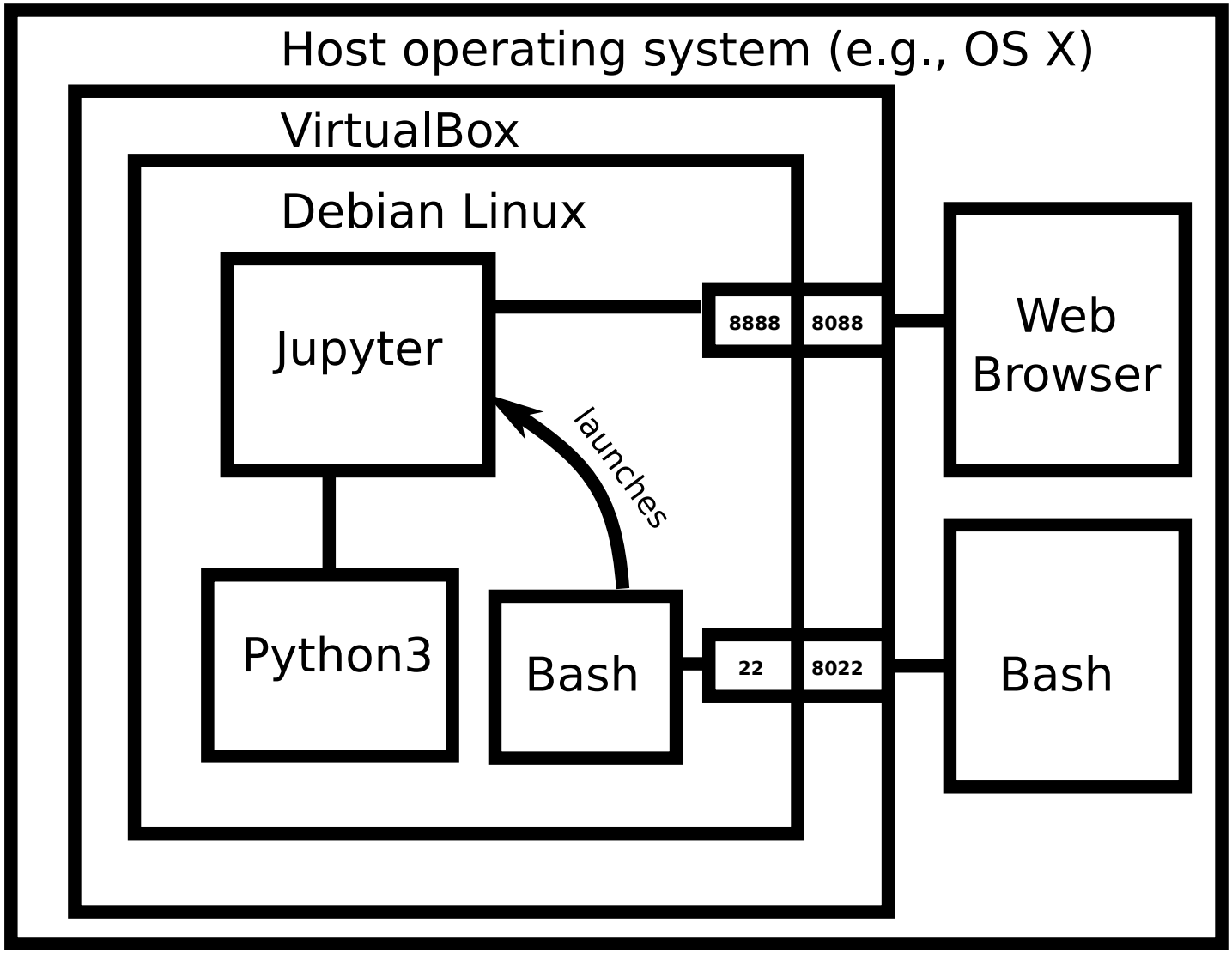
Practical Bioinformatics

Mark Voorhies

5/14/2019

Course platform: VirtualBox



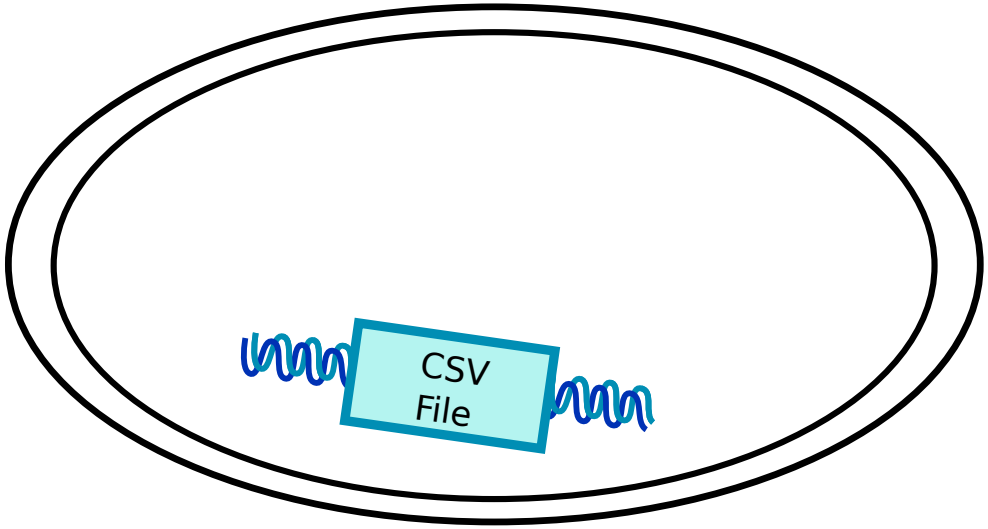
Starting the virtual machine

1. Start virtual box
2. Boot the VM guest
3. Open a bash terminal on the host
4. Log into the guest and start Jupyter:

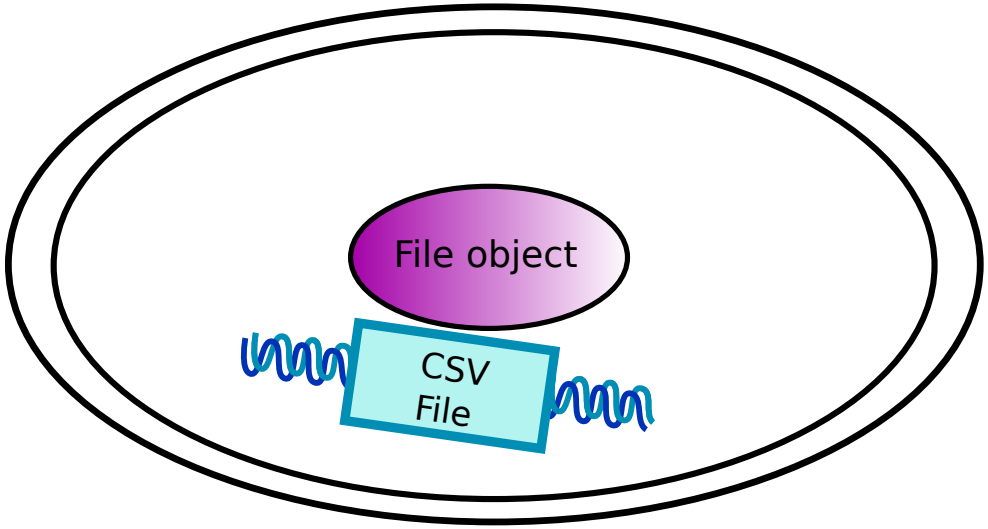
```
ssh-add ~/.ssh/VM_rsa
ssh -p 8022 explorer@localhost
jupyter notebook
```

5. In a host web browser, go to https://localhost:8088/

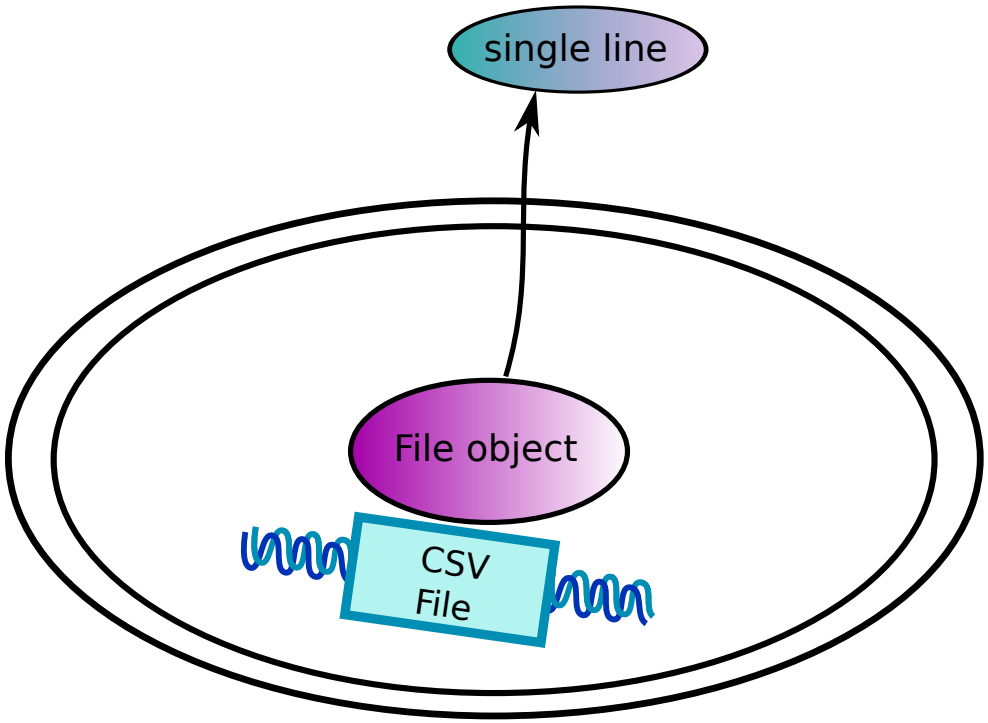
supp2data.csv



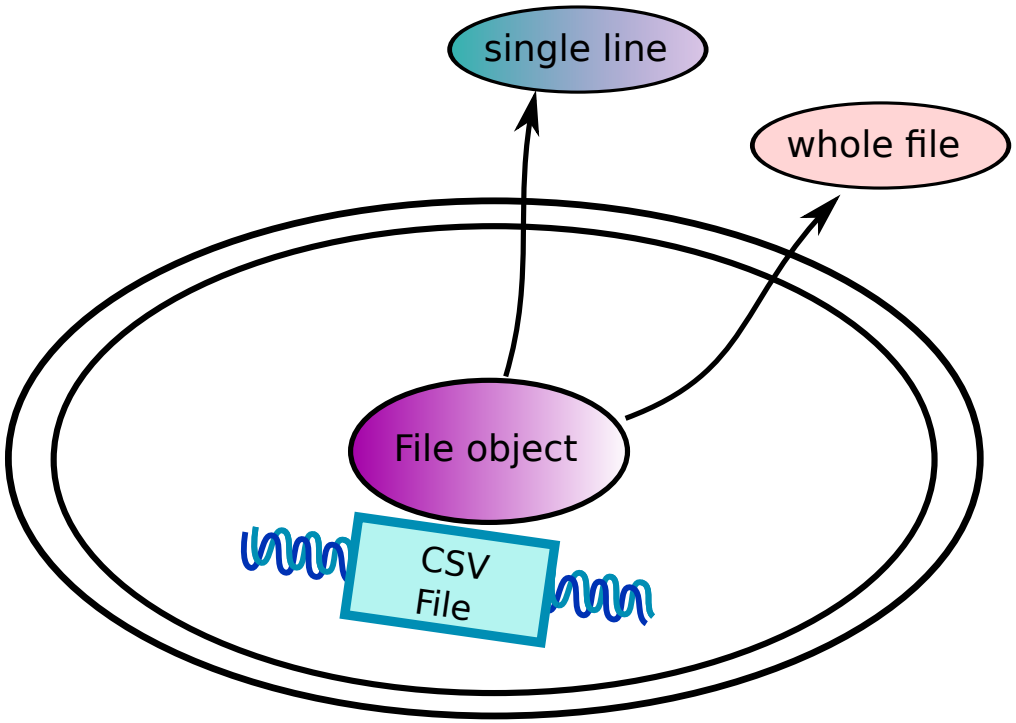
open("supp2data.csv")



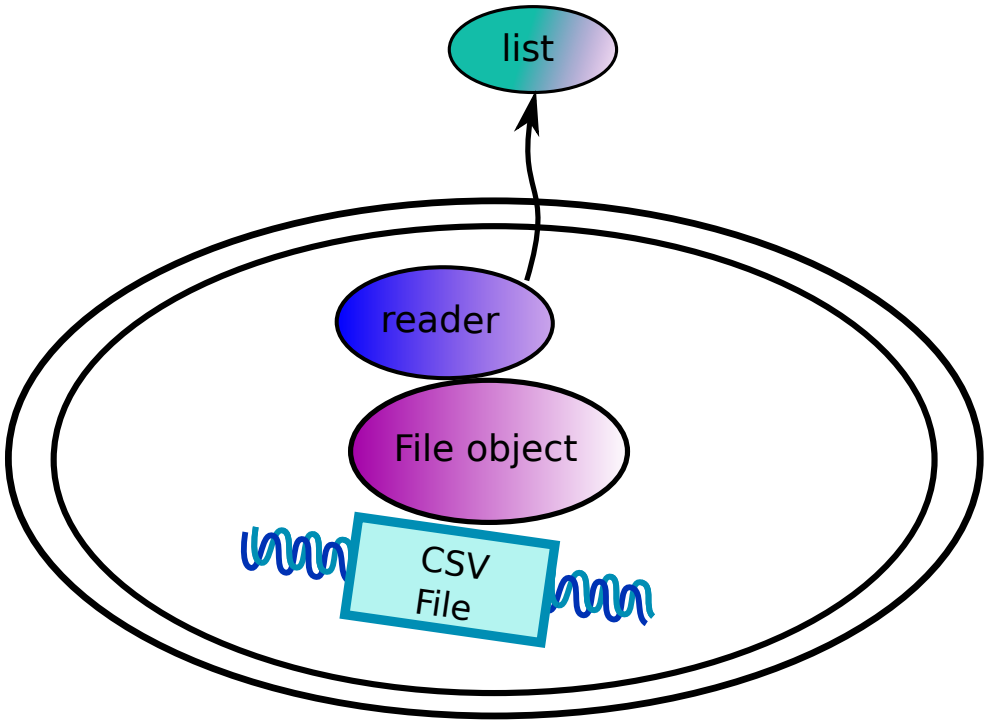
open("supp2data.csv").next()



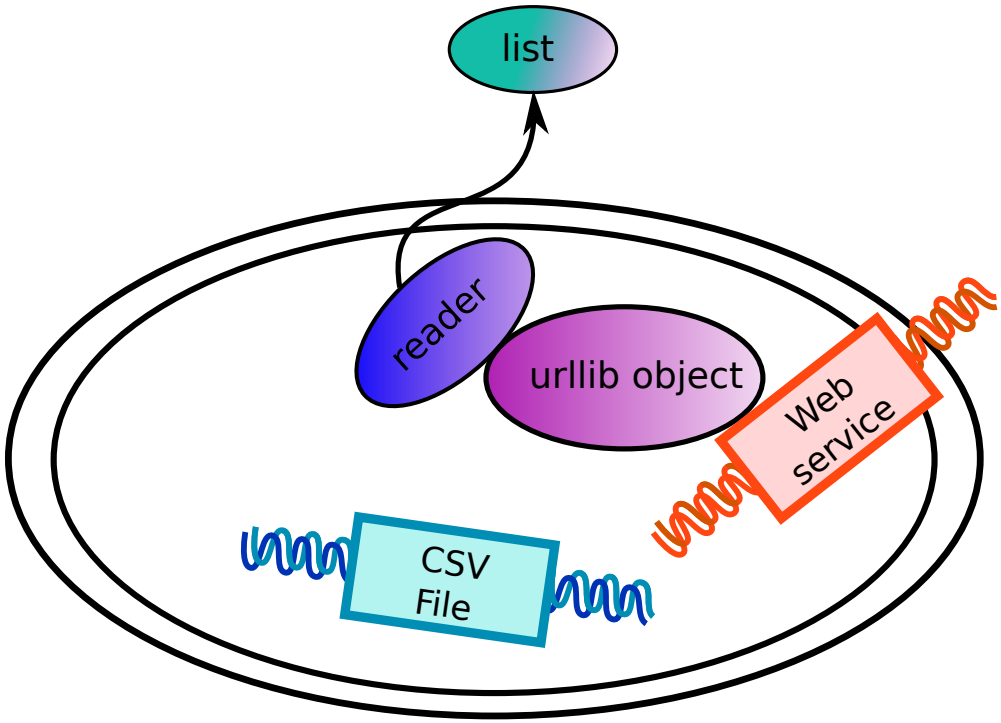
open("supp2data.csv").read()



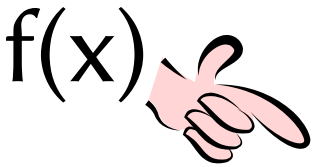
csv.reader(open("supp2data.csv")).next()



csv.reader(urlopen("http://example.com/csv")).next()

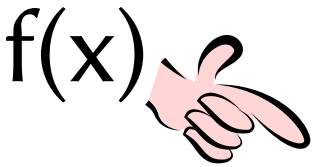


Anatomy of a Programming Language

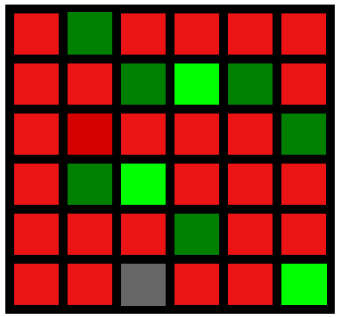


functions

Anatomy of a Programming Language

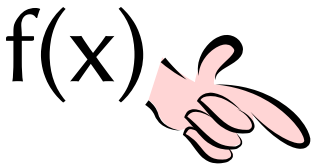


functions

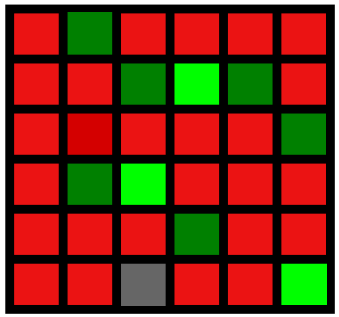


data structures

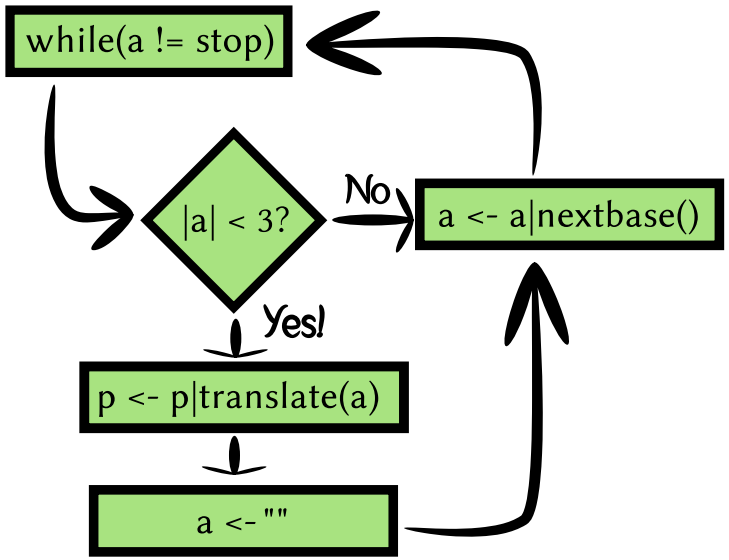
Anatomy of a Programming Language



functions

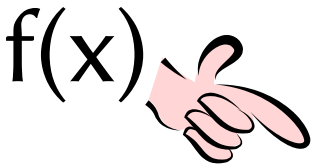


data structures

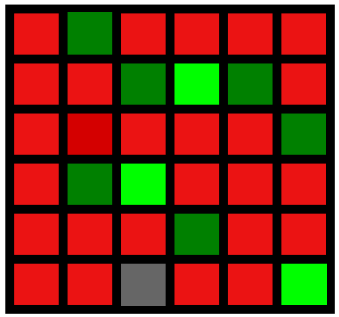


control statements

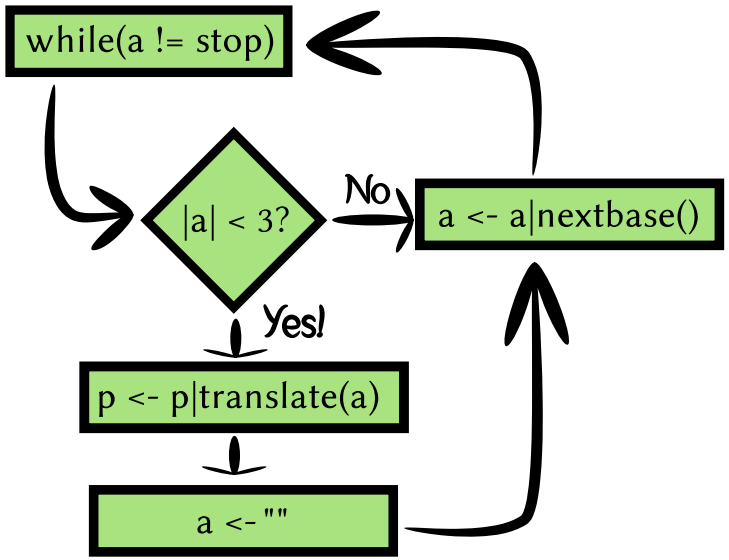
Anatomy of a Programming Language



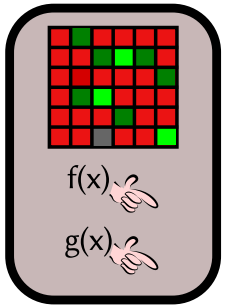
functions



data structures



control statements



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
5/3
# 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
```

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"
```

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
```

Making decisions

```
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
```

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", "while", and "if"

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.

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".

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".

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, ...)

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):

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