Taejoon Kwon
UNIST, Republic of Korea
Xenopus Bioinformatics Workshop, May 2016
Can you do a programming? (19 responses)

- Yes: 31.6%
- No: 68.4%

Which topics are you most interested in? (multiple choice) (19 responses)

- Transcriptomics: 19 (100%)
- Epigenome: 10 (52.6%)
- Proteome (M...): 6 (31.6%)
- Genome: 11 (57.9%)
- Other: 1 (5.3%)
Which frog (or species) are you mostly working with? (please describe your model system if you are not working with Xenopus)

(19 responses)

- Xenopus lae... 17 (89.5%)
- Xenopus tro... 5 (26.3%)
- Other 2 (10.5%)

Which operating system is running on your laptop? (you need to bring it to the workshop)

(19 responses)

- MS Windows 10 26.3%
- Mac OSX 10.11 (El Capitan) 21.1%
- Previous version of MS Windows (8, 7, etc) 21.1%
- Previous version of Mac OSX 21.1%
- Linux 10.5%
Programmers

21st century magicians

http://37.media.tumblr.com/tumblr_ly0dpda7ns1r7t60no1_500.jpg
I AM A PROGRAMMER
NOT A MAGICIAN
I AM A BIOCHEMIST (OR BIOLOGIST)
NOT A MAGICIAN
Too much recipes!?!?
My personal journey in programming

• ~ 1994: C (and little bit of BASIC)
• 1994 ~ 1995: Fortran(!)
• 1996 ~ 1998: Visual C++ (and little bit of Java)
• 1998 ~ 2008: PERL
  – 2002 ~ 2005: Java (in the company)
• 2008 ~ current: Python
  – & little bit of R, PHP5, JavaScript, Java, C# & Ruby
Programming language – my opinion

• C/C++ : Most powerful. Period.
  – Need to know a lot about computer itself (i.e. memory allocation).
  – Steep learning curve (even you know another language).

• Java & C# : Powerful & comprehensive.
  – Need to understand ‘object-orient programming’.
  – Ideal for ‘huge project’, but too ‘heavy’ to use in small tasks.

• JavaScript & PHP: A language for the web. Limited.

• Unix shell scripting (BASH, TCSH): A language for the command line. Powerful But Limited.
Programming language – my opinion

• PERL: Powerful in text manipulation
  – Check out Lincoln Stein’s article “How PERL save the human genome” in the wiki.
  – Still widely used in bioinformatics (i.e. EnsEMBL, GBrowse)
• Ruby: “New Kids On The Block”
  – Hybrid of PERL (flexibility) and Python (object oriented structure); but little bit premature yet.
• MATLAB: Powerful in machine learning & statistics.
  – Expensive (many institutes may have a site license, though)
• R: Powerful in statistics.
  – Little bit ‘strange’ syntax; steep learning curve
• If you can do it with python,
  – You can do it with PERL
  – You can do it with Ruby
  – You can do it with MATLAB or R
  – You can do it with C or C++
  – You can do it with Java or C#
  – (but may not with BASH, JavaScript, PHP)

• Just pick any of them, learn it, and use it everyday. Soon you will become a programmer (or a magician).

• Don’t be stressed to google it when you have a question. All programmers also do it.
  – It is same to check a protocol before the experiment. I don’t believe any biologist can memorize all parts of “Molecular Cloning” or “Xenopus handbook”.
Why Python?

• Compared to C/C++/Java/C#
  – Easier to learn.
  – More suitable for ‘simple tasks’ that we are interested in.

• Compared to Ruby
  – More mature (personal opinion).

• Compared to PERL
  – Easier to organize codes (more object-oriented).
  – All-in-one package (free from module dependency).
  – Bioinformatics community with python is getting bigger.
  – Useful libraries: numpy/scipy & matplotlib
  – Personally I don’t want to go back to PERL.

• Python3 has some good features, but many libraries do not support it yet. We will use python-2 instead here.
Two ways to work with python

• Traditional way
  – Write a code with your favorite text editor.
  – Run the code with ‘> python <my_code.py>’ command.

• Interactive way
  – Execute ‘> python’ in your command terminal.
  – Do the programming inside ‘interactive’ shell.
  – Check out ipython notebook function at http://ipython.org/notebook.html

• Find more comfortable way for yourself.
Ok, let’s get to work!

http://www.maniacworld.com/get-to-work.jpg
Installation of Anaconda Distribution
(http://continuum.io/downloads)

Download Anaconda

Anaconda is a completely free Python distribution (including for commercial use and redistribution). It includes over 195 of the most popular Python packages for science, math, engineering, data analysis.

CHOOSE YOUR INSTALLER:

- Windows 64-Bit Python 2.7 Graphical Installer
  Size: 334M
- Other Installers:
  - Windows 32-bit — Python 2.7 — Graphical Installer
    Size: 277M
  - Zipped Windows Installers
    zipped Windows executable files
    for those behind firewalls

INSTALLATION

After downloading the installer, double click the .exe file and follow the instructions on the screen.

For more information on installation, please read the documentation.

You can find MD5 information for Anaconda installers here.

*Anaconda comes with installers for Python 2.7 and 3.4. You can use Python 2.6 and 3.3 by selecting the specific version.
Run ‘ipython notebook’
@ the command windows
In [1]: `print "Hello World"
Hello World

In [ ]:
• Make your code beautiful.
  – It is like to “make your bench clean”. Nobody wants to do the experiment in dirty bench, even it is actually YOU who to make all those messes.

• If you don’t have any preference, just follow well-established coding style. There is reasons for this style, and you may know them in the future.
  – http://legacy.python.org/dev/peps/pep-0008/
  – http://google-styleguide.googlecode.com/svn/trunk/pyguide.html

• Spend a time before naming anything (variables, functions, filenames)
  – test1.py, test2.py, test3.py, ...
  – taejoon1.py, taejoon2.py, ....
Indentation matters in python
(be aware if you have experienced in other languages)

C code

```c
int main(int argc, char **argv) {
    /* Data */
    int p, n, niter;
    DATA data; /* all data */
    PAR par;
    char newName[256];

    const gsl_rng_type *T;
    gsl_rng *r;
    gsl_rng_env_setup();
    T = gsl_rng_default;
    r = gsl_rng_alloc(T);

    if (!((argc == 2 || argc == 4)) {
        fprintf(stderr, "usage: mbl[ data ] [ncompPos] [ncomp]
        fprintf(stderr, "usage: mbl[ data ] (default: ncomp"
        return 1;
    }
    FILE *fp = fopen(argv[1], "r");
    if (fp == NULL) {
        fprintf(stderr, "Data file %s does not exist.\n", argv[1])
        return 1;
    }
}
```

Python code

```python
#!/usr/bin/env python
import os
import sys

if (len(sys.argv) < 3):
    sys.stderr.write("usage: python msblender_out-to-pep_count-mFDR
    sys.exit(1)

filename_mb_out = sys.argv[1]
FDR_cutoff = float(sys.argv[2])
FDR_string = sys.argv[2].replace(';', ',')

error_model = 'mFDRpsm'
error_model_list = ['eFDRpsm', 'mFDRpsm']
if (len(sys.argv) == 4 and sys.argv[3] in error_model_list):
    error_model = sys.argv[3]

sys.stderr.write("FDR cutoff: %3f\nError model:%s\n"%(FDR_cutoff,e
filename_base = filename_mb_out.replace('.msblender_in', '').replace
psm_mvScore = dict()
psm_TD = dict()

f_mb_out = open(filename_mb_out, 'r')
f_mb_out.readline()  # tokens = line.strip().split("\t")
tmp_mvScore = float(tokens[-1])
tmp_psm = tokens[0]
psm_mvScore[tmp_psm] = float(tokens[-1])
psm_TD[tmp_psm] = tokens[1]
f_mb_out.close()
```
Major components in programming

• Variables: “How to store data?”
  – Scalar: number, string
  – Array/List
  – Dictionary/Hash

• Control flows: “How to process data to get a result?”
  – Conditions (if ... then ... else ...)
  – Loop (for ... while ...)

• Operations & functions

• Input/Output: “How to read data/write result?”
80 built-in functions

<table>
<thead>
<tr>
<th>abs()</th>
<th>divmod()</th>
<th>input()</th>
<th>open()</th>
<th>staticmethod()</th>
</tr>
</thead>
<tbody>
<tr>
<td>all()</td>
<td>enumerate()</td>
<td>int()</td>
<td>ord()</td>
<td>str()</td>
</tr>
<tr>
<td>any()</td>
<td>eval()</td>
<td>isinstance()</td>
<td>pow()</td>
<td>sum()</td>
</tr>
<tr>
<td>basestring()</td>
<td>execfile()</td>
<td>issubclass()</td>
<td>print()</td>
<td>super()</td>
</tr>
<tr>
<td>bin()</td>
<td>file()</td>
<td>iter()</td>
<td>property()</td>
<td>tuple()</td>
</tr>
<tr>
<td>bool()</td>
<td>filter()</td>
<td>len()</td>
<td>range()</td>
<td>type()</td>
</tr>
<tr>
<td>bytearray()</td>
<td>float()</td>
<td>list()</td>
<td>raw_input()</td>
<td>unichr()</td>
</tr>
<tr>
<td>callable()</td>
<td>format()</td>
<td>locals()</td>
<td>reduce()</td>
<td>unicode()</td>
</tr>
<tr>
<td>chr()</td>
<td>frozenset()</td>
<td>long()</td>
<td>reload()</td>
<td>vars()</td>
</tr>
<tr>
<td>classmethod()</td>
<td>getattr()</td>
<td>map()</td>
<td>repr()</td>
<td>xrange()</td>
</tr>
<tr>
<td>cmp()</td>
<td>globals()</td>
<td>max()</td>
<td>reversed()</td>
<td>zip()</td>
</tr>
<tr>
<td>compile()</td>
<td>hasattr()</td>
<td>memoryview()</td>
<td>round()</td>
<td><strong>import</strong>()</td>
</tr>
<tr>
<td>complex()</td>
<td>hash()</td>
<td>min()</td>
<td>set()</td>
<td>apply()</td>
</tr>
<tr>
<td>delattr()</td>
<td>help()</td>
<td>next()</td>
<td>setattr()</td>
<td>buffer()</td>
</tr>
<tr>
<td>dict()</td>
<td>hex()</td>
<td>object()</td>
<td>slice()</td>
<td>coerce()</td>
</tr>
<tr>
<td>dir()</td>
<td>id()</td>
<td>oct()</td>
<td>sorted()</td>
<td>intern()</td>
</tr>
</tbody>
</table>

https://docs.python.org/2/library/functions.html
Numbers: integer & float

| In [5]: | 2+2 |
| Out[5]: | 4 |

| In [6]: | print 2+2 |
|        | 4 |

| In [7]: | 7/2 |
| Out[7]: | 3 |

| In [8]: | 7.0/2 |
| Out[8]: | 3.5 |

| In [9]: | 7/2.0 |
| Out[9]: | 3.5 |

| In [10]: | 7/-2 |
| Out[10]: | -4 |

| In [11]: | 7/-2.0 |
| Out[11]: | -3.5 |

| In [12]: | width = 20 |
| In [13]: | height = 5 |

| In [14]: | width*height |
| Out[14]: | 100 |

| In [15]: | area = width*height |
| In [16]: | print area |
|        | 100 |

| In [19]: | 5%2 |
| Out[19]: | 1 |

| In [20]: | 6%2 |
| Out[20]: | 0 |

| In [21]: | 3+2*5 |
| Out[21]: | 13 |

| In [22]: | (3+2)*5 |
| Out[22]: | 25 |

| In [23]: | 10%7 |
| Out[23]: | 3 |

| In [24]: | 28%7 |
| Out[24]: | 0 |

| In [25]: | float(7)/2 |
| Out[25]: | 3.5 |

| In [26]: | int(7.0)/2 |
| Out[26]: | 3 |

| In [27]: | a = 2 |
| In [28]: | b = 3.1415 |

| In [29]: | print "%d,%02d\t%05d"%(a,a,a) |
|        | 2,02 00002 |

| In [30]: | print "%2f %.5f %.2e"%(b,b,b) |
|        | 3.14 3.14150 3.14e+00 |

https://docs.python.org/2/tutorial/introduction.html
**List/Array**

In [71]: a = ['spam', 'eggs', 100, 1234]

In [72]: a[0]
Out[72]: 'spam'

In [73]: a[3]
Out[73]: 1234

In [74]: a[-1]
Out[74]: 1234

In [75]: a[0:2]
Out[75]: ['spam', 'eggs']

In [76]: a[0:2]+['bacon', 2*2]
Out[76]: ['spam', 'eggs', 'bacon', 4]

In [77]: a[:len(a)-1]
Out[77]: [1234, 100, 'eggs', 'spam']

In [78]: a[2]+100
Out[78]: 200

In [79]: len(a)
Out[79]: 4

In [80]: a = []

In [81]: a
Out[81]: []

In [140]: a = ['banana', 'kiwi', 'pear', 'apple']

In [141]: sorted(a)
Out[141]: ['apple', 'banana', 'kiwi', 'pear']

In [82]: a = [10, 20] + ['chicken', 'egg']

In [83]: a
Out[83]: [10, 20, 'chicken', 'egg']

In [84]: a.append('frog')

In [85]: a
Out[85]: [10, 20, 'chicken', 'egg', 'frog']

In [86]: a.pop()
Out[86]: 'frog'

In [87]: a
Out[87]: [10, 20, 'chicken', 'egg']

In [88]: a.append([['hello', 'MBL']])

In [89]: a
Out[89]: [10, 20, 'chicken', 'egg', [['hello', 'MBL']]]

In [90]: len(a)
Out[90]: 5

In [91]: a[3]
Out[91]: 'egg'

In [92]: a[4]
Out[92]: [['hello', 'MBL']]

In [93]: a[4][1]
Out[93]: 'MBL'

In [144]: b = [23, 13, 53, 2]

In [145]: sorted(b)
Out[145]: [2, 13, 23, 53]
In [36]: a = 'Hello'
In [37]: print a
Hello
In [38]: b = '\tXenopus\tRocks!\t'
In [39]: print b
Xenopus Rocks! Hello
In [40]: print b.strip(), a
Xenopus Rocks! Hello
In [41]: print a.upper()
HELLO
In [42]: print a.lower()
hello
In [43]: print a.upper()+a.lower()
HELLOhello
In [44]: print '3'+ '5'
35
In [45]: print int('3') + int('5')
8
In [46]: print "%s"%(b)
Xenopus Rocks!
In [47]: print "xxx%syyy"%(b)
xxx Xenopus Rocks! yyy
In [56]: x = 'Xenopus Rocks!'
In [57]: print x
Xenopus Rocks!
In [58]: print x[2]

In [59]: print x[:2]
Xe
In [60]: print x[:6:-1]
!skcoR
In [61]: print x[::-1]
!skcoR suponeX
In [62]: len(x)
Out[62]: 14
In [63]: y = x.split(' ')
In [64]: print y
['Xenopus', 'Rocks!']
In [65]: print y[0]
Xenopus
In [66]: z='a\tb\tc\td\te'
In [67]: print z
a b c d e
In [68]: z_list = z.split("\t")
In [69]: print z_list
['a', 'b', 'c', 'd', 'e']
In [70]: print z_list[2]
c
Dictionary (a.k.a Hash) & Set

In [94]: tel = {'jack':4098, 'sape':4139}
In [95]: tel['quido'] = 4127
In [96]: tel
Out[96]: {'jack': 4098, 'quido': 4127, 'sape': 4139}
In [97]: print tel['jack']
4098
In [98]: del tel['sape']
In [99]: tel
Out[99]: {'jack': 4098, 'quido': 4127}
In [100]: len(tel)
Out[100]: 2
In [102]: another_tel = dict(sape=4139, guido=4127, jack=4098)
In [103]: another_tel
Out[103]: {'guido': 4127, 'jack': 4098, 'sape': 4139}
In [104]: tel = {}
In [105]: tel
Out[105]: {}
In [106]: tel['guido'] = 4127
In [107]: tel
Out[107]: {'guido': 4127}
In [108]: tel = dict()

In [112]: a = set('abracadabra')
In [113]: b = set('alacazam')
In [114]: a
Out[114]: {'a', 'b', 'c', 'd', 'r'}
In [115]: b
Out[115]: {'a', 'c', 'l', 'm', 'z'}
In [116]: a - b
Out[116]: {'b', 'd', 'r'}
In [117]: a | b
Out[117]: {'a', 'b', 'c', 'd', 'l', 'm', 'r', 'z'}
In [118]: a.union(b)
Out[118]: {'a', 'b', 'c', 'd', 'l', 'm', 'r', 'z'}
In [119]: a & b
Out[119]: {'a', 'c'}
In [120]: a.intersection(b)
Out[120]: {'a', 'c'}

In [123]: a|b - a&b
Out[123]: {'a', 'b', 'c', 'd', 'l', 'm', 'r', 'z'}
In [124]: (a|b) - (a&b)
Out[124]: {'b', 'd', 'l', 'm', 'r', 'z'}
In [125]: a^b
Out[125]: {'b', 'd', 'l', 'm', 'r', 'z'}
In [126]: (a|b) - (a^b)
Out[126]: {'b', 'd', 'l', 'm', 'r', 'z'}
In [127]: c = list(set(a&b))
In [128]: c
Out[128]: ['a', 'c']
In [129]: print c[0]
a
Advanced: modules

```python
In [130]: def add(tmp_a, tmp_b):
    ...:     return tmp_a+tmp_b
    ...

In [131]: add(20,50)
Out[131]: 70

In [132]: def raw_and_add(tmp_a, tmp_b):
    ...:     print tmp_a
    ...:     print tmp_b
    ...:     return tmp_a,tmp_b,tmp_a+tmp_b
    ...

In [133]: add(20,50)
Out[133]: 70

In [134]: raw_and_add(20,50)
20
50
Out[134]: (20, 50, 70)

In [135]: in_a, in_b, in_a_and_b = raw_and_add(20,50)
20
50

In [136]: in_a
Out[136]: 20

In [137]: in_b
Out[137]: 50

In [138]: in_a_and_b
Out[138]: 70
```
Flow control: if... elif ... else ...
(comparison: ==, !=, >, <, >=, <=)

In [146]: def rock_sissors_paper(tmp):
    ...:     if tmp == 'rock':
    ...:         return 'paper'
    ...:     elif tmp == 'sissor':
    ...:         return 'rock'
    ...:     elif tmp == 'paper':
    ...:         return 'sissor'
    ...:     else:
    ...:         return "I don't understand."
    ...

In [147]: rock_sissors_paper('rock')
Out[147]: 'paper'

In [148]: rock_sissors_paper('sissor')
Out[148]: 'rock'

In [149]: rock_sissors_paper('paper')
Out[149]: 'sissor'

In [150]: rock_sissors_paper('xenopus')
Out[150]: "I don't understand."

In [151]: def starts_with_X(tmp):
    ...:     if tmp.startswith('X'):
    ...:         return True
    ...:     else:
    ...:         return False
    ...

In [152]: starts_with_X('Xenopus')
Out[152]: True

In [153]: starts_with_X('Zebrafish')
Out[153]: False

In [154]: if starts_with_X('Xenopus'):
    ...:     print "Awesome!"
    ...:
Awesome!

In [155]: if starts_with_X('Zebrafish'):
    ...:     print "Awesome!"
    ...:     else:
    ...:         print "Boo~"
    ...:
Boo~
Flow control: for

In [164]: for animal in ['cat', 'frog', 'dog', 'lion']:
    ...:     if len(animal) > 3:
    ...:         print animal, "is cool"
    ...:     else:
    ...:         print animal, "is not cool"

  cat is not cool
  frog is cool
  dog is not cool
  lion is cool

In [165]: sum = 0

In [166]: for i in [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]:
    ...:     sum = sum + i
    ...:     print sum

  1
  3
  6
  10

In [169]: for i in range(1, 11):
    ...:     if i % 2 == 0:
    ...:         print "Even ", i
    ...:     continue
    ...:     if i == 7:
    ...:         break
    ...:     print i

  1
  Even 2
  3
  4
  5
  Even 6

In [171]: for n in range(2, 10):
    ...:     for x in range(2, n):
    ...:         if n % x == 0:
    ...:             print n, 'equals', x, '*', n/x
    ...:             break
    ...:     else:
    ...:         print n, 'is a prime number'

  2 is a prime number
  3 is a prime number
  4 equals 2 * 2
  5 is a prime number
  6 equals 2 * 3
  7 is a prime number
  8 equals 2 * 4
  9 equals 3 * 3
  10 is a prime number
Input & Output

• First, you need to open a file by “open()”
  – open(<filename>,’r’) for reading.

• Then, read contents by “read()”
  – Or use the iterator (see next slide)

• Then, close the file with “close()”

• First, you need to open a file by “open()”
  – open(<filename>,’w’) for writing.

• Then, write stuff by “write()”
  – f.write( “%d
”%(my_integer) )
  – sys.stdout.write() \rightarrow same as print()
  – sys.stderr.write()

• Then, close the file with “close()”
zip & gzip – a library for compressed file

In [46]:

```python
import zipfile
with zipfile.ZipFile("ERCC92.zip","r") as zf:
    for filename in zf.namelist():
        print "Filename: ", filename
        f = zf.open(filename,'r')
        print "First line: ", f.readline()
        f.close()
```

Filename: ERCC92.fa
First line: >ERCC-00002

In [1]:

```python
import gzip
gzf = gzip.open('ERCC92.fa.gz','rb')
print gzf.readline()
gzf.close()
```

>ERCC-00002
Codes I have used almost everyday

```
#!/usr/bin/env python
import os
import sys

filename_fa = sys.argv[1]

seqlen = dict()
seq_h = ""
f_fa = open(filename_fa,'r')
for line in f_fa:
    if( line.startswith('>') ):
        seq_h = line.strip().lstrip('>
        seq_len[seq_h] = 0
    else:
        seq_len[seq_h] += len(line.strip())
f_fa.close()
```

```
#!/usr/bin/env python
import os
import sys

filename_tsv = sys.argv[1]

f_tsv = open(filename_tsv,'r')
f_out = open('results.txt','w')
for line in f_tsv:
    if( line.startswith('#') ):
        continue
    tokens = line.strip().split("t")
    if( tokens[0].upper().find('BMP4') > 0 ):
        f_out.write('%%s\t%%s\n'(tokens[0],tokens[2]))
f_tsv.close()
f_out.close()
```
REALLY advanced: regular expression

- The way to perform ‘pattern matching’ with strings.
- If built-in function of string is not enough for your job...
  - split(), replace(), strip(), startswith(), endswith(), ...
- Google’s python course is good place to start (see below URL).
- Don’t cry if you don’t understand what they are talking about; it is not easy to get it at first sight.

```python
## Search for pattern 'iii' in string 'piiiig'.
## All of the pattern must match, but it may appear anywhere.
## On success, match.group() is matched text.
match = re.search(r'iii', 'piiiig') => found, match.group() == "iii"
match = re.search(r'igs', 'piiiig') => not found, match == None

## . = any char but \n
match = re.search(r'.g', 'piiiig') => found, match.group() == "iig"

## \d = digit char, \w = word char
match = re.search(r'\d\d\d', 'p123g') => found, match.group() == "123"
match = re.search(r'\w\w\w', '@abod!!') => found, match.group() == "abc"
```

https://developers.google.com/edu/python/regular-expressions
scipy/numpy

(http://www.numpy.org & http://www.scipy.org)

NumPy

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

NumPy is licensed under the BSD license, enabling reuse with few restrictions.

Getting Started

- Getting NumPy
- Installing the SciPy Stack
- NumPy and SciPy documentation page
- NumPy Tutorial
- NumPy for MATLAB® Users
- NumPy functions by category
- NumPy Mailing List

More Information

- NumPy SourceForge Home Page
- SciPy Home Page
- Interfacing with compiled code
- Older python array packages

SciPy (pronounced “Sigh Pie”) is a Python-based ecosystem of open-source software for mathematics, science, and engineering. In particular, these are some of the core packages:

- NumPy Base N-dimensional array package
- SciPy library Fundamental library for scientific computing
- Matplotlib Comprehensive 2D Plotting
- IPython Enhanced interactive console
- SymPy Symbolic mathematics
- pandas Data structures & analysis

More information...
Statistics

Order statistics

- `amin(a[, axis, out, keepdims])`: Return the minimum of an array or minimum along an axis.
- `amax(a[, axis, out, keepdims])`: Return the maximum of an array or maximum along an axis.
- `nanmin(a[, axis, out, keepdims])`: Return minimum of an array or minimum along an axis ignoring any NaNs.
- `nanmax(a[, axis, out, keepdims])`: Return maximum of an array or maximum along an axis ignoring any NaNs.
- `ptp(a[, axis, out])`: Range of values (maximum - minimum) along an axis.
- `percentile(a, q[, axis, out, ...)`: Compute the qth percentile of the data along the specified axis.

Averages and variances

- `median(a[, axis, out, overwrite_input, keepdims])`: Compute the median along the specified axis.
- `average(a[, axis, weights, returned])`: Compute the weighted average along the specified axis.
- `mean(a[, axis, dtype, out, keepdims])`: Compute the arithmetic mean along the specified axis.
- `std(a[, axis, dtype, out, ddof, keepdims])`: Compute the standard deviation along the specified axis.
- `var(a[, axis, dtype, out, ddof, keepdims])`: Compute the variance along the specified axis.
- `nanmean(a[, axis, dtype, out, keepdims])`: Compute the arithmetic mean along the specified axis, while ignoring NaNs.
- `nanstd(a[, axis, dtype, out, ddof, keepdims])`: Compute the standard deviation along the specified axis, while ignoring NaNs.
- `nanvar(a[, axis, dtype, out, ddof, keepdims])`: Compute the variance along the specified axis, while ignoring NaNs.

Correlating

- `corrcoef(x[, y, rowvar, bias, ddof])`: Return correlation coefficients.
- `correlate(a, v[, mode, old_behavior])`: Cross-correlation of two 1-dimensional sequences.

Code Example:

```python
%matplotlib inline
seq_len = dict()
seq_h = ''
f = open('ERCC92.fa', 'r')
for line in f:
    if line.startswith('> '):
        seq_h = line.strip().lstrip('>')
        seq_len[seq_h] = 0
    else:
        seq_len[seq_h] += len(line.strip())
f.close()

import scipy
print sum(seq_len.values()) * 1.0 / len(seq_len)
print scipy.mean(seq_len.values())
```

899.52173913
899.52173913
Introduction

matplotlib is a python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. matplotlib can be used in python scripts, the python and ipython shell (ala MATLAB® or Mathematica®), web application servers, and six graphical user interface toolkits.

matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, errorcharts, scatterplots, etc., with just a few lines of code. For a sampling, see the screenshots, thumbnail gallery, and examples directory.

For simple plotting the pyplot interface provides a MATLAB-like interface, particularly when combined with IPython. For the power user, you have full control of line styles, font properties, axes properties, etc, via an object oriented interface or via a set of functions familiar to MATLAB users.
```python
#matplotlib inline

seq_len = dict()
seq_h = ''
f = open('ERCC92.fa','r')
for line in f:
    if line.startswith('> '):
        seq_h = line.strip().lstrip('>
        seq_len[seq_h] = 0
    else:
        seq_len[seq_h] += len(line.strip())
f.close()

import matplotlib.pyplot as plt
fig = plt.figure(figsize=(4,4))
ax1 = fig.add_subplot(1,1,1)
ax1.hist(seq_len.values(), bins=10)
ax1.grid()
ax1.set_title("Length distribution of ERCC92")
ax1.set_xlabel("Length of sequence")
ax1.set_ylabel("Occurrence")
plt.show()
```
MATPLOTLIB Beginner’s Guide
(http://matplotlib.org/users/beginner.html)

Beginner’s Guide

Release: 1.4.3
Date: July 14, 2015

- Pyplot tutorial
  - Controlling line properties
  - Working with multiple figures and axes
  - Working with text
- Customizing plots with style sheets
  - Defining your own style
  - Composing styles
  - Temporary styling
- Interactive navigation
  - Navigation Keyboard Shortcuts
- Working with text
  - Text introduction
  - Basic text commands
  - Text properties and layout
  - Writing mathematical expressions
  - Typesetting With XeLaTeX/LuaLaTeX
  - Text rendering With LaTeX
  - Annotating text

- Screenshots
  - Simple Plot
  - Subplot demo
  - Histograms
  - Path demo
  - mplot3d
  - Streamplot
  - Ellipses
  - Bar charts
  - Pie charts
  - Table demo
  - Scatter demo
  - Slider demo
  - Fill demo
  - Date demo
  - Financial charts
  - Basemap demo
  - Log plots
  - Polar plots
  - Legends
  - Mathtext_examples
  - Native TeX rendering
  - EEG demo
  - XKCD-style sketch plots
Bar charts

Bar charts are simple to create using the `bar()` command, which includes customizations such as error bars:

(Source code, png, hires.png, pdf)

Scores by group and gender

It's also simple to create stacked bars (`bar_stacked.py`), candlestick bars (`finance_demo.py`), and horizontal bar charts (`barh_demo.py`).
More on Gallery
(http://matplotlib.org/gallery.html)
# XenBioinfo2016

by Taejoon Kwon at University of Texas at Austin

## Practice problems for Xenopus Bioinformatics Workshop 2016

<table>
<thead>
<tr>
<th>Num</th>
<th>Title</th>
<th>Solved By</th>
<th>Cost</th>
<th>Due Date</th>
<th>Questions</th>
<th>Solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Installing Python</td>
<td>0</td>
<td>1</td>
<td>May 3, 2016</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Variables and Some Arithmetic</td>
<td>0</td>
<td>1</td>
<td>May 3, 2016</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Strings and Lists</td>
<td>0</td>
<td>1</td>
<td>May 3, 2016</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Conditions and Loops</td>
<td>0</td>
<td>1</td>
<td>May 3, 2016</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Working with Files</td>
<td>0</td>
<td>1</td>
<td>May 3, 2016</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Dictionaries</td>
<td>0</td>
<td>1</td>
<td>May 3, 2016</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Mendel's First Law</td>
<td>0</td>
<td>1</td>
<td>May 3, 2016</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Counting DNA Nucleotides</td>
<td>0</td>
<td>1</td>
<td>May 3, 2016</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Inferring Protein from Spectrum</td>
<td>0</td>
<td>1</td>
<td>May 4, 2016</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Inferring Peptide from Full Spectrum</td>
<td>0</td>
<td>1</td>
<td>May 4, 2016</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Convert a Peptide into a Peptide Vector</td>
<td>0</td>
<td>1</td>
<td>May 4, 2016</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Convert a Peptide Vector into a Peptide</td>
<td>0</td>
<td>1</td>
<td>May 4, 2016</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Sequence a Peptide</td>
<td>0</td>
<td>1</td>
<td>May 4, 2016</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Find a Highest-Scoring Peptide in a Proteome against a Spectrum</td>
<td>0</td>
<td>1</td>
<td>May 4, 2016</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Implement PSMSearch</td>
<td>0</td>
<td>1</td>
<td>May 4, 2016</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>Compute the Size of a Spectral Dictionary</td>
<td>0</td>
<td>1</td>
<td>May 4, 2016</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>Compute the Probability of a Spectral Dictionary</td>
<td>0</td>
<td>1</td>
<td>May 4, 2016</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>Find a Highest-Scoring Modified Peptide against a Spectrum</td>
<td>0</td>
<td>1</td>
<td>May 4, 2016</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>Using the Spectrum Graph to Infer Peptides</td>
<td>0</td>
<td>1</td>
<td>May 4, 2016</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>Compute the Number of Peptides of Given Total Mass</td>
<td>0</td>
<td>1</td>
<td>May 4, 2016</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>Generate the Theoretical Spectrum of a Cyclic Peptide</td>
<td>0</td>
<td>1</td>
<td>May 5, 2016</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---


---

Rosalind