literals: strings and numbers

```
# string "addition" (concatenation operator)
>>> print "one + two " + "three"
one + two three
# mixing numbers and strings:
print "3 * 3 = "+ (2 + 2)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int' objects
>>> print "3 * 3 = ", (2 + 2)
3 * 3 =  4
>>>
# decimals and concatenations:
>>> print 2.3 + 2, 2 + 2., 2 + 2, 7/2, 7.0/2
4.3 4.0 4 3 3.5
>>> print 2.3 + 2
4.3
>>> print 2 + 2.
4.0
>>> print 2 + 2
4
>>> print 7/2, 7/2.
3 3.5
```

python values are "typed" cannot add (+) a string and a number

works because no (+), just another argument

floating point

integer

both

Practical Computing, Ch. 8

Python vs. bash scripts

- "`.py`" file extension, e.g. "`myScript.py`"
- begins with a "shebang"

  `#!/bin/env python`
- "`.py`" scripts need `chmod +x` to be executable:

  invoked with python: `python myScript.py`

  or directly: `./myScript.py`

Minimal python

- Variables:
  - `simple`
  - `array = (1,2,3,4,5); array[0] == 1;`
  - `dict = {'f_name':'Bill', 'l_name':'Pearson'}; dict['f_name'] == 'Bill'`
- Loops:
  - `while (condition):`
  - `for acc in accs :`
  - `break; continue;`
- Conditionals:
  - `if (condition1) : elif (condition2) : else:`
  - `if (line[0] == '^') : continue;`
- Python loop and conditional code blocks are specified with *indentation* only (a ':' requires indentation; block ends when indentation is done)
- Input/Output:
  - `import fileinput`
  - `for line in fileinput.input:`
    `process(line)`
  - `fd = open("my_data.dat",'r')`
    `for line in f:`
    `process(line)`
  - `print "\t".join(array);`

Practical Computing, Ch. 8-10

Python variables

- Like many scripted languages, python has several data types (numeric, sequence, set, class, etc). We will be using three in this class:
  - numeric (integers and floats) `four=4; pi=3.14`
  - sequences (strings, arrays, tuples), indexed starting at 0
    `seq="ACGT"; print seq[1];` strings are immutable (you can change the entire string, but not parts of it)
    `arr=[1,4,9,16,25]; print arr[2]`
    `num = 1;` and `num_str='1';` are different
    `tuple = (1, 3.13159, 'pi');` tuples are "immutable" (cannot be changed)
  - dicts (key, value pairs, aka "hashes")

```
seq_entry = {"acc":"P09488",
        "seq":"MPMILGYWDIRGLAHAIRLL"}
print seq_entry["acc"]; print seq_entry["seq"][0:3]
```

- Variables are not declared in advance; scalars (numerics), sequences (strings, arrays), and dict {} variables all look the same. Consider using naming conventions to distinguish them.

Practical Computing, Ch. 8

our first Python script: `myscript.py`

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

import sys

print sys.version

name="Bill"

print "my name is: "+name
```

Tell the shell this is a python script

use sys functions

print the python version

assign the string "Bill" to the variable "name"

print out the label and variable "name"

our first Python script

```
$ myscript.py
2.7.11 |Anaconda 2.4.0 (64-bit)| (default, Dec  6 2015, 18:08:32)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-1)]
my name is: Bill

$ chmod +x myscript.py
```


Python can act like bash

```
#!/bin/env python

import subprocess
accs=['P09488', 'P28161', 'P21266', 'Q03013', 'P46439']

for acc in accs:
  subprocess.call("curl --silent http://www.uniprot.org/uniprot/" +
                  acc + ".fasta", shell=True)
```

use subprocess functions (call)

why "acc", not "accs"?

Python can be a web-browser

```
#!/bin/env python

from urllib import urlopen
base_url = 'http://www.uniprot.org/uniprot/'
accs=['P09488', 'P28161', 'P21266', 'Q03013', 'P46439']

for acc in accs:
  print urlopen(base_url + acc + '.fasta').read(),
```

use one urlopen function (urllib)

why use "base_url"?

arrays and tuples (lists)

```
list=[1,2,3,4,5]; #
tuple=(100, 3.14159, "Pi");  # three different types, tuples
   are "immutable"; they cannot be assigned to:
   tuple[1] = 2.718281;  # illegal
nt=['a','c','g','t'];   # DNA
pur=['a', 'g']; pyr=['c', 't']
nt = [pur + pyr] == ['a','g','c','t']

nt2 = [pur, pyr] == [['a','g'],['c','t']]
# lists do not "flatten"

a = 'a'; c='c'... # what is the difference between a and 'a'
nt=[a, c, g, t];  # interpolation
[a, c, g, t] = nt;      # assigning to lists
lines = lots_of_lines.split("\n");
words = lots_of_words.split(" ");

# strings are sequences, like arrays, but an array of
   characters is not a string.
# strings, arrays, and tuples are indexed starting at 0:
arr = [1,2,3,4,5];  arr[0] == 1;  arr[length(att)-1] == 5;
```

Practical Computing, Ch. 9

python operators

- Arithmetic: addition, subtraction, multiplication, division, modulus (remainder)

```
a = 2 + 2;
a = 2 + 2 * 2; a = 2 + (2 * 2);
# operator precedence, use parens
c += (a + b) # increment by (a+b)
# division by integer != division by float
7/2 == 3;   7/2.0 = 3.5;   # python 2.7 vs 3
```

- Comparison

  `>, >=, ==, !=, <=, <` for numbers and strings
- Logical: and `'and'` or `'or'` not `'not'`
- python variables preserve 'type':

```
print 1 + 2  == 3
print '1'+'2' == '12'
print 1 + '2' == error
```

Practical Computing, Ch. 8

Flow control: `if : else :`

- `if:/elif:/else:`

```
[sig, border, not_sig] = [0,0,0];
   if (evalue <= 0.001):
      sig += 1
   elif (evalue <= 0.1):
      border += 1
   else:
     not_sig += 1
```

- python uses ':' with indentation, not { }, to define statement blocks
- Conditions can be connected by "boolean" operators:

```
if (evalue > 0.001 and evalue <= 0.1):
    border += 1
```

Practical Computing, Ch. 9

Flow control: for : loops

```
sum = 0; # always remember to initialize
for value in array_of_values:
    sum += value
```

python does not have the traditional C/java/perl/fortran indexed loop of the form:

~~`for (i=0; i<n; i++){sum += array[i];}`~~

instead, you can use `range()` or `enumerate()`

```
for i in range(length(array_of_values)):
    sum += array_of_values[i]
```

or

```
for index, value in enumerate(array_of_values):
    sum += array_of_values[index]
    sum_v += value   # sum and sum_v get the same values
```

but mostly, you do not use the index unless you need to look to the previous/next entry in the array

Practical Computing, Ch. 9

Flow control: while () {} loops

```
while ( line = fd.readline()):
  line = line.strip('\n') # always remove "\n" first
  columns = '\t'.split(line)
```

- '`continue`' skips the rest of the loop

```
for line in fileinput.input() :
  # match '>' at beginning of line, FASTA header
  if (line[0] == '>'):
    continue
  # do something to sequence lines
```

- '`break`' exits the loop

```
for line in fileinput.input():
  line = line.strip('\n') # always remove "\n" first
  columns = '\t'.split(line)
  if (columns[-2] > 0.001) :
    break
```

Practical Computing, Ch. 9

Input/Output I

- Data is read with the `fileinput.input()` function or by opening a file object `fd=open("filename")`

```
import fileinput
for line in fileinput.input()
```

  - If you put a file name on the command line, `fileinput` reads from the file
  - If you put a list of files `fileinput` reads them in order
  - If you don't put a file, `fileinput.input()` uses `stdin`

-
```
fd = open("filename")
for line in fd :
fd.close()   # files should be closed after reading
```

- Lines read from files have "\n" at the end; always remove it

```
line = line.strip('\n')
```

- If only one file read at a time, and one file for output, use `fileinput.input()` for input and '>' on the command line to write to `stdout`.

Practical Computing, Ch. 9, p 177

Input/Output II

- Files can also be opened for writing ('>' or '>>' – extend at end)

```
out_fd = open('outfile.name', 'w') # open file for writing
```

- To send data to a file (or stdout), use 'print'

```
print "accn:",prot_acc, "evalue:", evalue  # also
print "accn: %s evalue: %g" % (prot_acc, evalue)
```

Goes to `stdout` (> on the command line)

```
out_fd.write("accn: %s evalue: %s\n" % (prot_acc, evalue))
```

Goes to `'outfile.name'` because of `open()` above

- `.write()` lines (unlike `print` lines) always need `"\n"`
- If input is `<tab>` delimited, output often should be as well.

```
print "\t".join(map(str,(query_acc, subj_acc, evalue)))
```

- `out_fd.close();`

"map" list (array) to str (string) type

Practical Computing, Ch. 9, p 182

Input/Output Examples

Count lines in a file:

```
#/usr/bin/python
import fileinput
c = 0
for line in fileinput.input() :
  c += 1
print c
```

```
$ python count_lines.py gstm1_human
5
```

```
$ wc gstm1_human.aa
  5  14 311 gstm1_human.aa
```

Summarize blast output with python

Problem – write a python script to identify distant homologs, and re-search swissprot with those sequences

- Did problem before with bash

```
blastp –outfmt 6, cut –f 2
```

- With python, we can look at the expectation value to find distant homolog candidates (`0.1 < evalue < 2.0`)

python problem solving – initial steps

1. Look at the (raw) data
2. Identify what we need
3. Isolate the numbers needed, and save them
4. Do the necessary arithmetic/selection

see Practical Computing, chapter 10, for an almost identical problem and an outstanding illustration of typical data extraction/reduction strategies (Fig. 10.1)

1) Look at the data

```
sp|GSTM1_HUMAN   sp|GSTM1_HUMAN 100.00 218  0   0  1  218 1 218 7e-127 452
sp|GSTM1_HUMAN   sp|GSTM4_HUMAN  86.70 218 29   0  1  218 1 218 3e-112 403
sp|GSTM1_HUMAN   sp|GSTM1_MACFA  85.78 218 31   0  1  218 1 218 3e-110 397
sp|GSTM1_HUMAN   sp|GSTM2_PONAB  85.78 218 31   0  1  218 1 218 1e-109 395
sp|GSTM1_HUMAN   sp|GSTM2_MACFA  85.78 218 31   0  1  218 1 218 1e-109 395
sp|GSTM1_HUMAN   sp|GSTM5_HUMAN  87.61 218 27   0  1  218 1 218 1e-109 395
```

```
blastp –help
*** Formatting options
-outfmt <String>
  alignment view options:
    0 = pairwise,

    5 = XML Blast output,
    6 = tabular,

  When not provided, the default value is:
  'qseqid sseqid pident length mismatch gapopen qstart qend sstart send
  evalue bitscore', which is equivalent to the keyword 'std'
  Default = `0'
```


2) Identify/extract the data we need

qseqid	sseqid	pident	len	mis	gp	qs	qe	ss	se	evalue	bits
sp|GSTM1_HUMAN	sp|GSTM1_HUMAN	100.00	218	0	0	1	218	1	218	7e-127	452
sp|GSTM1_HUMAN	sp|GSTM4_HUMAN	86.70	218	29	0	1	218	1	218	3e-112	403
sp|GSTM1_HUMAN	sp|GSTM1_MACFA	85.78	218	31	0	1	218	1	218	3e-110	397
sp|GSTM1_HUMAN	sp|GSTM2_PONAB	85.78	218	31	0	1	218	1	218	1e-109	395
sp|GSTM1_HUMAN	sp|GSTM2_MACFA	85.78	218	31	0	1	218	1	218	1e-109	395
sp|GSTM1_HUMAN	sp|GSTM5_HUMAN	87.61	218	27	0	1	218	1	218	1e-109	395

1. Subject accession (sseqid)
2. evalue
3. Select hits with 0.1 < evalue < 2.0

2) Identify/extract the data we need

qseqid	sseqid	pident	len	mis	gp	qs	qe	ss	se	evalue	bits
sp|GSTM1_HUMAN	sp|GSTM1_HUMAN	100.00	218	0	0	1	218	1	218	7e-127	452
sp|GSTM1_HUMAN	sp|GSTM4_HUMAN	86.70	218	29	0	1	218	1	218	3e-112	403
sp|GSTM1_HUMAN	sp|GSTM1_MACFA	85.78	218	31	0	1	218	1	218	3e-110	397
sp|GSTM1_HUMAN	sp|GSTM2_PONAB	85.78	218	31	0	1	218	1	218	1e-109	395
sp|GSTM1_HUMAN	sp|GSTM2_MACFA	85.78	218	31	0	1	218	1	218	1e-109	395
sp|GSTM1_HUMAN	sp|GSTM5_HUMAN	87.61	218	27	0	1	218	1	218	1e-109	395

Get the data:

```
import fileinput
for line in fileinput.input():
  line = line.strip('\n')   # ALWAYS remove \n
  fields = line.split('\t') # break fields at tabs
  # fields[] is an array of strings
  # numbers must be converted to do arithmetic
...
```

Practical Computing, Ch. 9, p 183

3) Isolate the numbers, and save them

#fields[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
qseqid	sseqid	pident	len	mis	gp	qs	qe	ss	se	evalue	bits
sp|GSTM1_HUMAN	sp|GSTM1_HUMAN	100.00	218	0	0	1	218	1	218	7e-127	452
sp|GSTM1_HUMAN	sp|GSTM4_HUMAN	86.70	218	29	0	1	218	1	218	3e-112	403

How do we refer to the data? `fields = line.split('\t')`

1) Array:

```
fields[0], fields[1], fields[3], ...
```

or isolate the ones you need:

```
subj_acc, evalue = fields[1], fields[10]
subj_acc, evalue = fields[1], fields[-2]
```

The problem with arrays is that you need to remember where the data is. Is `fields[10]` the evalue, or the bit score?

2) Dict: (no longer need to remember array indexes, remember names)

```
field_names = ['qseqid', 'sseqid', ..., 'evalue', 'bits']
hit_dict = dict(zip(field_names, fields))
hit_dict = dict(zip(field_names, line.split('\t')))
# zip merges dict "keys" (field names) with "values" (data)
```

4) Do the necessary arithmetic/selection

qseqid	sseqid	pident	len	mis	gp	qs	qe	ss	se	evalue	bits
sp|GSTM1_HUMAN	sp|GSTM1_HUMAN	100.00	218	0	0	1	218	1	218	7e-127	452
sp|GSTM1_HUMAN	sp|GSTM4_HUMAN	86.70	218	29	0	1	218	1	218	3e-112	403

Identify the hits with 0.01 < evalue < 2.0

why is float() needed?

```
if (float(fields[-2]) > 0.1 and float(fields[-2]) <= 2.0) :
  print fields[1]+'\t'+fields[-2]

# alternatively
if (float(hit_dict['evalue']) > 0.1 and
    float(hit_dict['evalue']) <= 2.0) :
  print hit_dict['sseqid']+'\t'+hit_dict['evalue']
```


Summary – Introduction to python

- Variable types:
  - floats, ints, strings
  - arrays=[], tuples=() of floats, ints, strings
  - dict's (hashes) `fields = dict(zip(names, data))`
- Control structures:
  - `for x in list :`
  - `if (true) : ; elif (true): ; else :`
  - statement blocks are indented
- Functions are often applied to variables
  - `array.sort()`
  - `string.strip("\n"), string.split("\t")`
- Some functions are "imported"
  - `import fileinput`
    `for line in fileinput.input()`
  - `from urllib import urlopen`
    `print urlopen(url_string).read()`

Homework, due Monday, 5 Feb (biol4230/hwk3)

1. arrays of random numbers:
   a. Write a program to generate 10 random numbers between 0 and 100 (0 <= x < 100), calculate the mean (average). Print both the random numbers and the mean.
   b. write a program that calculates the average of 100 random "real" numbers between 0 and 200 (0 <= x < 200)
   c. write a program that generates 101 random numbers, stores them in an array, and calculates the median (hint, use the sorted() function to sort the array)
2. write a program that reads a file of Uniprot accession strings and downloads the sequences in FASTA format to "stdout" (use urllib), like hwk1
3. repeat parts of last week's homework using python:
   a. Using one of the blastp (or ssearch36) tabular output files you generated in the similarity searching lab, write a program that extracts the accession and E()-value from the output file. Just as you used "cut" last week, you can use ".split()" this week
   b. modify the program to extract the accessions, but only for results with E() < 0.001 (remember that the tabular output files are ordered by E()-value, so you can stop once you hit 0.001).
   c. write a program that downloads sequences for the blastp or ssearch36 accessions with 0.1 < E() <= 2.0, and runs another blastp search (tabular format) with the downloaded sequences, saving the results of each search in a separate file.
4. As always, document the scripts in hwk3.notes.