Python

Tools for Reproducible Research

Karl Broman

Biostatistics & Medical Informatics, UW–Madison

kbroman.org
github.com/kbroman
@kwbroman

Course web: kbroman.org/Tools4RR
Why python?

- Manipulating data files
- Simulations using others’ programs
Why python?

- Manipulating data files
- Simulations using others’ programs
- Web-related stuff
Why python?

- Manipulating data files
- Simulations using others’ programs
- Web-related stuff
- Alternative to R for data analysis and graphics
- Jupyter notebooks
Python 2 vs Python 3

▶ Many people are using Python version 2.7
▶ Python 3 was introduced in 2008
  – A number of large changes
  – Some important Python programs haven’t been ported
  – Few people seem to be using it day-to-day
▶ Ideally, go straight for Python 3
  – But be aware of differences
Installing Python

- On Mac or Unix, Python 2 should be pre-installed
  ```
  python --version
  ```

- For Windows (or to be current, or to alternate between Python 2 and 3), install Anaconda
  Includes NumPy, SciPy, Pandas, iPython, Matplotlib, …
  
  [continuum.io/downloads](http://continuum.io/downloads)
Learning a new language

- Find a good book
- Have good example tasks/problems
- Play around
- Force yourself to use the new language
- Develop a script illustrating different language features
Into the thick of it

Learn Python through one example

markers.txt
families.txt ➔ data.gen
genotypes.txt
Input: markers.txt

<table>
<thead>
<tr>
<th>Marker</th>
</tr>
</thead>
<tbody>
<tr>
<td>D20S103</td>
</tr>
<tr>
<td>D20S482</td>
</tr>
<tr>
<td>D20S851</td>
</tr>
<tr>
<td>D20S604</td>
</tr>
<tr>
<td>D20S1143</td>
</tr>
<tr>
<td>D20S470</td>
</tr>
<tr>
<td>D20S477</td>
</tr>
<tr>
<td>D20S478</td>
</tr>
<tr>
<td>D20S481</td>
</tr>
<tr>
<td>D20S159</td>
</tr>
<tr>
<td>D20S480</td>
</tr>
<tr>
<td>D20S451</td>
</tr>
<tr>
<td>D20S171</td>
</tr>
<tr>
<td>D20S164</td>
</tr>
</tbody>
</table>
Input: families.txt

<table>
<thead>
<tr>
<th>Family</th>
<th>Individual</th>
<th>Father</th>
<th>Mother</th>
<th>Sex</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
Input: genotypes.txt

<table>
<thead>
<tr>
<th>Marker</th>
<th>1-1</th>
<th>1-2</th>
<th>1-3</th>
<th>1-4</th>
<th>1-5</th>
<th>2-1</th>
<th>2-2</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>D20S103</td>
<td>100/98</td>
<td>98/98</td>
<td>98/98</td>
<td>98/98</td>
<td>100/100</td>
<td>100/96</td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>D20S1143</td>
<td>176/172</td>
<td>170/176</td>
<td>170/176</td>
<td>170/176</td>
<td>172/172</td>
<td>172/172</td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>D20S159</td>
<td>350/358</td>
<td>354/350</td>
<td>354/350</td>
<td>354/358</td>
<td>366/354</td>
<td>350/366</td>
<td>354/354</td>
<td>...</td>
</tr>
<tr>
<td>D20S164</td>
<td>191/207</td>
<td>207/207</td>
<td>207/207</td>
<td>207/207</td>
<td>207/207</td>
<td>207/207</td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>D20S171</td>
<td>141/135</td>
<td>137/141</td>
<td>137/141</td>
<td>137/135</td>
<td>137/141</td>
<td>137/143</td>
<td>135/139</td>
<td>...</td>
</tr>
<tr>
<td>D20S451</td>
<td>324/308</td>
<td>320/316</td>
<td>324/316</td>
<td>324/316</td>
<td>320/324</td>
<td>324/320</td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>D20S470</td>
<td>306/302</td>
<td>305/302</td>
<td>306/306</td>
<td>306/306</td>
<td>302/302</td>
<td>302/302</td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>D20S477</td>
<td>256/252</td>
<td>252/252</td>
<td>252/252</td>
<td>252/252</td>
<td>252/252</td>
<td>252/252</td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>D20S478</td>
<td>267/263</td>
<td>263/263</td>
<td>263/263</td>
<td>263/263</td>
<td>267/263</td>
<td>271/265</td>
<td>247/265</td>
<td>...</td>
</tr>
<tr>
<td>D20S480</td>
<td>304/284</td>
<td>304/284</td>
<td>304/284</td>
<td>304/284</td>
<td>304/284</td>
<td>296/300</td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>D20S481</td>
<td>229/237</td>
<td>237/237</td>
<td>237/237</td>
<td>237/237</td>
<td>237/237</td>
<td>237/237</td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>D20S482</td>
<td>155/159</td>
<td>167/159</td>
<td>159/155</td>
<td>167/159</td>
<td>167/147</td>
<td>155/159</td>
<td>155/155</td>
<td>...</td>
</tr>
<tr>
<td>D20S604</td>
<td>151/147</td>
<td>147/147</td>
<td>147/135</td>
<td>143/151</td>
<td>143/147</td>
<td>147/143</td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>D20S851</td>
<td>132/140</td>
<td>140/140</td>
<td>140/140</td>
<td>140/140</td>
<td>140/140</td>
<td>140/140</td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
Output: data.gen

5
14
D20S103
D20S482
...
D20S171
D20S164
1
5
1 0 0 1
0 0 155 159 132 140 151 147 176 172 306 302 256 252 0 0 ...
2 0 0 0
100 98 159 167 148 144 0 0 180 176 302 306 260 252 267 ...
3 2 1 1
98 98 159 159 132 144 147 135 176 180 302 306 252 252 ...
4 2 1 0
98 98 155 167 132 148 151 143 0 0 306 302 0 0 263 263 ...
5 2 1 0
98 98 159 167 132 148 151 143 172 180 302 302 256 252 ...
2
4
...
...
#!/usr/bin/env python
# Combine the data in "genotypes.txt", "markers.txt" and
# "families.txt" and convert them into a CRI-MAP .gen file
#
# This is the python 2 version

def read_markers (filename):
    "Read an ordered list of marker names from a file."
    with open(filename, 'r') as f:
        lines = f.readlines()
    return [line.strip() for line in lines]

class Person:
    "Person class, to contain the data on a subject."
    def __init__ (self,family, id, dad, mom, sex):
        self.family = family
        self.id = id
        self.dad = dad
        self.mom = mom
        self.sex = "0" if sex == "2" else sex # convert 1/2 -> 1/0
        self.famid = family + '-' + id
        self.gen = {}
if __name__ == '__main__':
    # file names
    gfile = "genotypes.txt"  # genotype data
    mfile = "markers.txt"    # list of markers, in order
    ffile = "families.txt"   # family information
    ofile = "data.gen"      # output file

    # read the data
    markers = read_markers(mfile)
    people = read_families(ffile)
    read_genotypes(gfile, people)

    # write the data
    write_genfile(ofile, people, markers)
Write functions & modules not scripts

- Write a set of reusable functions
- Your code will be easier to read
- You may actually reuse the code, this way
Try it out

$ convert2.py
$ diff data.gen data_save.gen

$ python       # (or ipython)

>>> import convert2

>>> help(convert2)
>>> help(convert2.read_markers)

>>> markers = convert2.read_markers("markers.txt")
>>> markers[0]
>>> len(markers)
>>> markers[-1]
>>> markers[0:2]
>>> markers[0:-1]
>>> markers[0:-1]
>>> markers[5:]
>>> markers[:5]
>>> markers[0:7:2]

>>> quit()
def read_markers (filename):
    "Read an ordered list of marker names from a file."
    with open(filename, 'r') as f:
        lines = f.readlines()
    return [line.strip() for line in lines]
class Person:
    "Person class, to contain the data on a subject."
    def __init__ (self, family, id, dad, mom, sex):
        self.family = family
        self.id = id
        self.dad = dad
        self.mom = mom
        self.sex = "0" if sex == "2" else sex # convert 1/2 -> 1/0
        self.famid = family + '-' + id
        self.gen = {}

Example use:

ind = Person("1", "3", "1", "2", "2")
def read_families (filename):
    "Read family info and return a hash of people."
    with open(filename, 'r') as file:
        file.readline() # header row
        people = {}
        for line in file:
            vals = line.strip().split()
            person = Person(vals[0],vals[1],vals[2],vals[3],vals[4])
            people[person.famid] = person
        return people
def parse_genotype (string):
    "Clean up string -> genotype"
    string = string.replace(' ', '')
    string = "0/0" if string == "" else string
    return string.replace('/', ' ')

def read_genotypes (filename, people):
    "Read genotype data, fill in genotypes within people hash"
    with open(filename, 'r') as file:
        header = file.readline().strip().split()
        header = header[1:] # omit the first field, "Marker"

        for line in file[1:] # omit the first field, "Marker"
            marker = line[:9].replace(' ', '')
            line = line[9:]
            for i in range(len(header)):
                person = header[i]
                start = i*7
                people[person].gen[marker] = \
                parse_genotype(line[start:(start+7)])
Some helper functions

```python
def get_families (people):
    "Return a vector of distinct families"
    return set([people[key].family for key in people])

def get_family_members (people, family):
    "Return a vector of famids for subjects within a family."
    return [key for key in people \
        if people[key].family == family]

def writeln (file, line, end="\n"):
    "Write a single line to a file."
    file.write(str(line) + end)
```
def write_genfile (filename, people, markers):
    "Write genotype data to a file, in CRI-MAP format."
    with open(filename, 'w') as file:
        families = sorted(get_families(people))
        writeln(file, len(families))
        writeln(file, len(markers))
        for marker in markers:
            writeln(file, marker)
        for family in families:
            writeln(file, family)
            members = sorted(get_family_members(people, family), \
                             key=lambda famid: int(people[famid].id))
            writeln(file, len(members))
            for famid in members:
                person = people[famid]
                writeln(file, "%s %s %s %s" % (person.id, \
                                               person.mom, person.dad, person.sex))
                for marker in markers:
                    writeln(file, person.gen[marker], " ")
                    writeln(file, "")
if __name__ == '__main__':
    # file names
    gfile = "genotypes.txt"  # genotype data
    mfile = "markers.txt"    # list of markers, in order
    ffile = "families.txt"   # family information
    ofile = "data.gen"       # output file

    # read the data
    markers = read_markers(mfile)
    people = read_families(ffile)
    read_genotypes(gfile, people)

    # write the data
    write_genfile(ofile, people, markers)
Basic types

► float
  x = 0.3

► int
  m = 5

► string
  s = "blah"

► bool
  x = True
  y = False

► None
  x = None

► complex
  x = 5+0j
Converting between types, and such

```python
n = 5
print(type(n))

s = str(n)
x = float(n)

"%s %s %s" % (n, s, x)
"%d %d %d" % (n, int(s), x)
"%.2f %.2f %.2f" % (n, float(s), x)

dir(s)
dir(x)

s = "blah"
print(len(s))
s[2:]  # s[2:] will print s[2:5], which is "lah"
s[:-1]   # s[:-1] will print s[0:-1], which is "bla"
for ch in s:
    print(ch)
```
Multi-element types

- **list**
  
  ```python
  x = [1, 2, 3, None, "blah"]
  y = [[1, 2], [3, 4, 5], 6]
  ```

- **dictionary**
  
  ```python
  h = {'x': 3, 'y': 5, 'name':"Karl"}
  ```

- **tuple**
  
  ```python
  x = (1, [2,3])
  ```

- **set**
  
  ```python
  S = set([5, 3, 5, 1, 2, 1])
  ```
matrices as lists of lists

\[
x = \begin{bmatrix}
[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]
\end{bmatrix}
\]

\[
x[1][3]
\]
for loops

```python
vec = range(4)
for x in vec:
    print (x+1)**2

import math
for i in xrange(len(vec)):
    print math.log( vec[i] + 1 )

h = {'x':3, 'y':4, 'z':2}
for k in h:
    print k, h[k]

for k in sorted(h.keys()):
    print k, h[k]

for k,v in h.iteritems():
    print k, v

for v in h.itervalues():
    print v
```
vec = range(10)
[v**2 for v in vec if v > 5]

h = {'x':3, 'y':4, 'zz':2}
[h[k]**2 for k in h]
[h[k]**2 for k in h if len(k) == 1]
[[k, v**3] for k,v in h.items()]
dict( [[k, v**3] for k,v in h.items()] )

x = [k+1 for k in range(6)]
y = [True, False, True, False, False, False]
[x[i] for i in range(len(x)) if y[i]]
More with strings

```python
x = "bread and jam"
y = x.split(" ")
z = " ".join(y)

dir(x)
help(x.index)

x.endswith("jam")
x.startswith("bre")
x.count("a")
x.find("and")
x.find("jelly")
x.index("and")
x.index("jelly")

x.replace("jam", "jelly")

x.capitalize()
x.title()
x.upper()
x.upper().lower()
```
import re

x = "Bread and Jam"
re.findall(r'[A-Z]', x)
re.split(r'[A-Z]', x)
re.sub(r'[A-Z]', '', x)

ph = "555-12-3456"
re.findall(r'-', ph)
re.findall(r'\d+', ph)
re.split(r'\D', ph)
re.sub(r'\D', '', ph)
# This is nosetest_convert2.py
#
# At command line, type "nose tests nosetest_convert2.py"

from nose.tools import assert_equal
from convert2 import *

def test_parse_genotype():
    assert_equal(parse_genotype(" "), "0 0")
    assert_equal(parse_genotype("100/98 "), "100 98")
    assert_equal(parse_genotype("90/96 "), "90 96")
    assert_equal(parse_genotype("90/ 96 "), "90 96")
    assert_equal(parse_genotype(" 3 / 8 "), "3 8")
#!/usr/bin/env python
# Test one of the functions in convert2.py
#
# on the command line, type "test_convert2.py"

import unittest
from convert2 import *

class check_parse_genotype(unittest.TestCase):
    def test_parse_genotype(self):
        self.assertEqual(parse_genotype(" "), "0 0")
        self.assertEqual(parse_genotype("100/98 "), "100 98")
        self.assertEqual(parse_genotype("90/96 "), "90 96")
        self.assertEqual(parse_genotype("90/ 96 "), "90 96")
        self.assertEqual(parse_genotype(" 3 / 8 "), "3 8")

if __name__ == '__main__':
    unittest.main()
Summary

- Learn a scripting language, like Python
  - Not just for manipulating data files, but worth the effort just for that.
- Force yourself to use it