Practical Bioinformatics

Mark Voorhies

6/14/2010

Goals

At the end of this class, you should have the confidence to take on the day to day tasks of "bioinformatics".

- Writing standalone scripts.
- Shepherding data between analysis tools.
- Aggregating data from multiple sources.
- Implementing new methods from the literature.

Goals

At the end of this class, you should have the confidence to take on the day to day tasks of "bioinformatics".

- Writing standalone scripts.
- Shepherding data between analysis tools.
- Aggregating data from multiple sources.
- Implementing new methods from the literature.

Goals

At the end of this class, you should have the confidence to take on the day to day tasks of "bioinformatics".

- Writing standalone scripts.
- Shepherding data between analysis tools.
- Aggregating data from multiple sources.
- Implementing new methods from the literature.

Goals

At the end of this class, you should have the confidence to take on the day to day tasks of "bioinformatics".

- Writing standalone scripts.
- Shepherding data between analysis tools.
- Aggregating data from multiple sources.
- Implementing new methods from the literature.

Goals

At the end of this class, you should have the confidence to take on the day to day tasks of "bioinformatics".

- Writing standalone scripts.
- Shepherding data between analysis tools.
- Aggregating data from multiple sources.
- Implementing new methods from the literature.

Goals

At the end of this class, you should have the confidence to take on the day to day tasks of "bioinformatics".

- We will focus on a single general-purpose tool: Python
- We will focus on a single research area: DNA microarrays

Goals

At the end of this class, you should have the confidence to take on the day to day tasks of "bioinformatics".

- We will focus on a single general-purpose tool: Python
- We will focus on a single research area: DNA microarrays

Course outline

- Introduction to Python
- File Formats
- Microarrays
    - Distance Metrics
    - Clustering
    - Statistics
- Sequence Analysis

Resources

Router:

- SSID: BMS270
- password: deoxyribose

Getting Python

- http://www.python.org/download
- python -m idlelib.idle

Course website:

- https://moodle.ucsf.edu/login/index.php
- http://www.library.ucsf.edu/services/galenaccounts

Resources on the course website:

- Syllabus
  - Papers and code (for downloading *before* class)
  - Slides and transcripts (available *after* class)
- On-line textbooks (Safari Bookshelf, Numerical Recipes, ...)
- Programs for this course (Python, Cluster3, JavaTreeView, ...)

Resources

Router:

- SSID: BMS270
- password: deoxyribose

Getting Python

- http://www.python.org/download
- python -m idlelib.idle

Course website:

- https://moodle.ucsf.edu/login/index.php
- http://www.library.ucsf.edu/services/galenaccounts

Resources on the course website:

- Syllabus
  - Papers and code (for downloading *before* class)
  - Slides and transcripts (available *after* class)
- On-line textbooks (Safari Bookshelf, Numerical Recipes, ...)
- Programs for this course (Python, Cluster3, JavaTreeView, ...)

Resources

Router:

- SSID: BMS270
- password: deoxyribose

Getting Python

- http://www.python.org/download
- python -m idlelib.idle

Course website:

- https://moodle.ucsf.edu/login/index.php
- http://www.library.ucsf.edu/services/galenaccounts

Resources on the course website:

- Syllabus
  - Papers and code (for downloading *before* class)
  - Slides and transcripts (available *after* class)
- On-line textbooks (Safari Bookshelf, Numerical Recipes, ...)
- Programs for this course (Python, Cluster3, JavaTreeView, ...)

Resources

Router:

- SSID: BMS270
- password: deoxyribose

Getting Python

- http://www.python.org/download
- python -m idlelib.idle

Course website:

- https://moodle.ucsf.edu/login/index.php
- http://www.library.ucsf.edu/services/galenaccounts

Resources on the course website:

- Syllabus
  - Papers and code (for downloading *before* class)
  - Slides and transcripts (available *after* class)
- On-line textbooks (Safari Bookshelf, Numerical Recipes, ...)
- Programs for this course (Python, Cluster3, JavaTreeView, ...)

Resources

Router:

- SSID: BMS270
- password: deoxyribose

Getting Python

- http://www.python.org/download
- python -m idlelib.idle

Course website:

- https://moodle.ucsf.edu/login/index.php
- http://www.library.ucsf.edu/services/galenaccounts

Resources on the course website:

- Syllabus
  - Papers and code (for downloading *before* class)
  - Slides and transcripts (available *after* class)
- On-line textbooks (Safari Bookshelf, Numerical Recipes, ...)
- Programs for this course (Python, Cluster3, JavaTreeView, ...)

Talking to Python: Nouns

```
# This is a comment
# This is an int (integer)
42
# This is a float (rational number)
4.2
# These are all strings (sequences of characters)
'ATGC'

"Mendel's Laws"

""">CAA36839.1 Calmodulin
MADQLTEEQIAEFKEAFSLFDKDGDGTITTKELGTVMRSLGQNPTEAEL
QDMINEVDADDLPGNGTIDFPEFLTMMARKMKDTDSEEEIREAFRVFDK
DGNGYISAAELRHVMTNLGEKLTDEEVDEMIREADIDGDGQVNYEEFVQ
MMTAK"""
```

Python as a Calculator

```
# Addition
1+1
# Subtraction
2-3
# Multiplication
3*5
# Division (gotcha: be sure to use floats)
5/3.0
# Exponentiation
2**3
# Order of operations
2*3-(3+4)**2
```

Fun with logarithms

In log space, multiplication and division become addition and subtraction:

$$\log(xy) = \log(x) + \log(y)$$
$$\log(x/y) = \log(x) - \log(y)$$

Therefore, exponentiation becomes multiplication:

$$\log(x^y) = y\log(x)$$

Also, we can change of the base of a logarithm like so:

$$\log_A(x) = \log(x)/\log(A)$$

Fun with logarithms

In log space, multiplication and division become addition and subtraction:

$$\begin{aligned} \log(xy) &= \log(x) + \log(y) \\ \log(x/y) &= \log(x) - \log(y) \end{aligned}$$

Therefore, exponentiation becomes multiplication:

$$\log(x^y) = y \log(x)$$

Also, we can change of the base of a logarithm like so:

$$\log_A(x) = \log(x) / \log(A)$$

Fun with logarithms

In log space, multiplication and division become addition and subtraction:

$$\begin{aligned} \log(xy) &= \log(x) + \log(y) \\ \log(x/y) &= \log(x) - \log(y) \end{aligned}$$

Therefore, exponentiation becomes multiplication:

$$\log(x^y) = y \log(x)$$

Also, we can change of the base of a logarithm like so:

$$\log_A(x) = \log(x)/\log(A)$$

Python as a Calculator

You measure the following $\log_2$ expression ratios for YFG:

$$\log_2\left(\frac{oxidative\ stress}{reference}\right) = 2.1$$
$$\log_2\left(\frac{heat\ shock}{reference}\right) = 3.2$$
$$\log_2\left(\frac{reductive\ stress}{reference}\right) = -1$$

1. What is the difference in expression between oxidative stress and heat shock as a $\log_2$ ratio?
2. If YFG is present at 10,000 mRNA/cell under reductive stress, how many copies are present under oxidative stress?

Saving and comparing objects

```
# Use a single = for assignment:
TLC = "GATACA"
YFG = "CTATGT"
MFG = "CTATGT"

# A name can occur on both sides of an assignment:
codon_position = 1857
codon_position = codon_position + 3

# Short-hand for common updates:
codon += 3
weight -= 10
expression *= 2
CFU /= 10.0
```

Checking values with print

```
# Use print to show the value of an object
message = "Hello, world"
print message
# Or several objects:
print 1,2,3,4
```

Saving and comparing objects

```
# Use double == for comparison:
YFG == MFG

# Other comparison operators:
#    Not equal:
TLC != MFG
#    Less than:
3 < 5
#    Greater than, or equal to:
7 >= 6
```

Saving and comparing objects

```
if(YFG == MFG):
    print "Synonyms!"

if(protein_length < 60):
    print "Probably too short to fold."
elif(protein_length > 10000):
    print "What is this, titin?"
else:
    print "Okay, this looks reasonable."
```

Collections of objects

```
# A list is a mutable sequence of objects
mylist = [1, 3.1415926535, "GATACA", 4, 5]
# Indexing
mylist[0] == 1
mylist[-1] == 5
# Assigning by index
mylist[0] = "ATG"
# Slicing
mylist[1:3] == [3.1415926535, "GATACA"]
mylist[:2] == [1, 3.1415926535]
mylist[3:] == [4,5]
# Assigning a second name to a list
also_mylist = mylist
# Assigning to a copy of a list
my_other_list = mylist[:]
```

Repeating yourself: iteration

```
# A for loop iterates through a list one element
# at a time:
for i in [1,2,3,4,5]:
    print i, i**2

# A while loop iterates for as long as a condition
# is true:
population = 1
while(population < 1e5):
    print population
    population *= 2
```

Summary statistics

Given a list of $\log_2$ expression ratios:
$x = [1.8, 2.0, 1.7, 1.9, 2.3, 1.6, 2.2, 1.8, 1.9, 4.0, 1.7]$

1. Print the corresponding expression ratio values
2. Calculate the mean (average) $\log_2$ ratio:

$$\bar{x} = \frac{\sum_{i=1}^{N} x_i}{N}$$

3. Calculate the mean expression ratio
4. In what situations do either of these mean values capture useful information about our measurements?

Verb that noun!

return_value = function(parameter, ...)
"Python, do *function* to *parameter*"

```
# Built-in functions
#  Generate a list from 0 to n-1
a = range(5)
#  Sum over an iterable object
sum(a)
#  Find the length of an object
len(a)
```

Summary statistics

Given a list of $\log_2$ expression ratios:
$x = [1.8, 2.0, 1.7, 1.9, 2.3, 1.6, 2.2, 1.8, 1.9, 4.0, 1.7]$

1. Calculate the mean (average) $\log_2$ ratio:

$$\bar{x} = \frac{\sum_{i=1}^{N} x_i}{N}$$

   using functions to simplify your calculation.

Verb that noun!

return_value = function(parameter, ...)
"Python, do *function* to *parameter*"

```
# Importing functions from modules
import math
math.sqrt(9)
math.log(8)/math.log(2)

from math import log
log(16)/log(2)
```

Summary statistics

Given a list of expression ratios:

$r = [4.00, 4.59, 3.73, 4.29, 5.66, 3.48, 5.28, 4.00, 4.29, 18.38, 3.73]$

1. Write a for loop to convert the list to $\log_2$ ratios
2. How can you do this conversion without destroying the original list?

Short-hand for converting lists

```
from math import log
log2 = log(2)
logratios = [log(i)/log2 for i in ratios]
```

New verbs

```
def function(parameter1, parameter2):
    """Do this!"""
    # Code to do this
    return return_value
```

Setting IDLE's working directory

OS X

- Open a terminal
- cd path/to/working/directory
- python -m idlelib.idle

Windows

- Use (file) explorer to find the path to your working directory and copy it to the clipboard.
- Right-click your IDLE menu item and choose "properties".
- Paste your working directory into the "Start in" field, making sure that it is quoted.

Loading and re-loading your functions

```
# Use import the first time you load a module
#   (And keep using import until it loads
#     successfully)
import my_module

my_module.my_function(42)

# Once a module has been loaded, use reload to
#   force python to read your new code
reload(my_module)
```

Make your own Fun

Write functions for these calculations:

1. Mean:

$$\bar{x} = \frac{\sum_i^N x_i}{N} \qquad (1)$$

2. Standard deviation:

$$\sigma_x = \sqrt{\frac{\sum_i^N (x_i - \bar{x})^2}{N - 1}} \qquad (2)$$

3. Correlation coefficient (Pearson's r):

$$r(x, y) = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2}\sqrt{\sum_i (y_i - \bar{y})^2}} \qquad (3)$$

Summary

- Python is a general purpose programming language.
- We can extend Python's built-in functions by defining our own functions (or by importing third party modules).
- We can define complex behaviors through control statements like "for", "while", and "if".
- We can use an interactive Python session to experiment with new ideas and to explore data.
- Saving interactive sessions is a good way to document our computer "experiments".
- Likewise, we can use modules and scripts to document our computer "protocols".
- Most of these statements are applicable to any programming language (Perl, R, Bash, Java, C/C++, FORTRAN, ...)

Summary

- Python is a general purpose programming language.
- We can extend Python's built-in functions by defining our own functions (or by importing third party modules).
- We can define complex behaviors through control statements like "for", "while", and "if".
- We can use an interactive Python session to experiment with new ideas and to explore data.
- Saving interactive sessions is a good way to document our computer "experiments".
- Likewise, we can use modules and scripts to document our computer "protocols".
- Most of these statements are applicable to any programming language (Perl, R, Bash, Java, C/C++, FORTRAN, ...)

Summary

- Python is a general purpose programming language.
- We can extend Python's built-in functions by defining our own functions (or by importing third party modules).
- We can define complex behaviors through control statements like "for", "while", and "if".
- We can use an interactive Python session to experiment with new ideas and to explore data.
- Saving interactive sessions is a good way to document our computer "experiments".
- Likewise, we can use modules and scripts to document our computer "protocols".
- Most of these statements are applicable to any programming language (Perl, R, Bash, Java, C/C++, FORTRAN, ...)

Summary

- Python is a general purpose programming language.
- We can extend Python's built-in functions by defining our own functions (or by importing third party modules).
- We can define complex behaviors through control statements like "for", "while", and "if".
- We can use an interactive Python session to experiment with. new ideas and to explore data.
- Saving interactive sessions is a good way to document our computer "experiments".
- Likewise, we can use modules and scripts to document our computer "protocols".
- Most of these statements are applicable to any programming language (Perl, R, Bash, Java, C/C++, FORTRAN, ...)

Summary

- Python is a general purpose programming language.
- We can extend Python's built-in functions by defining our own functions (or by importing third party modules).
- We can define complex behaviors through control statements like "for", "while", and "if".
- We can use an interactive Python session to experiment with. new ideas and to explore data.
- Saving interactive sessions is a good way to document our computer "experiments".
- Likewise, we can use modules and scripts to document our computer "protocols".
- Most of these statements are applicable to any programming language (Perl, R, Bash, Java, C/C++, FORTRAN, ...)

Summary

- Python is a general purpose programming language.
- We can extend Python's built-in functions by defining our own functions (or by importing third party modules).
- We can define complex behaviors through control statements like "for", "while", and "if".
- We can use an interactive Python session to experiment with. new ideas and to explore data.
- Saving interactive sessions is a good way to document our computer "experiments".
- Likewise, we can use modules and scripts to document our computer "protocols".
- Most of these statements are applicable to any programming language (Perl, R, Bash, Java, C/C++, FORTRAN, ...)

Summary

- Python is a general purpose programming language.
- We can extend Python's built-in functions by defining our own functions (or by importing third party modules).
- We can define complex behaviors through control statements like "for", "while", and "if".
- We can use an interactive Python session to experiment with. new ideas and to explore data.
- Saving interactive sessions is a good way to document our computer "experiments".
- Likewise, we can use modules and scripts to document our computer "protocols".
- Most of these statements are applicable to any programming language (Perl, R, Bash, Java, C/C++, FORTRAN, ...)