sLecture 1

I. Syllabus
    A. grading
        1. homework
        2. mention programming projects
            a) project examples
    B. outline
        1.
II. Introduction to self and course
III. What is UNIX?
    A. Originally it was operating system
        1. Emphasizing portability, multi-tasking, multi-user, time sharing
    B. UNIX history
        1. Originally developed by AT&T in 1969
        2. Popular because of portability and stability
    C. Since its inception it has become more of a concept than a single operating system. UNIX now means that the OS conforms to a set of standards known as POSIX which is developed by IEEE
        1. POSIX = Portable Operating System Interface
        2. IEEE = Institute of Electrical and Electronics Engineers
            a) Nonprofit organization
        3. show UNIX tree
    D. UNIX vs UNIX-like
        1. True unix = AIX, IRIX, HP/UX, Mac OS X (Darwin), others
        2. UNIX –like = LINUX, BSD
            a) We will be using LINUX – which itself has many flavors
                (1) RedHat, fedora, SUSE, Ubuntu, etc
                (2) Kernel developed by Linus Torvalds –thus linux
IV. How to UNIX?
    - Login
        - ssh
    - ls
    - pwd
    - mkdir example
    - ls
    - cd example
    - pwd
    - ls
    - vi our_first_script.py
        - x=4+4
        - print x
    - execute script
    - ls –l
    - explain permissions
    - chmod +x
    A. what is a path?

1. explain directory structure in UNIX
    a) hierarchical structure
        (1)

- UNIX command
    - command -options source destination
- cp our.. sldfkj
- ls
- rm jsldkfj
- cp our… to home directory
- have class do demonstration
- more
- tail
- history
- mv
- wc
- top
- ps
- tar
- *
- ~
- grep
- find
- ?
- fg
- bg
- &
- >
- <
- |
- nedit
- !

A. what is a shell?
    1. shell is the command line interpreter
    2. two most common are bash and tcsh (or some csh variant)
    3. echo $SHELL
    4. most important aspect of the shell for our purposes is that it defines our executable search path and other environment variables
        a) $SHELL is an environment variable
        b) environment variables are used by other programs to define information such as the locations of resources
            (1) important enviroment variable for our purposes will be PYTHONPATH
                (a) directory where we keep libraries
                    i) mkdir lib
                    ii) cd lib

iii) make ilikeyou.py

V. How to vi:

A. insert and view modes
B. save
C. quit
D. Inserting and typing text:
E. i insert text (and enter input mode)
F. $a append text (to end of line)
G. ESC re-enter command mode
H. J join lines
I. Cursor movement:
J. h left
K. j down
L. k up
M. l right
N. ^ beginning of line
O. $ end of line
P. 1 G top of document
Q. G end of document
R. <n> G go to line <n>
S. ^F page forward
T. ^B page backward
U. w word forwards
V. b word backwards
W.
X. Deleting and moving text:
Y. Backspace delete character before cursor
Z. (only works in insert mode)
AA.x delete character under cursor
BB.dw delete word
CC. dd delete line (restore with p or P)
DD. <n> dd delete n lines
EE.d$ delete to end of line
FF. dG delete to end of file
GG. yy yank/copy line (restore with p or P)
HH. <n> yy yank/copy <n> lines
II.
JJ. Search and replace:
KK.%s/<search string>/<replace string>/g
LL.
MM. Miscellaneous:
NN. u undo
OO. :w save file
PP.:wq save file and quit
QQ. ZZ save file and quit
RR. :q! quit without saving

VI. HTML

A. markup language vs programming language
1. markup language only deals with formatting, layout, and structure (metadata)
a) does not do any math

   2. programming language - all about math

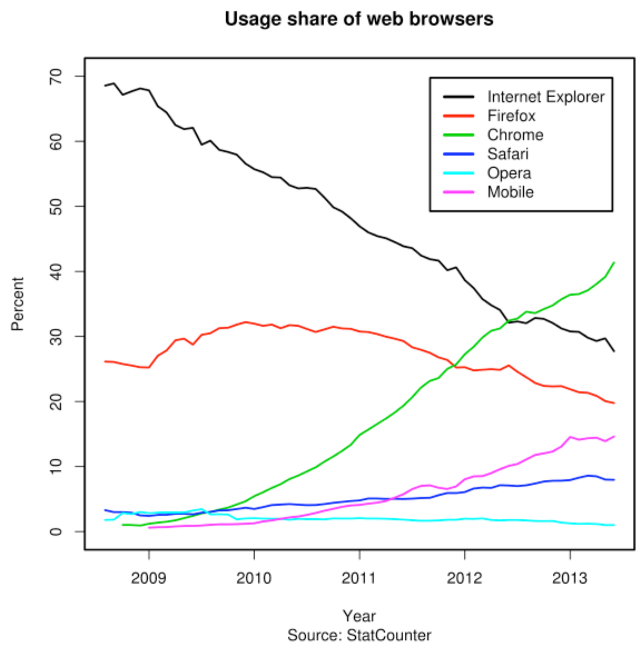
B. what browser sees vs. what you see

C. history

   1. originally there was HTML
   2. basic page

```
<!DOCTYPE html>
<html lang=”en”>
<head>
        <meta charset=”utf-8”/>
        <title>This is my first webpage</title>
</head>
<body>
        Hello World!
</body>
</html>
```

   3. webpages are simply marked up text with the format: elements, attributes, values
   4. elements are also called tags
   5. we will be using HTML5, but it’s useful to know a little about the fractious history of html
      a) browser wars
         (1) mosaic, vs. netscape vs. explorer
            (a) each vying for internet dominance.
            (b) each created proprietary tags that other browsers could not interpret
         (2)



   6. In 2000, the world wide web consortium (W3C) made a move to standardize html
      a) introduced XHTML as a way to standardize tags and syntax
      b) XHTML is HTML written in XML
         (1) XML is a markup language that allows one to annotate and share data

(2) main difference between the html and xhtml is that xhtml is more strict about syntax

7. in 2005 a group called the Web Hypertext Application Technology Working Group (WHATWG) frustrated with the slow developments and limitations of xhtml picked up html development
   a) ultimately produced the HTML5 specification
      (1) still being finalized
      (2) adopted by the W3C in 2007
      (3) most current browsers are mostly complaint
      (4) future versions will be called simply HTML
   b) it seems likely that HTML5 will be adopted by all browsers in the future

D. anatomy of a webpage

E. doctype
   1. because of different standards for what is HTML, you have to tell the browser how to interpret code
   2. frameset, transitional, strict
      a) html5
   3. standards vs. quirks mode
   4. document type definition (dtd)

F. why do we have HTML?
   1. formatting in text

G. css is for style

H. http://www.w3schools.com/

VII. HTML basics

A. elements, attributes, values
   1. inserting a link
      a) <a href="url"> text </a>
      b) elements are also called tags

B. making our 1st webpage
   1. look at example.html
   2. name it index.html or default.html or main.html or whatever else
   3. required elements
      a) doctype, html, head, title, body

C. h1, etc

D. inserting an image
   1. <img alt="image text" title="hover image text" height="npix" width="npix" src="path" />
      a) alt, height, width

E. inserting a link
   1. <a href="url"> text </a>

F. inserting a paragraph
   1. p

G. elements are block-level or inline
   1. block-level cause new lines
   2. inline do formatting without line break

H. <!-- comment -->

I. some inline tags
   1. em, b, u, i,

J. table
   1. tr
   2. td

   K. ol, ul
      1. li

VIII. Styling individual elements
   A. div
      1. block-level
      2. in example, create two articles
         a)
   B. span
      1. inline
   C. introduce two important attributes that can apply to any tag
      1. class = “name”
      2. id = “name”
         a) must be unique
         b) can only use once per page

IX. do example css
   A. defined by a “declaration”
   B. selector, property, value
   C. h1 {color:red;}
   D. comments
      1. /* comment */

X. More on styles

```
body {
        background: black;
        font-family: Arial, sans-serif;
        font-size: em, px
        color: white;
}
```

class corresponds to .

id corresponds to #

XI. fonts
   A. font-style: normal, italic, oblique
   B. font-weight: normal, bold, bolder

XII. text
   A. text-align: left, right, center, justify
   B. text-indent: px, em

XIII. layout
   A. css properties
      1. float : left, right, both
      2. padding: px, em
      3. width: px, em
      4. height: px, em
      5. border: px, em

XIV. links
   A. special because they depend on the state of the link
   B. a:link
   C. a:visited
   D. a:hover
      1. text-decoration: underline, overline, line-through, blink

I. HTML continued
   A. WYSIWYG
   B. static vs. dynamic pages

1. dynamic pages serve content based on behavior of client
    a) use scripting languages like PHP, asp, coldfusion, javascript
        (1) perfect example is facebook
            (a) uses php (I think) to generate pages based on user interactions
2.

II. colors

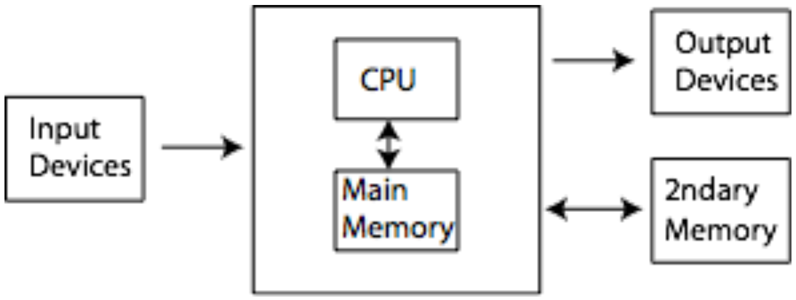
A. color: rgb(255,255,255), common colors, hex, other color models

B. any color can be expressed as a combination of red blue and green

1. css rgb example
    a) each color goes 0-255 (256 colors)
    b) that yields a total of 256*256*256= 16,777,216
2. about numbers
    a) we use base 10
3. hexadecimal
    a) hexadecimal is base 16
        (1) 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
    b) 123 in hex
    c) 256 16 1
    d) 0 7 B
    e) to specify hex in css color: #FFFFFF
4. humans use base 10 because we have 10 fingers
5. numbers
    a) computers express all data as a string of ones and zeros
        (1) this has to do with the nature of the way data is stored
        (2) on/off
            (a) for RAM it is charge on a capacitor
            (b) for hard drive it's magnetic
    b) express 53 in binary
        (1) 64 32 16 8 4 2 1
        (2) 0 1 1 0 1 0 1

C. What is a computer?

1. Our textbook defines a computer as "a machine that stores and manipulates information under the control of a changeable program"
    a) Draw a box with arrows



    b)
    c) main memory - RAM - random access memory goes away when power is off
    d) 2ndary memory - hard drive, CD, flash drive - stays regardless of power
    e) CPU - central processing unit

D. OK, so now what is a program?

    a) A program is a detailed step-by-step set of instructions telling a computer exactly what processes to perform

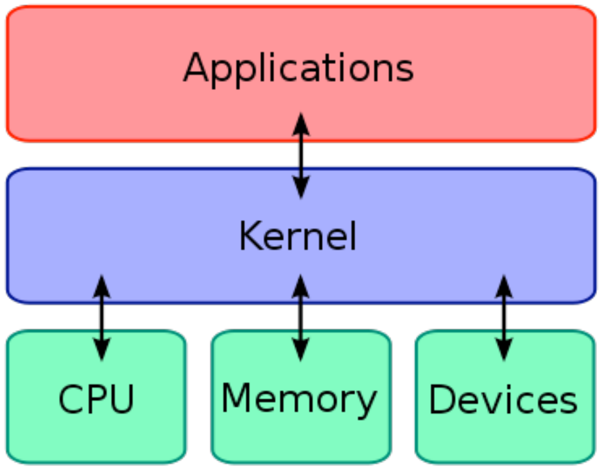
III.

- a) What are some languages?
  - (1) Make students come up with a list
  - (2) Python, C, C++, C#, Fortran, Perl, Java, PHP, BASIC, Matlab, cobal, tcl/tk, fortran, ruby, Delphi, sql, visual basic, unix programming,
- b) Types of languages
  - (1) Compiled and interpreted
    - (a) compiled
      - i) human readable code is read by a compiler and converted into an executable file
        - (1) executable contains computer's instructions in machine language
        - (2) machine language - a non-human readable set of instructions for the CPU
          - (a) load data
          - (b) perform arithmetic
          - (c) etc
          - (d) unique for each different computer
    - (b) interpreted
      - i) human readable code is read and converted to machine language on-the-fly
    - (c) each has advantages and disadvantages in terms of portability speed of execution, speed of development
- c) I have experience in these programs . . .

IV. Operating system - responsible for the management and coordination of activities and the sharing of the resources of the computer

V. Concept of the kernel

- A. In computer science, the kernel is the central component of most computer operating systems (OS). Its responsibilities include managing the system's resources (the communication between hardware and software components).[1] As a basic component of an operating system, a kernel provides the lowest-level abstraction layer for the resources (especially memory, processors and I/O devices) that application software must control to perform its function. It typically makes these facilities available to application processes through inter-process communication mechanisms and system calls.



- B.

VI. Announce test

A. Take on your own time, but I recommend using class time. I'll email out on Thursday morning. Turn it in under my door by 5:00 PM.

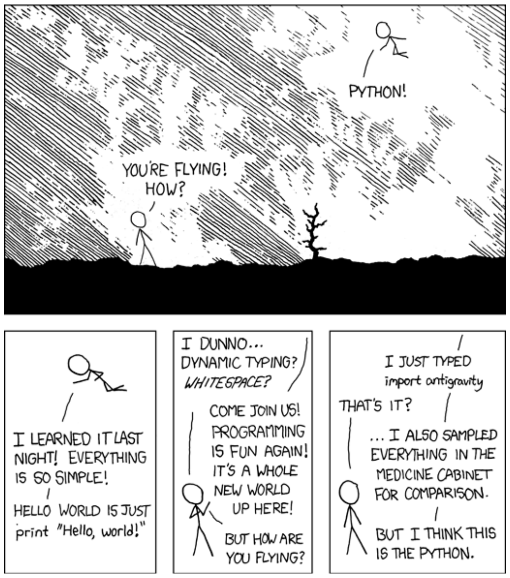
B. Will cover everything through today.
   1. Know UNIX commands
   2. HTML and CSS
   3. number systems
   4. python basics
   5. everything covered in lecture is fair game

VII. Python

A. Why Python?
   1. Free, open source, open development
   2. Available on most major operating systems (usually comes standard w/ OS)
   3. High-level language
      a) reads like English
      b) takes care of details automatically
      c) as opposed to low level language
         (1) must explicitly define instructions
   4. Intuitive syntax and structure
   5. Very powerful
   6. Increasingly used in scientific applications
      a) VMD, NAMD
      b) Chimera
      c) EMAN, Leginon, Spider
      d) CHARMM
      e) Great thing is that you can harness these programs to do exactly what you want them to
   7. Extensive library support
      a) Bioinformatics - biopython
      b) numerical processing
         (1) numpy, scipy
      c) graphing
         (1) matplotlib
      d) quick example

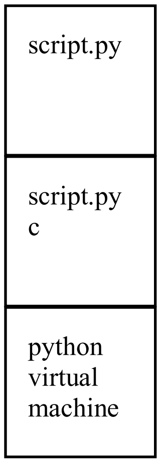
```
import numpy
from matplotlib import pyplot
r=numpy.arange(0,3.0,0.01)
theta=2*numpy.pi*r
pyplot.plot(theta,r**2)
pyplot.show()
pyplot.polar(theta,r**2)
pyplot.show()
```

   8. Great language for beginners
      a) Bringing programming to the masses. The point of this class is that you don't have to be a computer scientist to write a program. AND programming is an immensely powerful method of analyzing data


PYTHON!
YOU'RE FLYING! HOW?
I LEARNED IT LAST NIGHT! EVERYTHING IS SO SIMPLE!
HELLO WORLD IS JUST print "Hello, world!"
I DUNNO... DYNAMIC TYPING? WHITESPACE?
COME JOIN US! PROGRAMMING IS FUN AGAIN! IT'S A WHOLE NEW WORLD UP HERE!
BUT HOW ARE YOU FLYING?
I JUST TYPED import antigravity
THAT'S IT?
... I ALSO SAMPLED EVERYTHING IN THE MEDICINE CABINET FOR COMPARISON.
BUT I THINK THIS IS THE PYTHON.

9.
10. Written by Guido van Rossum a dutch programmer, first version 0.1 in1991 now at 3.4
    a) BD4L

VIII. Python interpreter
  A. first script
    1. write script that says hello world and does 2**10
  B. execution
    1. byte compilation



      (1) source byte code pvm
    2. pvm loops through byte code and sequentially executes line by line
  C. interpreter as calculator
  D. variable
    1. = means something different from mathematical = (i.e. assignment)
      a) This is perfectly legal: x=x+1
  E. expression vs. statement
    1. Expressions are sequences of operators, operands, and punctuators that specify a computation.
      a) expression is something that evaluates to return a value
      b) 3+2

        2. A statement, the smallest independent computer instruction, specifies an action to be performed.
            a) statement is basically a line of code. contains expressions. Different from expression in that it does not return a value
            b) x=3+2
    F. assignment statement

IX. How to write a program
    A. vi, nedit, IDLE GUI

X. literal – a representation of a specific value e.g. 3, “scott”

XI. operator – used to combine expressions e.g. *, **, +

XII. program
    A. a collection of statements
    B. assignment statement - the most basic statements
        1. most basic x=1
        2. more complex
            a) simultaneous assignment
                (a) x_squared, y_squared=x*x,y*y
                (b) x,y=x*x,y*y

XIII. Python’s core data types - give examples of each
    A. Numbers
    B. Strings
    C. Lists
    D. Dictionaries
    E. Tuples
    F. Files
    G. Sets
        1. s=set([1,2,3,3,5])
    H. Other core types
        1. Bool, types, None
    I. Programming units
        1. functions, modules, classes
    J. etc

XIV. Python is strongly but dynamically typed
    A. dynamically typed - means that types are determined automatically instead of having to explicitly define type for each variable
    B. strongly typed - you can only perform operations that are valid for type

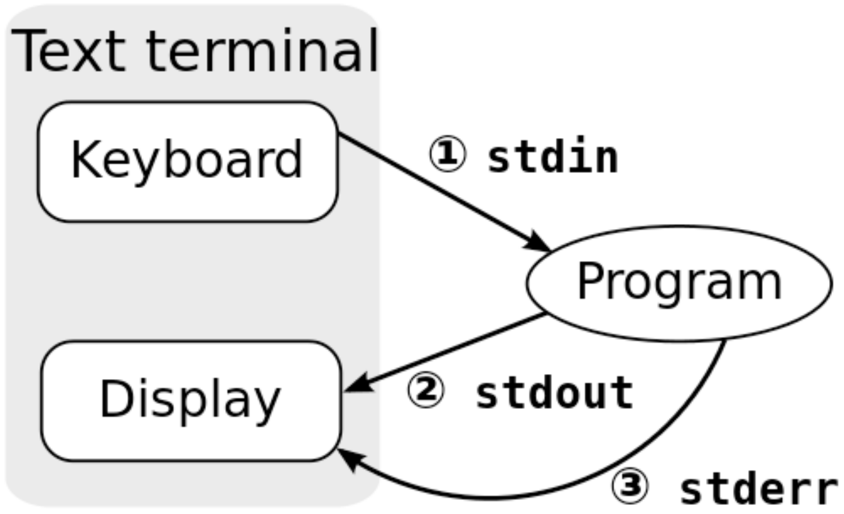
XV. Numbers
    A. ints, and floats
        1. http://en.wikipedia.org/wiki/Integer_(computer_science)
    B. Int
        1. just a whole number
        2. so how to represent the number 4 in computer?
            a) 100
            b) actually on a 32 bit computer, stored as
            c) 00000000000000000000000000000100
            d) has to do with how computer deals with numbers
                (1) address in memory is 32 bits long
                (2) so 2^{32} numbers can be represented on 32 bit computer
                    (a) half are positive and have are negative so range is $2^{32}/2 = 2^{31}$ negative and 2^{31} positive
                        i) range is -2^{31} to $2^{31}-1$

                (1) -1 accounts for zero
            (b) Limits of int
                i) Number of bits
                ii) -2**63 to 2**63-1
                iii) 2**62-1+(2**62-1) = largest number on 64 bit machine
                iv) limit of long is the amount of memory available
                v) math on longs is less efficient
                    (a)

C. bit level
    1. so for 64 bit computer 2^{64} ints can be stored
    2. if need a larger number, converted to a long int
        a) uses multiple addresses in memory
            (1) in practice, the largest number that computer can deal with depends on the amount of RAM available
D. type function
E. ==
F. while
G. floating point error
    1. sum=0.0
    2. while sum!=10:
        a) sum+=0.1
        b) print sum

XVI. std_in, std_out, std_err

A. three data streams for text data



1.
2. we'll only worry about stdin and stdout
3. input vs. raw_input
4. raw_input deprecated in 3.0
5. old way - input evaluates the string
    a) input in 2.x is equivalent to eval(raw_input)
6. in python 3.x *input* works the way *raw_input* did in python 2.x

B. raw_input – examples

```
x=raw_input("please enter some data for 'raw_input' ")
print type(x)
```

```
y=input('please enter some data for "input" ')
print type(y)
```

XVII. rules for var. names
- A. unlimited length
- B. start w/ letter or underscore
  1. _var, var1, but not 1var or any other special char
- C. case sensitive
  1. VAR is not the same as var
- D. cannot be reserved keywords p. 292 of textbook
  1. and, as,assert,break,class,continue,if,else,for,etc
     a) should also avoid built-ins like True, False, None, etc

XVIII. Modules and libraries
- A. library - a collection of useful functions or classes that can be imported and used in a program. also used to mean a collection of modules
- B. module - generally any relatively independent part of a program. In Python, the term is also used to mean a file containing code that can be imported and executed

XIX. demonstrate math library (aka module)
  1. math.sqrt calculate the distance between two points
- B. Write the dist_calc.py script
- C. nedit
- D. #! - shabang line
- E. explain env - man env

```
import math

x1=1
y1=2

x2=3
y2=4

dx=x1-x2
dy=y1-y2

d=(dx*dx + dy*dy)**0.5

d2=math.sqrt(dx**2 + dy**2)
print d,d2
```

XX. Numbers continued ints, longs, floats, and complex numbers
- A. Round
  1. round(1.5)
  2. round(1.5555, 2)
- B. Abs
  1. is not in math library
- C. Math.ceil
- D. Math.floor
- E. Integer division and modulo
  1. 5/3

2. 5%3
    a) in 3.0, / becomes 'true' division
        (1) in 3.x 5/3 = 1.666666667
    b) // becomes floor division

F. Complex #s
1. X=-6+3j
    a) x.real
    b) x.imag
        (1) x=-4+0j
        (2) x**0.5
    c) doesn't work with math lib

G.
1. in class exercise
    a) devise a one statement algorithm utilizing math.ceil or math.floor to round a float to nearest whole number

```
import math
import sys

x=float(raw_input('''This program will round numbers to the nearest
integer
please enter a float: '''))
print x
print type(x)
x=math.floor(x+0.5)
print x
```

    b) write a program where the input is a time in 24 hour format and some number of hours ahead. Output should be in 12 hour time
        (1) hoursahead=37
        (2) now=13
        (3) then=(now+hoursahead%24)%12
        (4) could even extend this by using integer division to determine number of days ahead

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

now=int(raw_input("Please enter a time in 24 hour format: ")) #time in
24 hour format
hoursahead=int(raw_input("Please enter some hours to add to the
original time: "))
then24=((now+hoursahead)%24)
then12=then24%12

print "The new time is", then24, then12

dayselapsed=hoursahead/24
print "Days elapsed is", dayselapsed
```

XXI. explain what a function is
A. named set of operations on some data
B. demonstrate dir function
1.

XXII. Sequences

- A. Strings
  - 1. s.split()

P	Y	T	H	O	N	

XXIII.

XXIV.

- A. Introduce lists
  - 1. list indices
  - 2. list slices
  - 3. [-1]
  - 4. [-1:]
  - 5. [0:5:2]

XXV. Sequences

- A. introduce tuples
- B. + - concatenation
  - 1. l=[1,2,3,4]
  - 2. l+l
- C. * - repetition
  - 1. l*4
- D. immutable
  - 1. s[4]='s' not legal

XXVI. Strings – Chapter 7

- A. ', ", and """
- B. string methods
  - 1. different data types have built-in functions called methods
    - a) can see string methods by
      - (1) dir(s)
      - (2) help(s.title)
  - 2. listed on p 173
  - 3. split
    - a) "Strings are lists of characters enclosed by quotes"
  - 4. strip
  - 5. title
  - 6. capitalize
  - 7. upper
  - 8. lower
  - 9. zfill
  - 10. find

XXVII. ASCII and Unicode

- A. ASCII – American Standard Code for Information Interchange
  - 1. American keyboard centric
  - 2. Published in 1963
- B. In the computer, each character is assigned to an integer
  - 1. A assigned to 65
  - 2. a assigned to 97
  - 3. can also designate them as hex
    - a) get them to figure out 65 in hex
    - b) print '\x41' - prints A
  - 4. now you see another reason why it is so important to keep track of types
  - 5. http://www.ascii-code.com/

C. Ord function returns ASCII encoding, chr returns char corresponding to int

D. Unicode – universal code attempts to encode all the characters for all the major languages

   1. ASCII = 2^8 = 256 while Unicode = 2^16 = 65536
      a) if, for instance, you want to represent Å, the unicode representation is
         (1) x=u''\xc5"
      b) using unicode chars is an advanced topic. To find out more, use Python website

XXVIII. grep usage

A. for example download stagg lab webpage

B. grep 'li' files {search files for lines with 'smug'}

C. grep '^li' files {'smug' at the start of a line}

D. grep '\^s' files {lines starting with '^s', "\" escapes the ^}

E. [ ] match one character with characteristics in the brackets

F. grep '[Ss]mug' files {search for 'Smug' or 'smug'}

G. grep 'B[oO][bB]' files {search for BOB, Bob, BOb or BoB }

H. grep '^$' files {search for blank lines}

I. grep '[0-9][0-9]' file {search for pairs of numeric digits}

XXIX. Regular expressions

operator	action
. (period)	Match any single character.
^ (caret)	Match the empty string that occurs at the beginning of a line or string.
$ (dollar sign)	Match the empty string that occurs at the end of a line.
A	Match an uppercase letter A.
a	Match a lowercase a.
\d	Match any single digit.
\D	Match any single non-digit character.
\w	Match any single alphanumeric character; a synonym is [:alnum:].
[A-E]	Match any of uppercase A, B, C, D, or E.
[^A-E]	Match any character except uppercase A, B, C, D, or E.
X?	Match no or one occurrence of the capital letter X.
X*	Match zero or more capital Xs.

operator	**action**
X+	Match one or more capital Xs.
X{n}	Match exactly n capital Xs.
X{n,m}	Match at least n and no more than m capital Xs. If you omit m, the expression tries to match at least n Xs.
(abcldef)+	Match a sequence of at least one abc and def; abc and def would match.

A.

a)

XXX. At this point in the course, I usually get impatient to start doing real programming, so let’s jump ahead a little to loops

A. sequences are iterable

B. Intro to loops

1. definite vs. indefinite
    a) definite - repeat for a defined number of iterations
    b) indefinite - have a conditional
2. for
    a) loops over a sequence
3. while
    a) requires a conditional
4. for now, we will focus on the for statement
5. These are both compound statements
    a) statements that are split over multiple lines
    b) syntax - ends in :
        (1) then must have indention
        (2) indention must stay the same
            (a) I recommend using tabs

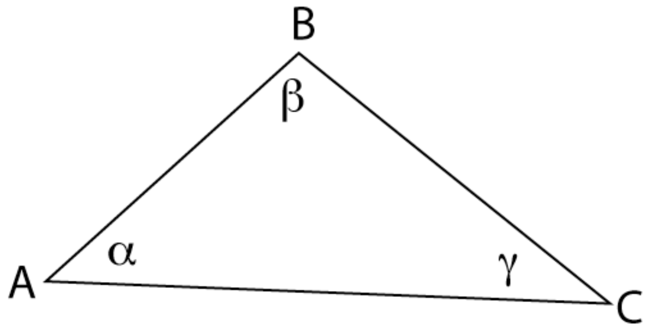
XXXI. demonstrate encode program

1. demonstrate encode program

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

import random

#ask for input
phrase=raw_input('Please input a phrase to be encrypted: ')

phrase=phrase.lower()

#create a key for encryption
key=random.randint(1,100)

newphrase="" #placeholder for encrypted string
#encode data
for l in phrase:
    i=ord(l)-96
    new=(i+key)%26
    newphrase+=chr(new+96)

print "The encoded phrase is"
print newphrase

#decode
print "Now I will decode the phrase given the key", key
for l in newphrase:
    i=ord(l)-96
    new=(i-key)%26
    print chr(new+96)

print "Done!"
```

XXXII.

XXXIII. Go over homework

A. 

B. $$\cos(\gamma) = \frac{a^2 + b^2 - c^2}{2ab} .$$

C. use this homework to demonstrate the use of split

XXXIV. Formatted output i.e. pretty printing

- A. print example
  1. repr - show string representation of data
  2. eval - evaluate string as specific data type
  3. l=[1,2,3]
  4. print l
     - a) internally does repr
  5. input function
     - a) does eval on input string
- B. %[name][flags][width][.precision]code
  1. name - dictionary key
  2. flags
     - a) justification - ‘ - ‘ is left justified
     - b) + - means show positive or negative explicitly
  3. width
     - a) x.y
       - (1) x - total minimum width

                    (2) y - precision, i.e. number of decimal places
            4. format codes
                a) %s, %c, %d, %e, %f, %%
                b) found on p. 181
                    (1) or http://docs.python.org/2/library/string.html
                c) s string
                d) r string but uses repr instead of str
                e) c character
                f) d integer in the decimal numbering scheme
                g) i integer
                h) o octal
                i) x hex
                j) X uppercase hex
                k) e float exponent
                l) f float
            5. examples
                a) print “%5.3f” % 3.2
                b) print “%+5.3f” % 3.2
                c) print "%-5s" % "sco"
                d) print "%5s" % "sco"
                e) print "%-5d" % 5
                f) print "%5d" % 5
                g) x=1234103482013.130230958
                h) print “%5.3e” % x
                i) dict
                    (1) d={‘owner’:‘Scott’, ‘pet’:‘Boone’}
                    (2) print “The owner of %(pet)s is %(owner)s” % d
        C. escape sequences \ \n \\ etc
            1. p 159
                a) or http://docs.python.org/2/reference/lexical_analysis.html
            2. \\, \a – bell, \n, \t, \’, \”
                    (1)
XXXV. The string format method only in python 2.6 and 3.x
        A. s="{0}, and a, {1}, and a {2}"
        B. s.format("one”, "two", 3)
        C. since we can’t even practice these, I won’t go over them
        D. % may disappear (deprecated), but currently being debated
            1. I sincerely doubt it
XXXVI. Lists
        A. are mutable while strings are immutable
        B. ex
            1. s=‘Scott’
                a) s[1]=’t’
                b) notice that string methods return a new string
                c) help (s.replace)
                d) s.replace('c','t',1)
            2. l=[0,1,2]
        C. Append – to end
        D. Extend – adds sequence to end
        E. l+l2 vs l.extend(l2)
            1. addition returns a new list while extend modifies original

F. Count – counts occurrences of x
G. Index – returns smallest i where i==x
H. Insert – inserts x into s at i
I. Pop – returns I and removes from s
J. Remove – searches for x and removes it
K. Reverse – reverses in place
L. Sort – sorts based on cmp function
M. del l[i]
N. del[i:k'
O. range(10)

XXXVII. Nested lists

A. m=[[1,2,3],
B.   [4,5,6],
C.   [7,8,9]]

XXXVIII. Uses of for n in l: and for n in range(x):

A. Range
   1. Returns a list
B. Two lists example
   1. Names=["Steve", "Elijah", "Arlo", "Frank"
   2. Grades=[95.5, 69.3, 81.0, 89.7]
C. for n,m in enumerate():

XXXIX. demonstrate that copies of lists and lists with internal references to lists are mutable and can result in some inconsistencies

A. l1=[1,2,3]
B. l2=[4,5,6]
C. l3=[l1,l2]
D. l1[0]=10
E. demonstrate that this would be different with an int or other immutable

```
x=10
y=x
x=9
print x, y

l=[1,2,x]
x=8
print l

l1=[1,2,3]
l2=[4,5,6]
l3=[l1,l2]
l1[0]=10
print l3
```

F. tmp

XL. file in and out

A. open("file")
   1. mode r, w, a – append to end, r+ - read and write
B. f.read - read all lines into a single string
C. f.readline
D. f.readlines
E. f.tell – report the current position
F. f.seek – move to a particular position

G. f.close
H. f.write()

XLI. demonstrate sys.argv()

XLII. demonstrate working with a pdb file

A. show what it is with chimera

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

from matplotlib import pyplot

#read in pdb file
f=open('2FA9noend.pdb','r')
lines=f.readlines()
f.close()

temperaturelist=[]
#loop over lines
for line in lines:
    words=line.split()
    temp=float(words[10])
    temperaturelist.append(temp) #build up temp factor list


#write out new list of temp factors
f=open('templist.txt','w')
for n in temperaturelist:
    f.write('%.2f\t' % (n))


pyplot.plot(temperaturelist)
pyplot.show()
```

XLIII. Control of flow

A. A program is sequential execution of statements
B. Can control sequence with loops and conditionals

XLIV. Conditional statements

A. if, elif, else
  1. require a Boolean
    a) special type
      (1) True or False
  2. also, any other literal but 0, None, empty sequence returns true
    a) True
    b) False
    c) 1 = True
    d) 0 = False
    e) None = False
    f) I don’t recommend depending on this
  3. example
    a) write script

```
l=[“Luke”,”Anakin”,”Palpatine”]
for name in l:
        if name==”Luke”:
            force="light"
        elif name=="Palpatine":
          force="dark"
        else:
          force="undetermined"
  print name, force
```

B. Conditional (relational) operators
   1. magnitude tests
      a) <
      b) <=
      c) >
      d) >=
      e) == - identity test
      f) != not the same - ! is negation
   2. is - tests to see if the objects are the same
   3. not
      a) negation - reverses boolean
   4. is not
   5. in
   6. not in
   7. Examples
      a) x=1
      b) y=1
      c) x==y, x is y
         (1) in this case value, and object are the same
      d) x=1.0
      e) y=1.0
      f) x is y
         (1) in this case, b/c we are using floats, objects are different
      g) x=[1,2,3]
      h) y=[1,2,3]
      i) x is y
         (1) false
      j) x=y
      k) x is y
         (1) true
         (2) because they are the same object

C. Counting As, Bs, and Cs

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

    import random, sys

    ntests=int(sys.argv[1])
    grades=[]
    for n in range(ntests):
        grades.append(random.gauss(80,10))
        print grades[-1],

    a=0
    b=0
    c=0
```

```
d=0
f=0

print
for n in grades:
    if n >= 90:
        a+=1
    elif n>= 80:
        b+=1
    elif n>=70:
        c+=1
    elif n>=65:
        d+=1
    else:
        f+=1
print "As =", "*"*a
print "Bs =", "*"*b
print "Cs =", "*"*c
print "Ds =", "*"*d
print "Fs =", "*"*f
```

XLV. Comparison operators

A. A logician's wife is having a baby. The doctor hands the baby to the dad. His wife asks if it's a boy or girl. The logician replies "Yes."

B. return Boolean (type bool)

C. Rules about *and, or,* and *not*

1.

P	Q	P and Q
T	T	T
T	F	F
F	T	F
F	F	F

P	Q	P or Q
T	T	T
T	F	T
F	T	T
F	F	F

P	not P
T	F
F	T

XLVI. Conditionals - order of operations (operator precedence)

A. a=True

B. b=False

C. c=True

D. what happens when you combine not, and, or

1. a or not b and c
2. a or ((not b) and c)\

E. A programmer's wife asks him to pick up a loaf of bread and, if they have eggs, get a dozen. The programmer comes home with a dozen loaves of bread.

F. etc

1. full list of operator precedence on p. 109
2. http://docs.python.org/2/reference/expressions.html#operator-precedence
    a) in reverse order of precedence
    b) yield, lambda, x if y else z, or, and, not in, is, <, <=,>,>=, ==,!=, |, ^, & <<,>>,+,-,*,%, / ,// , -x, **, l[n], l[1:2:3], func(), x.attr, (tuple), [list], {dict}, (expr)

I. Ternary expression
  A. expression if bool else expression

```
if X:
        A=1
else:
  A=3

becomes

A=1 if True else 3
```

    1.

II. Do a quick example
  A. use and or to parse pdbs more properly with REMARKS and HETATMS

III. Loops
  A. For – definite
  B. While – indefinite
    1. for statements are generally faster than while with a counter b/c for performance has been thoroughly optimized
        a)
  C. for and while
    1. three control of flow statements that are important in loops are:
        a) continue, pass, and break
            (1) continue - go to the next iteration
            (2) break - escape out of the loop
            (3) pass - do nothing placeholder
            (4) make diagram that demonstrates the behavior of these
                (a) while <condition>:
                    i) <statements 1>
                    ii) if <condition>:
                        (1) break # exit the loop and do not execute the else statements
                    iii) if <condition>:
                        (1) continue # skip statements 2 and go on to the next iteration
                    iv) <statements 2>
                (b) else:
                    i) <statements 3> # run if we didn't hit a break
                    ii) what is the difference between having else and just having code outside of the loop
                        (1) else is only executed if we don't hit a break
                        (2) <u>if there is a break - execution skips over else</u>
        b) here mention the different behavior of if and elif in above diagram
            (1) mention how boths ifs will always be executed
            (2) if one of them was an elif, it would only be executed if the if above returned False

D. both for and while can have an optional else statement
    1. this is executed if the loop **doesn't** hit a break

IV. Indefinite loops
  A. while example with break from book p. 332
    1. y=13
    2. x=y//2 # tests for factor of 2, makes code enter loop and determines greatest possible factor
    3. while x > 1:
      a) if y%x == 0:
        (1) print "%d has factor %d" % (y, x)
        (2) break
      b) x -=1
    4. else:
      a) print "%d is prime" % y

V. Dictionaries
  A. Also called a mapping or associative array or hashes
  B. <u>Unordered and mutable</u>
  C. Dictionary usage
    1. d={'Scott':'man', 'Boone':'dog'}
    2. can be nested
      a) d={'Scott':{'grades':[100,90,80],'gender':'male'}}
    3. del d[key]
    4. len(d)
    5. d.clear()
    6. d.copy()
    7. d.get(key)
    8. d.has_key(key)
    9. d.items()
    10. d.keys() - returns list is 2.x, but returns an iterator in 3.x. An iterator is a different data type that allows us to loop in different ways. We will cover this later. So, in 3.x, we would have to say list(d.keys()).
    11. d.popitem()
    12. d1.update(d2)
    13. d.values()
    14. keys need not always be strings; they can be any nonmutable type
      a) n=1
      b) d{n:[1,2,3,4]}
  D. Atom coords example
    1. modify homework code
  E. Arbitrarily expandable
    1. d[key]=value

VI. Go over homework

VII. Functions
  A. A named block of statements that perform operations
    1. maximize code reuse and minimize redundancy
  B. def name():
    1. statements
  C. Compound statement
    1. Get students to list compound statements
  D. Examples
    1. def do_nothing():

        2. pass
        3. def print_hello():
            a) print “hello”
        4. def iseven(num):
            a) print num%2==0
        5. def iseven():
            a) return (num%2==0)
    E. Emphasize modularity
    F. Functions are not interpreted when they are defined – only when they are called
    G. Example intersection

```
def intersection(seq1, seq2):
        inboth=[]
        for n in seq1:
                if n in seq2:
                        if n not in inboth:
                                inboth.append(n)
        return inboth
```

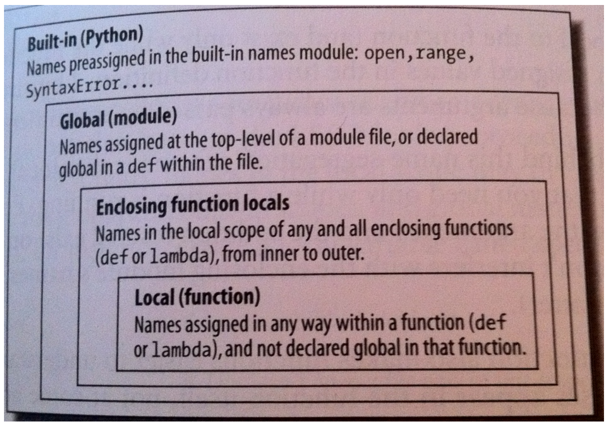
    H. generate two lists for above via random.randint and use in example
    I. Docstring
        1. Demonstrate docstring in python interpreter
        2. help, func.__doc__
    J. Returning multiple values from a function

```
import math

def mean_and_sig(seq):
        avg=sum(seq)
         n=float(len(seq))
        avg=avg/n
        sig=0
        for x in seq:
                sig+=(x-avg)**2
        sig=math.sqrt(sig/(n-1))
        return avg,sig
l=[2,2,7,8,2,8,3,7,3,3]

print mean_and_sig(l)
```

    K. Passing mutable and immutable variables

VIII. Intro to namespace and scope
    A. demonstrate that the namespace of interpreter is __main__
    B. python
        1. __name__ - name attribute of current namespace
        2.
    C. if name==”__main__”:
        1. example
            a) make a module with interesting functions
            b) use if name == main to test functions
    D. demonstrate how to change environment variables so that lib dir is in PYTHONPATH
    E. functions have their own “local” namespace
        1. way to lookup value associated with variable follows scoping rules

IX. Scoping rules

A. scopes - the places where variables are defined and looked up
   1. aka namespace
   2. follows lexical scoping rules - variable scopes are determined entirely by the locations of the variable in the code
   3. LEGB rule - the order in which names are looked up
      a) 

Built-in (Python)
Names preassigned in the built-in names module: `open`, `range`, `SyntaxError`....

Global (module)
Names assigned at the top-level of a module file, or declared global in a `def` within the file.

Enclosing function locals
Names in the local scope of any and all enclosing functions (`def` or `lambda`), from inner to outer.

Local (function)
Names assigned in any way within a function (`def` or `lambda`), and not declared global in that function.

B. Local

```
x=99

def f1():
        x=88
        print "in function", x


if __name__ == "__main__":
        print __name__

        print "before function", x
        f1()

        print "after function", x
```

C. Enclosing:

```
x=99
def f1 (x):
    x=88
    def f2():
            print x
    f2()
f1()
```

D. Global variables

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

x=40

def p():
     print x

p()
```

E. p knows about x because x is in global (module) scope
F. p, however cannot modify x

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

x=40

def p():
        x+=1
        print x

p()
```

G. we can use global to make the scope of a variable inside the function global

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

x=40

def p():
        global x
        x+=1
        print x

p()
```

H. useful in some situations, but can lead to unwanted dependencies. Better practice is to pass variable as an argument
I. for instance, what is x here?
   1. depends on when functions are called

```
x=40

def p1():
        global x
        x=80
        print x

def p2():
        global x
        x=160
        print x

p()
```

J. builit ins
   1. predefined variables (objects) that are instantiated on startup
   2. can get list of builtins by:
      a) python
      b) dir (__builtins__)
      c) notice that these aren't reserved keywords
         (1) str=1 is perfectly legal

X. Positional arguments

A. Arguments are passed in the order they are specified
B. Specifying keyword arguments
    1. useful for functions that have a lot of parameters
    2. Must follow positional args
    3.
C. Combining positional and keyword args
D. A function can be assigned to a variable

XI. Arbitrary args
    a. * specifies arbitrary positional args
    b. ** specifies arbitrary keyword args

```
def mean(name,*nums):
    print type(nums)
    if len(nums)==0:
        return 0.0
    else:
        sum=0.0
        for x in nums:
            sum+=x
        m=sum/len(nums)
        print name, m
        return m

def grades(course,**gradebook):
    print type(gradebook)
    print gradebook
    for key in gradebook.keys():
        sum=0.0
        for grade in gradebook[key]:
            sum+=grade
        avg=sum/len(gradebook[key])
        print course, key, avg

mean('scott', 87,88,91,90)
grades('bch5887',scott=[87,88,91,90], anne=[98,93,100,96],boone=[77,63,82,75])
```

1. lambda
    a. Creates one line anonymous functions
    b. def is a statement
    c. lambda is an expression
    d. Times=lambda x,y: x*y
    e. Type times
    a. Times (3,4)
    b. l=[(lambda x : x**2), (lambda x : x**3), (lambda x : x**4)]
    c. for f in l:
        i. print f(2)
2. list comprehension
    a. [*expression* for *variable* in *list*]
    b. l=range(10)
    c. [x**2 for x in l]
    d. m=[[1,2,3],
          [4,5,6],
          [7,8,9]]

        e. [x[0] for x in m]
    3. map
        a. like list comprehension, but you can apply a function to list
        b. takes a function and a sequence repeatedly calls the function using the sequence as args. returns result as sequence

```
def f (x):
        if x%2==0:
                return x
        else:
                return x+1
l=range(10)
l2=map(f,l)
```

        c. map(lambda x: x*x, range(5))

XII. Sorting
    A. sort function takes function as a parameter
        1. function should be in the form

```
sortfunc(x,y):
    if x<y:
            return -1 #any negative number
    elif x>y:
            return 1
    else:
            return 0
```

```
l=[["Scott",90],["Boone",70],["Anne",80]]
l.sort()
def gradesort(x,y):
        return x[1]-y[1]
l.sort(gradesort())
```

            i)

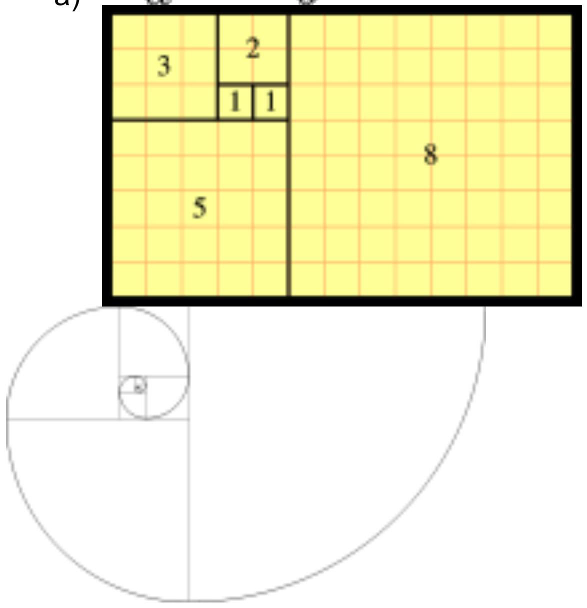
XIII. Recursive functions:
    A. a function that calls itself
    B. a way of looping without using for or while
    C. sum example from book

```
def mysum(L):
        print L
        if not L:
                return 0
        else:
                return L[0] + mysum(L[1:])
mysum([1,2,3,4,5])
```

    D. Must have terminating condition

```
def power(x,n):
        if n==0:
                return 1
        elif n>0:
                return x*power(x,n-1)

def factorial(n):
        If n==0
                Return 1
        Else:
                Return n*factorial(n-1)
```

E. Define the fibonacci series and see if we can write the fibonacci function as a class.
   1. 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144
   2. ratio of height to width = phi = 1.618...
      a) $$\frac{a+b}{a} = \frac{a}{b} \equiv \varphi,$$



```
def fibonacci(n):
        #returns the nth element of the fib series skipping 0 and 1
        If n < 2:
                Return 1
        Else:
                Return fibonacci(n-1)+fibonacci(n-2)
```

XIV. good libraries
   A. http://www.scipy.org/Topical_Software

XV. numpy, scipy, pylab
   A. history
      1. originally there were numeric and numarray
         a) developed by various groups - mainly the Space Telescope Science Institute
            (1) sponsored by NASA for processing images from Hubble
      2. written in C and Fortran compiled into objects for python
         a) fast
      3. became very popular
      4. pylab started as matplotlib as a free matlab
         a) used numeric for numerical operations
         b) also became very popular

5. each of the branches started developing things that the other packages wanted and there was also some overlap
    a) this gave birth to the scipy open source, open development project

B. scipy
1. attempts to unify all packages into scipy, numpy, and pylab (matplotlib)
2. incredibly powerful, better than matlab, free

XVI. http://www.scipy.org/Tentative_NumPy_Tutorial

A. basic object is the ndarray
1. table of elements (usually numbers), all of the same type, indexed by integers.
2. multiple dimensions called axes. The number of axes is rank.

B. basic methods
1. ndarray.ndim - the number of axes (dimensions) of the array. AKA rank
2. ndarray.shape - the dimensions of the array. This is a tuple of integers indicating the size of the array in each dimension.
3. ndarray.size - the total number of elements of the array. This is equal to the product of the elements of shape.
4. ndarray.dtype - the type of the elements in the array. Additionally NumPy provides types of its own. numpy.int32, numpy.int16, and numpy.float64 are some examples

C. Basics
1. a=numpy.arange(10,100)
2. a.max()
3. a.min()
4. a.argmax()
5. a.argmin()
6. a.mean()
7. a.std()

```
from numpy  import *
>>> a = arange(10).reshape(2,5)
>>> a
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]])
>>> a.shape
(2, 5)
>>> a.ndim
2
>>> a.dtype.name
'int32'
>>> a.itemsize
4
>>> a.size
10
>>> type(a)
numpy.ndarray
>>> b = array([6, 7, 8])
>>> b
array([6, 7, 8])
>>> type(b)
numpy.ndarray
```

D.

E. slicing[start:stop:step, start:stop:step]
F.
G.
```
>>> a = arange(10)**3
>>> a
array([  0,   1,   8,  27,  64, 125, 216, 343, 512, 729])
>>> a[2]
8
>>> a[2:5]
array([ 8, 27, 64])
>>> a[:6:2] = -1000    # equivalent to a[0:6:2] = -1000; 
from start to position 6, exclusive, set every 2nd element 
to -1000
>>> a
array([-1000,     1, -1000,    27, -1000,   125,   216,   
343,   512,   729])
>>> a[ : :-1]                                 # reversed a
array([  729,   512,   343,   216,   125, -1000,    27, -
1000,     1, -1000])
>>> for i in a:
...         print i**(1/3.),
...
nan 1.0 nan 3.0 nan 5.0 6.0 7.0 8.0 9.0
```
Multidimensional arrays can have one index per axis. These indices are given in a tuple separated by commas:

```
>>> def f(x,y):
...         return 10*x+y
...
>>> b = fromfunction(f,(5,4),dtype=int)
>>> b
array([[ 0,  1,  2,  3],
       [10, 11, 12, 13],
       [20, 21, 22, 23],
       [30, 31, 32, 33],
       [40, 41, 42, 43]])
>>> b[2,3]
23
>>> b[0:5, 1]                       # each row in the 
second column of b
array([ 1, 11, 21, 31, 41])
>>> b[ : ,1]                        # equivalent to the 
previous example
array([ 1, 11, 21, 31, 41])
>>> b[1:3, : ]                      # each column in the 
second and third row of b
array([[10, 11, 12, 13],
       [20, 21, 22, 23]])
```

When fewer indices are provided than the number of axes, the missing indices are considered complete slices:

```
>>> b[-1]                                   # the last row.
Equivalent to b[-1,:]
array([40, 41, 42, 43])
```

The expression within brackets in `b[i]` is treated as an `i` followed by as many instances of `:` as needed to represent the remaining axes. NumPy also allows you to write this using dots as `b[i,...]`.

The **dots** (`...`) represent as many colons as needed to produce a complete indexing tuple. For example, if `x` is a rank 5 array (i.e., it has 5 axes), then

- `x[1,2,...]` is equivalent to `x[1,2,:,:,:]`,
- `x[...,3]` to `x[:,:,:,:,3]` and
- `x[4,...,5,:]` to `x[4,:,:,5,:]`.

```
>>> c = array( [ [[  0,  1,  2],              # a 3D array
(two stacked 2D arrays)
...                [ 10, 12, 13]],
...
...               [[100,101,102],
...                [110,112,113]] ] )
>>> c.shape
(2, 2, 3)
>>> c[1,...]                                    # same as
c[1,:,:] or c[1]
array([[100, 101, 102],
       [110, 112, 113]])
>>> c[...,2]                                    # same as
c[:,:,2]
array([[  2,  13],
       [102, 113]])
```

Iterating over multidimensional arrays is done with respect to the first axis:

```
>>> for row in b:
...         print row
...
[0 1 2 3]
[10 11 12 13]
[20 21 22 23]
[30 31 32 33]
[40 41 42 43]
```

However, if one wants to perform an operation on each element in the array, one can use the `flat` attribute which is an iterator over all the elements of the array:

```
>>> for element in b.flat:
...         print element,
...
0 1 2 3 10 11 12 13 20 21 22 23 30 31 32 33 40 41 42 43
```

1. first let's make a 2d array
   a) a=arange(1,10

XVII. fitting a line

$$y = mx + b$$

$$SS_{xx} = \sum_{i=1}^{n} (x_i - \bar{x})^2$$

$$SS_{xx} = \left(\sum_{i=1}^{n} x_i^2 \right) - n\bar{x}^2$$

$$SS_{yy} = \left(\sum_{i=1}^{n} y_i^2 \right) - n\bar{y}^2$$

$$SS_{xy} = \left(\sum_{i=1}^{n} x_i y_i \right) - n\bar{x}\bar{y}$$

$$a = \frac{SS_{xy}}{SS_{xx}}$$

$$b = \bar{y} - a\bar{x}$$

$$r^2 = \frac{SS_{xy}^2}{SS_{xx} SS_{yy}}$$

A.
B. a=SSxy/SSxx
C. b=meany - a*meanx
D. fitting a line with scipy

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

import numpy
from matplotlib import pyplot
import scipy

####create data
n=1000
x=numpy.linspace(0,15,n)

a=1.3
b=3

y=a*x+b
```

```
noise=numpy.random.normal(0,1,n)

#y=noise+y
y+=noise


#### analyze data
####old school way
meanx=x.mean()
meany=y.mean()

sx=(x**2).sum()
sy=(y**2).sum()
sxy=(x*y).sum()

ssxx=sx-n*meanx**2
ssyy=sy-n*meany**2
ssxy=sxy-n*meanx*meany

a=ssxy/ssxx
b=meany-a*meanx

print "old school", a, b

#### scipy way
p=scipy.polyfit(x,y,1)
print p


linex=numpy.array([x[0],x[-1]])
liney=numpy.array([p[1],p[0]*linex[-1]+p[1]])
pyplot.plot(x,y,'ro')
pyplot.plot(linex,liney)
pyplot.show()
```

XVIII. AUC example

A. presentation
B. go over AUC presentation and code

XIX. linear algebra

A. $$(AB)_{i,j} = \sum_{k=1}^{p} A_{ik}B_{kj},$$
B.
C. a=[[1,2],[0,3]] , b=[[2,3],[2,1]]
D. =[6,5],[6,3]
E. rotation matrices
F. 1NSF.pdb - snare detangler

XX. Demonstrate sys, os, glob

A. help os
    1. show list of functions and demonstrate

   B. help sys
   C. help glob

XXI. option parsing

   A. optparseexample.py

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

import optparse

parser=optparse.OptionParser()
parser.add_option('--firstarg', '-s', dest='first', help="print stuff")
parser.add_option('--secondarg', dest='second')
parser.add_option('--flag1', dest='flag', action='store_true', default=True, hel
p='make flag true')
parser.add_option('--flag2', dest='flag', action='store_false', help='make flag
false')


options, args=parser.parse_args()
print options, args
print "keyword example cp", options.first, options.second #run as if we were running
the cp command
#print "positional example cp", args[0], args[1]
if options.flag:
        print "more stuff"
```

XXII. In class example - sort a structure by distance between consecutive atoms

   A. use this as an example of how to do top down design
      1. Top-down design
         a) Express a general problem in terms of smaller problems
            (1) Attack each smaller problem in the same way
            (2) Find closest atoms
               (a) findclosest.py

```
getinput
parse pdb
find distance pairs
sort pairs by distance
output closest pairs
def finddistance(atomlist):
    #loop through atomlist and find the distance of neighbors
    l=[]
    for n in range(len(atomlist)-1):
            dx=atomlist[n]['x']-atomlist[n+1]['x']
            dy=atomlist[n]['y']-atomlist[n+1]['y']
            dz=atomlist[n]['z']-atomlist[n+1]['z']
            d=math.sqrt(dx*dx+dy*dy+dz*dz)
            l.append([n,n+1,d])
    return l

def compare(x,y):
    return int((x[2]-y[2])*1000)
```

```
#get the imput
inputpdb=sys.argv[1]

#parse pdb
atomlist=pdbtools.readpdb(inputpdb)

#find the distance pairs
distlist=finddistance(atomlist)

#sort the pairs
distlist.sort(compare)

#output the closest pairs
print distlist[0]
```

B. alternatively could use
    1. dist = numpy.linalg.norm(a-b)

XXIII. exception handling

A. Look at the example of entering from std in

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

numbers=0
avg=0

while True:
  inp=raw_input('Please give me a number or the word "Done": ')
  if inp=="Done":
          break
  else:
          x=float(inp)
          avg+=x
          numbers+=1
avg=avg/numbers
print avg
print "Done!"
```

B. What we need is something that will test to see if a type conversion will work

C. We need to “try” one way, and if it works do one thing, otherwise, do something else
    1. try, except

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

numbers=0
avg=0

while True:
        inp=raw_input('Please give me a number or the word "Done": ')
        try:
                x=float(inp)
                avg+=x
                numbers+=1
        except ValueError:
                if inp=="Done":
                        break
                else:
                        print "Incorrect value entered. Please enter a
number or 'Done'"
#                       continue
avg=avg/numbers
print avg
print "Done!"
```

XXIV. Types of errors
A. logic error, syntax error, runtime error
B. logic error
1. faulty algorithm causes program to produce incorrect results
2. program still runs
3. ex dividing by n instead of n-1 in sig calculations
C. syntax error
1. violates one of Pythons grammatical rules
2. prevents running
3. No :, ask class for other examples
D. runtime error
1. the result of an illegal operation
2. execution error that occurs while program is running
3. example – divide by zero, convert string of characters to an int
E. Catch runtime and syntax errors with exceptions:
1. Types of exceptions
a) http://docs.python.org/library/exceptions.html
b) examples:
(1) ZeroDivisionError - 1/0
(2) ValueError - float(“stagg”)
(3) IndexError - l[1]
(4) KeyError - d[1]
(5) NameError - print x #when x doesn’t exist
(6) SyntaxError - for:
2. for while example:
a) try:
(1) avg=avg/numbers
b) except ZeroDivisionError:
c)   blah
d) except ZeroDivisionError, e:
e)   print e
f)   blah
(a) e is argument
g) else:
(1) executes if try completes without an exception
h)

XXV. Object Oriented Programming
A. (OOP)
B. What is an object

1. Object oriented programming (OOP): a class of programming techniques based on the concept of an "object" which is a data structure encapsulated with a set of routines which operate on the data.
2. an object combines both data and operations
   a) objects know stuff and do stuff
   b) assistant professor object
      (1) know stuff
         (a) funding agencies
         (b) tenure requirements
      (2) do stuff
         (a) teach class
         (b) give talk
         (c) write proposal
         (d) do experiment
   c) math object
   d) everything is an object in Python

C. Using classes tends to be of more use to people who work in *strategic* mode (doing long-term development) than those who work in *tactical* mode (short one off projects).

D. emphasizes code reuse

E. Class, instance, instantiation
1. class – defines the properties of an object
   a) we define an object by using the class syntax

```
class C1:
        def setdata(self,value):
                self.data=value
        def display(self):
            print self.data
```

2. instance – a particular object of some class

```
>>> a=C1
>>> type(a)
<type 'classobj'>
>>> a=C1()
>>> type(a)
<type 'instance'>
>>> a=C1 #
>>> dir(a)
['__doc__', '__module__', 'setdata']
>>> a.setdata(1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unbound method setdata() must be called with C1
instance as first argument (got int instance instead)
```

F. inheritance

```
class C2(C1):
        def display(self):
                print ('Current value is: %s' % self.data)
```

1. base classes and subclasses
   a) base class = superclass = parent class
   b) subclass = child class

G. constructor – a method that creates an instance of a class

```
def __init__(self, args):
```

   1. augmenting methods - a type of polymorphism
      a) class Manager(Employee):
         (1) def giveRaise(self, percent, bonus=0.1):
            (a) Employee.giveRaise(self,percent+bonus)

H. encapsulation - group data with methods designed to operate on that data so that redundancy is minimized. Makes updating more straightforward.

I. getX, getY are accessors
   1. methods that allow access to info

J. mutators – change the state of an obj

K. oop conventions
   1. capitalize FirstLetterOfWords when defining classes
   2. lowercase firstLetterOfFirstWord but capitalize first letter of all subsequent words

L. OOP is often used with graphics

M. GUI, interactive programming, widgets

N. import graphics
   1. win=graphics.GraphWin()
   2. p=graphics.Point(50,60)
   3. p2=graphics.Point(120,100)
      a) instance of the Point class
      b) two instances of the same class have same attributes
      c) create a new instance with a call to a constructor

O. do bounce as exercise

```
import graphics as g

steps=10000
stepsize=1

#create a window that is 400 x 400 pixels
win=g.GraphWin("bounce",400,400)
width=win.width
height=win.height

#create a circle
p=g.Point(200,200)
c=g.Circle(p,10)
c.draw(win)

#define velocity in x and y
vx=2.0
vy=0.5

#do a loop where the circle moves according to the velocity
for n in range(steps):
        dx=vx*stepsize
        dy=vy*stepsize
        c.move(dx,dy)

        #make circle bounce if it "hits" a wall
        p=c.getCenter()
        if p.getX() > width or p.getX() < 0:
                vx=vx*-1
        if p.getY() > height or p.getY() < 0:
                vy=vy*-1
```

P. demonstrate our own classes with employees.py
    1. Operator overloading
        a) __add__
        b) __mult__

Q. Inheritance continued.
    1. parent methods can be
        a) inherited, overwritten, or modfied/extended
    2. Restaurant example

```
class Employee:
        def __init__ (self,name,idnum,salary=0):
                self.name=name
                self.salary=salary
                self.idnum=idnum

        def work(self):
                print self.name, "does stuff"


        def giveRaise(self,percent):
                self.salary=self.salary+ (self.salary*percent/100)

        def __str__(self):
                return "Employee: %s, ID: %d, Salary: %f" %
(self.name,self.idnum,self.salary)

        def __add__(self,other):
                return (self.salary + other.salary)

        def __mul__(self,other):
                return (self.salary * other.salary)

class Chef(Employee):
        def __init__(self,name,idnum):
                Employee.__init__(self,name,idnum,salary=50000)

        def work(self):
                print self.name, "makes food"

class Manager(Employee):
        def __init__(self,name,idnum):
                Employee.__init__(self,name,idnum,salary=75000)

        def giveRaise(self,percent,bonus=0.1):
                Employee.giveRaise(self,percent+bonus)
```

3.

XXVI. Polymorphism

A. gives an object different behavior based on how it is constructed

```
class Animal:
   def __init__(self, name):    # Constructor of the class
      self.name = name
   def talk(self):              # Abstract method, defined by convention only
      raise NotImplementedError("Subclass must implement abstract method")

class Cat(Animal):
   def talk(self):
      return 'Meow!'

class Dog(Animal):
   def talk(self):
      return 'Woof! Woof!'

animals = [Cat('Missy'),
           Dog('Lassie')]

for animal in animals:
   print animal.name + ': ' + animal.talk()

# prints the following:
#
# Missy: Meow!
# Lassie: Woof! Woof!
```

XXVII. GUI development

A. GUI development typically requires an application programming interface (API) that provides lower level access to computer function
B. typical API for python GUI programming is tkinter
   1. from Tkinter import Tk
C. GUI APIs consist of a series of widgets with specific functions.
   1. GUI widgets are graphical elements, which are used to build the human-machine-interface of a program
D. another common widget is Label
   1. from tkinter import Tk, Label
   2. Label displays non-interactive text and images
      a) almost all widgets have the following options
         (1) text
         (2) image
         (3) width - in pixels
         (4) height - in pixels
         (5) relief - border style options are: FLAT, GROOVE, RAISED, RIDGE, SUNKEN (those attributes must be imported
            (a) i.e. from Tkinter import GROOVE
         (6) borderwidth - in pixels
         (7) foreground - color string
         (8) background - color string
         (9) font - tuple of strings?

(10) padx, pady - pixel padding added to widget

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

from Tkinter import Tk, Label

root=Tk()

hello=Label(master=root, text='Here is some text. It could be a gif too.', height=500)
hello.pack()
root.mainloop()
```

E. Image widget
   1. import PhotoImage

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

from Tkinter import Tk, Label, PhotoImage, LEFT, RIGHT, BOTTOM,GROOVE,
SUNKEN,RAISED

root=Tk()
root.title("Logo displayer")

message=Label(master=root, text='Go Seminoles', relief=RAISED,
background='black', foreground='white', font=('Helvetica', 16))
message.pack(side=BOTTOM)

logoimage=PhotoImage(file='fsu-seal-full-color.gif')
logo=Label(master=root, image=logoimage, relief=GROOVE)
logo.pack(fill='both')
root.mainloop()
```

F. Grid method for placing widgets

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

from Tkinter import Tk, Label, RAISED

root=Tk()

#make keypad labels as list of lists
labels=[['7','8','9'],['4','5','6'],['1','2','3'],['0','.','=']]

for r,row in enumerate(labels):
                for c,element in enumerate(row):
                        print element, r,c
                        label=Label(master=root,relief=RAISED, padx=10,
text=element,font=('Helvetica',24))
                        #element is the same as label[r][c]
                        label.grid(row=r,column=c)
root.mainloop()
```

G. Event handling
   1. The button widget

2. executes a defined function when clicked

XXVIII. graphics.py from Zelle

A. in interactive mode

```
import graphics
win=graphics.GraphWin()
win.close()
```

B. more complicatied

```
win=GraphWin(title="mybox",width=400,height=400)
p=Point(50,60)
p.getX()
p.getY()
p.draw(win)
move
another point
etc
c=Circle(p,30)
c.setFill('red')
c.setOutline('red')
dir (c)
plus=range(100)
minus=range(100)
minus.reverse()
combo=plus+minus
demo other shapes
demo clone method
```

C. bounce

```
import graphics as g

steps=10000
stepsize=1

#create a window that is 400 x 400 pixels
win=g.GraphWin("bounce",400,400)
width=win.width
height=win.height

#create a circle
p=g.Point(200,200)
c=g.Circle(p,10)
c.draw(win)

#define velocity in x and y
vx=2.0
vy=0.5

#do a loop where the circle moves according to the velocity
for n in range(steps):
                    dx=vx*stepsize
                    dy=vy*stepsize
                    c.move(dx,dy)
```

```
                                #make circle bounce if it "hits" a wall
                                p=c.getCenter()
                                if p.getX() > width or p.getX() < 0:
                                        vx=vx*-1
                                if p.getY() > height or p.getY() < 0:
                                        vy=vy*-1
```

D. potatohead

E. demonstrate importance of getting mouse clicks

```
import graphics as g

#use while true to keep window from closing
while True:
                                #create graphics window
                                win=g.GraphWin("potatohead",300,300)

                                #use getMouse function to define a point from a mouse
click
                                p=win.getMouse()

                                #draw a circle centered on p with radius 30
                                c1=g.Circle(p,30)
                                c1.draw(win)

                                #draw another circle, etc
                                p=win.getMouse()
                                c2=g.Circle(p,30)
                                c2.draw(win)

                                p=win.getMouse()
                                c3=g.Circle(p,10)
                                c3.draw(win)

                                #draw a polygon from three mouse clicks
                                p1=win.getMouse()
                                p2=win.getMouse()
                                p3=win.getMouse()
                                poly=g.Polygon(p1,p2,p3)
                                poly.draw(win)
```

F.

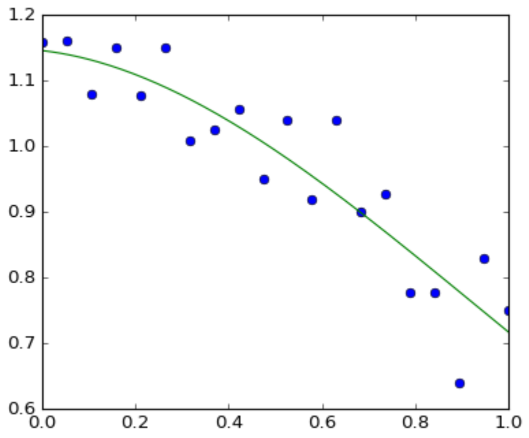
XXIX. More Numpy/Scipy/Matplotlib

A. NumPy also contains polynomials in different bases:

1. For example, $3x^2 + 2x - 1$:

```
>>> p = np.poly1d([3, 2, -1])
>>> p(0)
-1
>>> p.roots
```

```
array([-1.        ,  0.33333333])
>>> p.order
2
>>>
>>> x = np.linspace(0, 1, 20)
>>> y = np.cos(x) + 0.3*np.random.rand(20)
>>> p = np.poly1d(np.polyfit(x, y, 3))

>>> t = np.linspace(0, 1, 200)
>>> plt.plot(x, y, 'o', t, p(t), '-')
[<matplotlib.lines.Line2D object at ...>, <matplotlib.lines.Line2D object at ...>]
```



2. NumPy also has a more sophisticated polynomial interface.
   $3x^2 + 2x - 1$:

```
>>>
>>> p = np.polynomial.Polynomial([-1, 2, 3]) # coefs in different order!
>>> p(0)
-1.0
>>> p.roots()
array([-1.        ,  0.33333333])
>>> p.degree()  # In general polynomials do not always expose 'order'
2
```

3. Matplotlib tools.

```
import numpy as np
import matplotlib.pyplot as plt

X = np.linspace(-np.pi, np.pi, 256, endpoint=True)
C, S = np.cos(X), np.sin(X)

plt.plot(X, C)
plt.plot(X, S)

plt.show()
```

   a) defaults

```
import numpy as np
import matplotlib.pyplot as plt

# Create a figure of size 8x6 inches, 80 dots per inch
plt.figure(figsize=(8, 6), dpi=80)

# Create a new subplot from a grid of 1x1
plt.subplot(1, 1, 1)

X = np.linspace(-np.pi, np.pi, 256, endpoint=True)
C, S = np.cos(X), np.sin(X)

# Plot cosine with a blue continuous line of width 1 (pixels)
plt.plot(X, C, color="blue", linewidth=1.0, linestyle="-")

# Plot sine with a green continuous line of width 1 (pixels)
plt.plot(X, S, color="green", linewidth=1.0, linestyle="-")

# Set x limits
plt.xlim(-4.0, 4.0)

# Set x ticks
plt.xticks(np.linspace(-4, 4, 9, endpoint=True))

# Set y limits
plt.ylim(-1.0, 1.0)

# Set y ticks
plt.yticks(np.linspace(-1, 1, 5, endpoint=True))

# Save figure using 72 dots per inch
# plt.savefig("exercice_2.png", dpi=72)

# Show result on screen
plt.show()
```

b) changing colors and line widths

```
plt.figure(figsize=(10, 6), dpi=80)
plt.plot(X, C, color="blue", linewidth=2.5, linestyle="-")
plt.plot(X, S, color="red",  linewidth=2.5, linestyle="-")
```

c) changing limits

```
plt.plot(X, C, color="blue", linewidth=2.5, linestyle="-")
plt.plot(X, S, color="red",  linewidth=2.5, linestyle="-")

plt.xlim(X.min() * 1.1, X.max() * 1.1)
plt.ylim(C.min() * 1.1, C.max() * 1.1)
```

d) ticks

```
plt.plot(X, C, color="blue", linewidth=2.5, linestyle="-")
plt.plot(X, S, color="red",  linewidth=2.5, linestyle="-")
plt.xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi])
plt.yticks([-1, 0, +1])
```

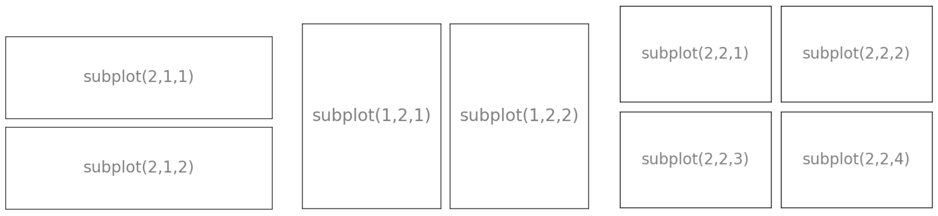
e) tick labels

```
plt.plot(X, C, color="blue", linewidth=2.5, linestyle="-")
plt.plot(X, S, color="red",  linewidth=2.5, linestyle="-")

plt.xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi], ['-pi','-pi/2','0', 'pi/2','pi'])
plt.yticks([-1, 0, +1])
```

f) axis labels

```
plt.xlabel('angle')
```

g) With subplot you can arrange plots in a regular grid. You need to specify the number of rows and columns and the number of the plot. Note that the gridspec command is a more powerful alternative.



XXX. Scipy least squares

```
import numpy as np
from matplotlib import pyplot as plt
from scipy.optimize import leastsq

def gen_data(t,a,b,c, noise=0, n_outliers=0, random_state=0):
                                y=a+b*np.exp(t*c)
                                rnd=np.random.RandomState(random_state)
                                error=noise*rnd.randn(t.size)
                                outliers=rnd.randint(0,t.size,n_outliers)
                                error[outliers]*=10
                                return y+error

def fun(x,t,y):
                                return x[0]+x[1]*np.exp(x[2]*t)-y


a=0.5
b=2.0
c=-1
t_min=0
t_max=10
n_points=15

t_train=np.linspace(t_min,t_max,n_points)
y_train=gen_data(t_train,a,b,c,noise=0.1)

x0 = np.array([1.0, 1.0, 0.0])
res_lsq = leastsq(fun, x0, args=(t_train, y_train))
#res_soft_l1 = leastsq(fun, x0, loss='soft_l1', f_scale=0.1, args=(t_train, y_train))
```

```
#res_log = leastsq(fun, x0, loss='cauchy', f_scale=0.1, args=(t_train, y_train))

print res_lsq
t_test = np.linspace(t_min, t_max, n_points * 10)
y_true = gen_data(t_test, a, b, c)
y_lsq = gen_data(t_test, *res_lsq[0])
#y_soft_l1 = gen_data(t_test, *res_soft_l1.x)
#y_log = gen_data(t_test, *res_log.x)

plt.plot(t_train, y_train, 'o')
plt.plot(t_test, y_true, 'k', linewidth=2, label='true')
plt.plot(t_test, y_lsq, label='linear loss')
#plt.plot(t_test, y_soft_l1, label='soft_l1 loss')
#plt.plot(t_test, y_log, label='cauchy loss')
plt.xlabel("t")
plt.ylabel("y")
plt.legend()

#plt.plot(t_train,y_train,'o')
plt.show()
```

A. virtualenv bch5884 python
   1. activate
   2. pip upgrade
   3. pip search
   4. pip install
   5. biopython