# Practical Bioinformatics

Mark Voorhies

4/2/2018

# Resources

Course website:

- http://histo.ucsf.edu/BMS270/

Resources on the course website:

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

# Homework

- E-mail Mark your python sessions (.ipynb files) after class
- E-mail Mark any homework code/results before tomorrow's class

# Goals

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

# Goals

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

- Analyzing data.

# Goals

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

- Analyzing data.
- Writing standalone scripts.

# Goals

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

- Analyzing data.
- Writing standalone scripts.
- Shepherding data between analysis tools.

# Goals

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

- Analyzing data.
- Writing standalone scripts.
- Shepherding data between analysis tools.
- Aggregating data from multiple sources.

# Goals

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

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

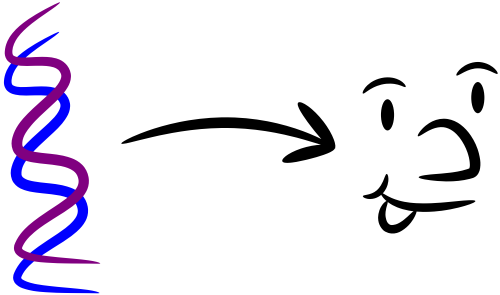
# Goals

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

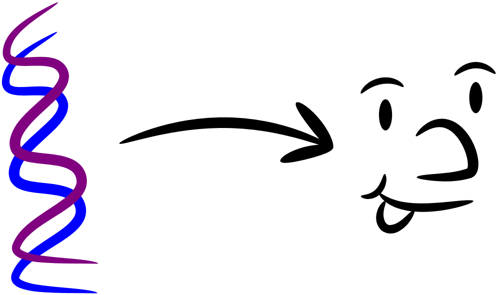
- Analyzing data.
- Writing standalone scripts.
- Shepherding data between analysis tools.
- Aggregating data from multiple sources.
- Implementing new methods from the literature.

This is also good preparation for communicating with computational collaborators.

# Course problems: expression and sequence analysis



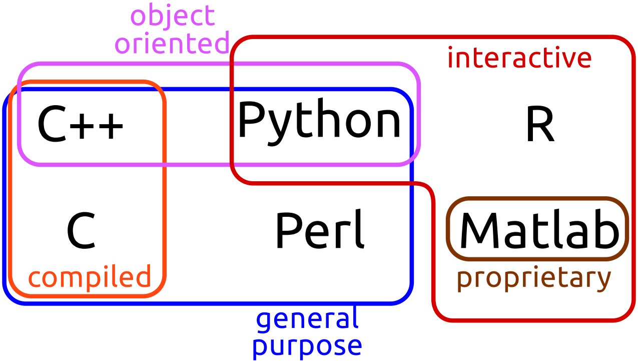
# Course problems: expression and sequence analysis



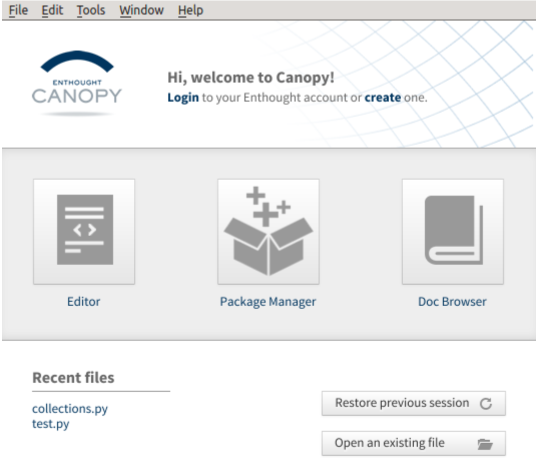
Part 2: Genotype
(Sequence analysis)

Part 1: Phenotype
(Expression profiling)

# Course tool: Python



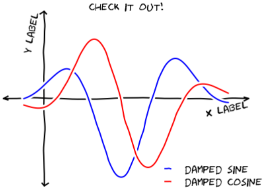
# Python distribution: Enthought Canopy



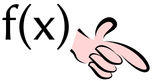
# Python shell: ipython (jupyter) notebook

In [5]:
```
np.random.seed(0)

ax = pylab.axes()

x = np.linspace(0, 10, 100)
ax.plot(x, np.sin(x) * np.exp(-0.1 * (x - 5) ** 2), 'b', lw=1, label='damped sine')
ax.plot(x, -np.cos(x) * np.exp(-0.1 * (x - 5) ** 2), 'r', lw=1, label='damped cosine')

ax.set_title('check it out!')
ax.set_xlabel('x label')
ax.set_ylabel('y label')

ax.legend(loc='lower right')

ax.set_xlim(0, 10)
ax.set_ylim(-1.0, 1.0)

#XKCDify the axes -- this operates in-place
XKCDify(ax, xaxis_loc=0.0, yaxis_loc=1.0,
        xaxis_arrow='+-', yaxis_arrow='+-',
        expand_axes=True)
```

Out[5]: <matplotlib.axes.AxesSubplot at 0x2fecbd0>

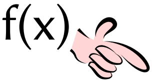


# Anatomy of a Programming Language

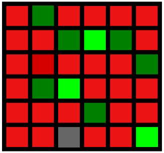



functions

# Anatomy of a Programming Language

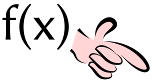



functions

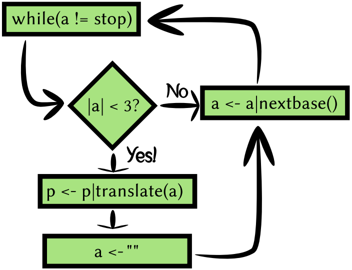


data structures

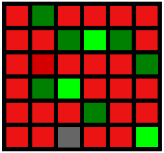
# Anatomy of a Programming Language




functions

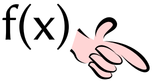



control statements

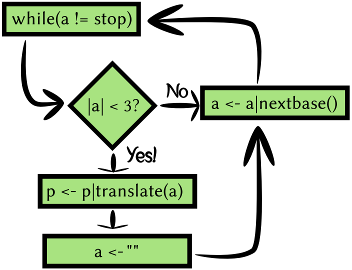


data structures

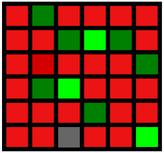
# Anatomy of a Programming Language



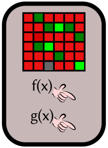

functions




control statements



data structures




objects

# 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
```

# Remembering 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
```

# Python as a Calculator

1. Calculate the molarity of a 70mer oligonucleotide with $A_{260} = .03$ using the formula from Maniatis:

$$C = \frac{.02A_{260}}{330L} \tag{1}$$

2. Calculate the $T_m$ of a QuickChange mutagenesis primer with length 25bp ($L = 25$), 13 GC bases ($n_{GC} = 13$), and 2 mismatches to the template ($n_{MM} = 2$) using the formula from Stratagene:

$$T_m = 81.5 + \frac{41n_{GC} - 100n_{MM} - 675}{L} \tag{2}$$

# Displaying 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)
# Older versions of Python use a
# different print syntax
print "Hello, world"
```

# 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
```

# 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)
```

# Verb that noun!

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

```
# Importing functions from modules
import numpy
numpy.sqrt(9)

import matplotlib.pyplot as plt
fig = plt.figure()
plt.plot([1,2,3,4,5],
         [0,1,0,1,0])

from IPython.core.display import display
display(fig)
```

# New verbs

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

# Summary

- Python is a general purpose programming language.

# 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).

# 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"

# 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"
- We can use an interactive Python session to experiment with new ideas and to explore data.

# 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"
- 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".

# 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"
- 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".

# 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"
- 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, ...)

# Homework: Make your own Fun

Write functions for these calculations, and test them on random data:

1. Mean:

$$\bar{x} = \frac{\sum_i^N x_i}{N}$$

2. Standard deviation:

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

3. Correlation coefficient (Pearson's r):

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