Python Programming 2  
Regular Expressions, Arrays, Dictionaries, Debugging

Biol4230  Thurs, Feb 9, 2017  
Bill Pearson  wrp@virginia.edu  4-2818  Jordan 6-057

• String matching and regular expressions:
  ```python
  import re
  if (re.match('>' , fasta_line)):  # match beginning of string
      re_acc_parts = re.compile(r'>(\w+|\w+)')  # extract parts of a match
  if (re_acc_parts.search(ncbi_acc)):
      (db,acc,id) = re_acc_parts.groups()
  file_prefix = re.sub('.aa', '', file_name)  # substitute
  ```
• Working with arrays (lists)
• Dictionaries (dicts[]) and zip()
• python debugging – what is your program doing?
• References and dereferencing – multi-dimensional arrays and dicts

To learn more:
• Practical Computing: Part III – ch. 7 – 10, merging files: ch. 11
• regular expressions:
  – https://docs.python.org/2/howto/regex.html#regex-howto
• Learn Python the Hard Way: learnpythonthehardway.org/book/
• Think Python (collab) www.greenteapress.com/thinkpython/thinkpython.pdf
• Exercises due 5:00 PM Monday, Feb. 13 (save in biol4230/hwk4)
  See: http://fasta.bioch.virginia.edu/biol4230/labs/matrix_hwk4.html
Regular expressions

used for string matching, substitution, pattern extraction

- **import re**
- `r'^>' matches `>sp\|$ | (\w+)`, line) # extract acc with ()
  
  ```python
  gi = re.match.group(1); {
  (acc,id) # match without version number
  = re.match(r'^>sp\|$ | (\w+)\d*| (\w+)',line).groups()
  re.sub(r'\.aa\$','file) # delete ".aa" at end
  re.sub(r'\>(.*)\$','>',line) # substitution
  re.sub(r'^','>',line,1) # same thing (simpler),
  # substitution is global, use ,1 for once
  - '^' – beginning of line; '$' – end of line
  ```

- **plaintext**
  - one<two' # alternation
  - (one|two)|three' # grouping with
  # parenthesis(capture)
- `r'^>' matches `>sp\|$ | (\w+)`, line) # `beginning of line`
  
  ```python
  r'.+ (\d+) aa\$' # $ end of line
  ```
- `a\bc' # bc,abc,aabc, ... # repetitions
  - `a?bc' # abc, bc
  - `a+bc' # abc, aabc, ...`
Regular Expressions, III

>sp|P20432.3|GSTT1_DROME Glutathione S-transferase 1-1

- Matching classes:
  - `r'^>[a-z]+\[[A-Za-z][0-9A-Z]+\.[?d*]']`
    - `[a-z]` [0-9] -> class
    - `[^a-z]` -> negated class
  - `r''>[a-z]+\[\w+.\]'`
    - `\d` -> number [0-9] `\D` -> not a number
    - `\w` -> word [0-9A-Za-z_] `\W` -> not a word char
    - `\s` -> space [ \t\n\r] `\S` -> not a space

- Capturing matches:
  - `r''>({a-z})\[\(\w+\).?\d*\]'`
    - `.group(1)` `.group(2)`
      `(db,db_acc) =`
      `.match(r''>({a-z})\[\(\w+\)\]'`,line).groups()`

Regular expressions – modifiers ignore case requires re.compile()

If your regular expression needs a `'\'` (e.g. `'\'`, `'\d'`, `'\w'`, `'\n'`, be sure to prefix with `'r'` - `r'\d_+\w+\n'`)

```python
import rerayzo = re.compile('That',re.I)  # re.IGNORECASE
if re1.search("this or that"):
    re2=re.compile('#(\w+)\w+\n')  # treat as multiple lines
    re3=re.compile('[\n]',re.S)
    # treat as single long line with internal 'n's
    re3.sub('''',string)  # remove \n in multiline entry
```

fasta.bioch.virginia.edu/biol4230
String expressions  
(with regular expressions)

```python
if re.match(r'^\w{2,3}\\',line):
    while ( not re.match(r'^\w{2,3}\\',line) ) :
        Substitution:
            new_line = re.sub(r'\\',':',old_line)
Pattern extraction:
    (db,db_acc) =
        re.match(r'^([a-z])\((\w+)\\',line).groups()
re.split(r'\s+', line)  # like sseqid.split()
```

Regular expression summary

- regular expressions provide a powerful language for pattern matching
- regular expressions are very very hard to get right
  - when they're wrong, they don't match, and your capture variables are not set
  - always check your capture variables when things don't work
Working with arrays (lists) I –

- Create array:
  ```python
  array=[]
  array_str="cat dog piranha"; array = array_str.split(" ")
  array1=range(1,10)
  [1, 2, 3, 4, 5, 6, 7, 8, 9] # no 10!!!, 9 elements
  array1=range(0,10)
  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] # still no 10, but 10 elements
  array2=range(1,20,2) # second number is max+1
  [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
  ```
- Extract/set individual element:
  ```python
  value=array[1]; value=array[i]
  array[0]=98.6; array[i]=101.4
  ```
- Extract/set list of elements (array slice)
  ```python
  (first, second, third) = array[0:3] # [start:end-1]
  ```
- Python array elements do not have a constant type; array[0] can be a "string" while array[1] is a number.

Working with arrays (lists) II –

- Add to array (array gets longer, at end or start)
  - add one element to end of array
    ```python
    array.append(value) # array[-1]==value
    ```
  - Add elements to end of array
    ```python
    array.extend(list)
    ```
  - add to beginning, less common, less efficient
    ```python
    array.insert(0,value) # array[0] == value
    ```
  - (inserts can go anywhere)
- Remove from array (array gets shorter/smaller)
  ```python
  first_element=array.pop(0)
  last_element=array.pop();
  ```
- Parts of an array (slices, beginning, middle, end)
  ```python
  second_third_array = array[1:3] = array[start:end+1]
  ```
Working with arrays (lists) III–

• Array assignments are *aliases*, NOT copies:
  ```python
  >>> array2
  [1, 'second', 5, 7, 9, 11, 13, 15, 17, 19]
  >>> array2_notcopy = array2
  >>> array2_notcopy.pop()
  19
  >>> array2
  [1, 'second', 5, 7, 9, 11, 13, 15, 17]
  >>> array2_notcopy.pop(0)
  1
  >>> array2_notcopy
  ['second', 5, 7, 9, 11, 13, 15, 17]
  >>> array2
  array2
  ['second', 5, 7, 9, 11, 13, 15, 17]
  ```
• To create a genuine copy, "list comprehensions"
  ```python
  array2_copy = [ x for x in array2 ]
  ```

Working with arrays (lists) IV–

• Two functions: `array.sort()` and `sorted(array)`
  ```python
  num_array = [2.48, 1.72, 2.15, 1.55]
  num_array.sort() # .sort() sorts in place
  [1.55, 1.72, 2.15, 2.48]
  num_array.sort(reverse=True)
  [2.48, 2.15, 1.72, 1.55]
  ```
  ```python
  str_array = ['Bat', 'Aardvark', 'Dog', 'Cat']
  str_array.sort() # or sorted(str_array)
  ['Aardvark', 'Bat', 'Cat', 'Dog']
  ```
• Build new array: list comprehension
  ```python
  new_array = [ x*x for x in num_array ]
  ```
• Build a subset of an array: list comprehension
  ```python
  no_a_animal
  = [ x for x in str_array if not re.search('[aA]',x) ]
  no_a_animal == ['Dog']
  ```
python dictionaries (dicts) –
Arrays with names, not positions

months = ['Jan', 'Feb', 'Mar', 'Apr', ...]  # list
months[0] == 'Jan'; months[3]=='Apr'
month_days = [31, 28, 31, 30, ...]  # month_days[1] == 28

month_day_dict={'Jan':31, 'Feb':28, 'Mar':31, 'Apr':30,...}
# alternatively:
month_day_dict=dict(zip(months, month_days))
month_day_dict['Feb'] == 28; month_day_dict.get('Feb') == 28
month_day_dict['XYZ'] == error; month_day_dict.get('XYZ') == None

data_dict = {}
data_dict[key] = value;
for key in data_dict.keys():
    print key, data_dict[key]  # note keys are not ordered

Practical Computing, Ch 9, pp. 151-158

python dicts (cont.)

• dict keys can be checked with 'in' or '.get()'
  'Meb' in month_day_dict == False
  month_day_dict.get('Meb') == None

• "in" is convenient for checking for duplicates, e.g.
  if ('P09488' in acc_dict): #do something
  else: acc_dict['P09488'] = evalue # now it is defined

• Unlike an array=[], a dict={} is unordered:
  for month in months:  # prints months in order;
    for month in month_dict.keys():
      # could be Dec, Mar, Sep, etc.
  If you need the elements of a dict in order, either keep a separate array
  (months), or make a 2-D dict with an index (see next)
python loves arrays (lists). Most python programs NEVER refer to individual data elements with an index (no array[i]). How to easily isolate the information desired (sseqid; evalue)? How do we refer to the data?

1) Array slice:
   data[0], data[1], data[3], ...
   or isolate the ones you need: (array slice, just pick what you want)
   ```python
   hit_data = [data[0:4] + data[10]]
   hit_data = [data[0:4] + data[-2]]
   ```

Python provides continuous "slices", and has list/dict comprehensions

2) dict:
   ```python
   hit_dict = dict(zip(['qseqid', 'sseqid', ... 'evalue', 'bits'], data))
   field_name_str = 'qseqid sseqid ... evalue bits'
   field_names = field_name_str.split(' ')
   hit_dict = dict(zip(field_names, data))
   hit_dict = dict(zip(field_names, line.split(' \t')))
   print "\t".join([hit_dict[sseqid], str(hit_dict[evalue])])
   ```

fasta.bioch.virginia.edu/biol4230
python debugging

1. Fix syntax errors (undeclared variables, missing ':' or ')')
   python script_name.pl

2. Use 'print'

3. If the program does not work (or prints nonsense), or if you just want to watch it work, add:
   python –mpdb script_name.py # then
   script_name.py # immediately stops for debugging
   – 'n': next (over functions)
   – 's': step (into functions)
   – 'b': break # 'disable # to remove break #
   – 'c': continue
   – 'q': quit
   – 'h': help

4. The debugger is a python interpreter, so you can try anything you like.
   (Pdb) print re.split('s+','this is a short string')
   ['thi', ' i', ' a ', 'hort ', 'tring']

downloading using 'print'

#!/bin/env python

import fileinput
import subprocess

base_url = "http://www.uniprot.org/uniprot"
for line in fileinput.input():
    line = line.strip('\n')
    fields = line.split('\t')
    if (float(fields[-2]) >= 0.1 and float(fields[-2]) < 2.0):
        parts = fields[1].split('\n')
        acc = parts[3]
        curl_cmd = "curl -O "+base_url+acc+".fasta"
        print curl_cmd
        # subprocess.call(curl_cmd, shell=True)
debugging using 'print'

```python
#!/bin/env python
import fileinput
import subprocess
base_url = "http://www.uniprot.org/uniprot/
for line in fileinput.input():
    line = line.strip('\n')
    fields = line.split('\t')
    if (float(fields[-2]) >= 0.1 and float(fields[-2]) < 2.0):
        parts = fields[1].split('|')
        acc = [parts[3].split('.')][0]
        curl_cmd = 'curl -O "+base_url+acc+.fasta"'
        print curl_cmd
        # subprocess.call(curl_cmd, shell=True)
```

```bash
python good_hwk3.py gstm1_swissp.bl_tab
curl -0 http://www.uniprot.org/uniprot/P30713.fasta
curl -0 http://www.uniprot.org/uniprot/P0CG30.fasta
curl -0 http://www.uniprot.org/uniprot/P0CG29.fasta
curl -0 http://www.uniprot.org/uniprot/Q13155.fasta
curl -0 http://www.uniprot.org/uniprot/Q85B60.fasta
curl -0 http://www.uniprot.org/uniprot/Q2NHU0.fasta
```

the python debugger: pdb

```python
#!/bin/env python
import pdb; pdb.set_trace()  # load the debugger, or python -mpdb
month_str = 'Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec'
months = month_str.split(' ')
month_days = [31, 28, 31, 30, 31, 30, 31, 31, 31, 30, 31, 31]
month_dict = {}
for i in range(len(months)):
    month_dict[months[i]] = month_days[i]
for month in months:  # line 14
    print month
for month in months:  # line 17
    print month, month_dict[month]
month_dict2 = dict(zip(months, month_days))
for month in months:
    print month, month_dict2[month]
```

```bash
fasta.bioch.virginia.edu/bio4230
```
franklin: 2 $ python -mpdb dict_intro.py
> /net/t102/users/wrp/biol4230/scripts/dict_intro.py(5)<module>()
  --> month_str = 'Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec'
(Pdb) m # next step
> /net/t102/users/wrp/biol4230/scripts/dict_intro.py(6)<module>()
  --> months = month_str.split(' ')
(Pdb) m # next step
> /net/t102/users/wrp/biol4230/scripts/dict_intro.py(7)<module>()
(Pdb) print months
['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', ..., 'Nov', 'Dec']
(Pdb) m
  --> month_dict = {}
(Pdb) m
  --> for i in range(len(months)):
(Pdb) m
  --> month_dict[months[i]] = month_days[i]
(Pdb) m
  --> for i in range(len(months)):
(Pdb) b 14 # break at line 14, for month in months
Breakpoint 1 at /net/t102/users/wrp/biol4230/scripts/dict_intro.py:14
(Pdb) c # continue to breakpoint
> /net/t102/users/wrp/biol4230/scripts/dict_intro.py(14)<module>()
  --> for month in months:
(Pdb) b 17 # break at line 17, second for month in months
Breakpoint 2 at /net/t102/users/wrp/biol4230/scripts/dict_intro.py:17
(Pdb) c # breakpoint is at for ..., but stops at every loop
Jan
(Pdb) c
> /net/t102/users/wrp/biol4230/scripts/dict_intro.py(14)<module>()
  --> for month in months:
(Pdb) b 17 # set breakpoint at next loop
Breakpoint 2 at /net/t102/users/wrp/biol4230/scripts/dict_intro.py:17
(Pdb) c # breakpoint is at for ..., but stops at every loop
Apr
(Pdb) c
> /net/t102/users/wrp/biol4230/scripts/dict_intro.py(14)<module>()
  --> for month in months:
(Pdb) disable 1 # delete (disable) breakpoint
(Pdb) c
> /net/t102/users/wrp/biol4230/scripts/dict_intro.py(14)<module>()
  --> for month in months:
(Pdb) disable 2 # show breakpoint status
(Pdb) b
Num Type    Disp    Enb    Where
1  breakpoint keep no  at /net/.../biol4230/scripts/dict_intro.py:14
breakpoint already hit 5 times
2  breakpoint keep no  at /net/.../biol4230/scripts/dict_intro.py:17
breakpoint already hit 1 time
(Pdb) quit()

Arrays of arrays (and dicts of dicts)
Python variables are references (already)

- python arrays and dicts are always one-dimensional, but data is usually (at least) two-dimensional.
- How do we build data structures that have multiple dimensions?

```python
hit[1]['percid'] == 86.70
hit[1]['evalue'] == 3e-112
```

Variable dereferencing
To build multi-dimensional (complex) data structures in python, simply put the simple object into the more complex structure (all variables are references in python, no need for reference type):

```python
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"

hit_dict = dict(zip(field_names,line.split('	')))  
hit_list.append(hit_dict)
print hit_list
```
Variable dereferencing

/bin/env python
import fileinput
#import pdb; pdb.set_trace()

field_str = 'qseqid sseqid pident length mismatch ... evalue bitscore'
fields = field_str.split(' ')

hits = []  # list of best hits
for line in fileinput.input():
    line = line.strip('
')
    data_dict = dict(zip(fields, line.split('	')))
    hits.append(data_dict)  # hit[n] = {data}

for hit in hits:
    print hit['sseqid'], hit['evalue']

fasta.bioch.virginia.edu/biol4230

Variable dereferencing

franklin: 20 $ python read_hits.py hit_list.data
> /net/t102/users/wrp/biol4230/scripts/read_hits.py(6)<module>()
-> field_str = 'qseqid sseqid pident length mismatch ... evalue bitscore'
(Pdb) a
-> for line in fileinput.input():
(Pdb) print hits[0]
*** IndexError: list index out of range  # have not appended anything, list empty

(Pdb) print hits[0]  # hits[0] == (data)
(Pdb) a
({}, 'bitscore': '452', 'evalue': '7e-127', ...

(Pdb) print hits[0]['sseqid']
sp|GSTM1_HUMAN
(Pdb) print hits[1]['sseqid'], hits[1]['evalue']
sp|GSTM1_HUMAN 7e-127
...
(Pdb) print hits[2]['sseqid'], hits[2]['evalue']
sp|GSTM1_MACPA 3e-110

fasta.bioch.virginia.edu/biol4230
keeping order with dicts[]

When keeping track of a list of hits (or a list of scoring matrices), one often needs two variables

1. a list of the data sets (matrix1, matrix2, matrix3)
2. a list of the results, indexed (keyed) on the dataset names

In the homework, you are asked to report summaries of alignment length and percent identity for multiple searches with multiple scoring matrices. You will need to keep track of the matrix specific data, and the query specific data.

One way to do this is with a list of matrices:

```python
mat_list=['mat1', 'mat2', 'mat3', etc.]
```
as well as

```python
result_dict={mat1:array_of_hits, mat2:array_of_hits, etc.}
```

for the homework, you will need to read a set of files (with the matrix name part of the file name), extract the matrix name, add it to the list of matrix names, and then add the hits to a dict[] that uses the matrix name as the key.

To simplify the process of keeping track of your search queries, search results, and matrix names by using a consistent naming scheme. For example, have q200_0.aa, q200_2.aa, ... q200_9.aa, and results q200_0.bl_blosum62, ... q200_9.bl_blosum62, q200_0.bl_blosum45, etc.

Homework, due Monday, 13 Feb (biol4230/hwk4)

Follow the instructions at:

fasta.bioch.virginia.edu/biol4230/labs/matrix_hwk4.html