Intermediate/Advanced Python
Michael Weinstein
(Day 2)
Topics

› Review of basic data structures
› Accessing and working with objects in python
› Numpy
  – How python actually stores data in memory
  – Why numpy can help
  – Dot product example
  – Extending our SAMLine object from yesterday
  – Making and analyzing our quality score matrix
› Pandas
› Matplotlib
  – Making a histogram
› Scipy
A true array (from C++ or similar languages)

- A true array is stored sequentially in a fixed space in the computer’s memory

A 10 integer array

Possibly other stuff

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
</table>

Maybe passwords or instructions?
Seriously, that other stuff isn’t a joke

HOW THE HEARTBLEED BUG WORKS:

SERVER, ARE YOU STILL THERE?
IF SO, REPLY "POTATO" (6 LETTERS).

User Meg wants these 6 letters: POTATO. User
eda wants pages about "irl games". Unlocking
secure records with master key 5130985733434.
Seriously, that other stuff isn’t a joke
Seriously, that other stuff isn’t a joke
A python list uses pointers for flexibility

- A pointer is a value that points to a specific location in the computer’s memory

A 10 integer list
A python list uses pointers for flexibility

› This method of storing data increases flexibility

Deleting a number from a 10 integer list
A python list uses pointers for flexibility

› This method of storing data increases flexibility

Deleting a number from a 10 integer list
A python list uses pointers for flexibility

› This method of storing data increases flexibility

Deleting a number from a 10 integer list
A python list uses pointers for flexibility

- This method of storing data increases flexibility

Adding a number to a 10 integer list
A python list uses pointers for flexibility

› This method of storing data increases flexibility

Adding a number to a 10 integer list
A python list uses pointers for flexibility

- This method of storing data increases flexibility

Replacing a number in a 10 integer list with a string
A python list uses pointers for flexibility

› This method of storing data increases flexibility

Replacing a number in a 10 integer list with a string
A python list uses pointers for flexibility

› This method of storing data increases flexibility

Replacing a number in a 10 integer list with a string

I am to misbehave
A python list uses pointers for flexibility

- This method of storing data increases flexibility

Replacing a number in a 10 integer list with a string

I am to misbehave
Numpy gives us access to true arrays

› Your numpy objects will have limited flexibility
› Data in your numpy objects will have to be homogeneous
› There will be some overhead in terms of computing time to turn your python object into a numpy object

› Why on Earth would we want to do this?
A true array (from C++ or similar languages)

<table>
<thead>
<tr>
<th>Possibly other stuff</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>Maybe passwords or instructions?</th>
</tr>
</thead>
</table>

Send me the first value in the array at 0x08C4
A true array (from C++ or similar languages)

Possibly other stuff

0 1 2 3 4 5 6 7 8 9

Maybe passwords or instructions?
A true array (from C++ or similar languages)

Prefetching (big speed-up)
Let’s prove this (and learn some useful functions)
And the commands...

```python
start = datetime.datetime.now()
runManual(firstMatrixPython, secondMatrixPython, iterations)
print("Python and manual run in %s" % (datetime.datetime.now() - start))

start = datetime.datetime.now()
runManual(firstMatrixNumpy, secondMatrixNumpy, iterations)
print("Numpy and manual run in %s" % (datetime.datetime.now() - start))

start = datetime.datetime.now()
runNumpy(firstMatrixNumpy, secondMatrixNumpy, iterations)
print("Numpy automatic run in %s" % (datetime.datetime.now() - start))
```
Results

- Python is already pretty fast for this operation
- Going back and forth between numpy and standard python makes it much less fast
- Running as much of this in numpy as possible gave about a 10x speed increase
Topics

› Review of basic data structures
› Accessing and working with objects in python
› Numpy
   – How python actually stores data in memory
   – Why numpy can help
   – Dot product example
   – Extending our SAMLine object from yesterday
   – Making and analyzing our quality score matrix
› Pandas
› Matplotlib
   – Making a histogram
› Scipy
Quality Scores

<table>
<thead>
<tr>
<th>Score</th>
<th>Sanger</th>
<th>Phred+33, raw reads typically (0, 40)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Score</td>
<td>X - Solexa</td>
<td>Solexa+64, raw reads typically (-5, 40)</td>
</tr>
<tr>
<td>Score</td>
<td>I - Illumina 1.3+</td>
<td>Phred+64, raw reads typically (0, 40)</td>
</tr>
<tr>
<td>Score</td>
<td>J - Illumina 1.5+</td>
<td>Phred+64, raw reads typically (3, 40)</td>
</tr>
<tr>
<td>Score</td>
<td>L - Illumina 1.8+</td>
<td>Phred+33, raw reads typically (0, 41)</td>
</tr>
</tbody>
</table>
Time to code a little more

› Create a new file in your working folder called qualityStringHandler.py
› Make a copy of your working samReader.py called samReader2.py
How to convert ASCII values

C:\Users\mweinstein\Documents\pythonClass2>python
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:57:36) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> ord('!')
33
>>> ord('@')
64
>>> ord('M')
77
>>> chr(33)
'!
>>> chr(64)
'@
>>> chr(77)
'M'
Our new class for quality strings

```python
class QualityString(object):
    def __init__(self, rawQualityString, base = 33):
        assert type(rawQualityString) == str, "Quality string must be a string."
        self.qualityString = rawQualityString.strip()
        self.qualityArray = self.calculateQualityArray(base)

    def calculateQualityArray(self, base = 33):
        collection = []
        for character in self.qualityString:
            collection.append(ord(character) - base)
        return collection

    def __str__(self):
        return self.qualityString
```
Modify SAMLine to handle the new data
Goal:

› Generate a histogram to look at the average quality score distribution in the first 50 bases of any read of length 100 or more from the sample data

› What we know
  – We can already break down this data very well
  – We will have to filter on read length (easy, since we already store it)
  – We will have to iterate over lines and build up a matrix
  – We will have to take an average across each row (read) in the matrix
  – We will have to generate a histogram of these values

› New file: histogramMaker.py
How to get the list of lists

```python
>>> import samReader2
>>> data = samReader2.readSAMFileLines("sampleData.sam")
Read 500172 lines
>>> collection = []
>>> for line in data:
...    if line.readLength >= 100:
...        collection.append(line.quality.qualityArray[0:50])
...    if len(collection) >= 10:
...        break
...

>>> collection

[[40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 37, 40, 40, 40, 40, 40, 40, 39, 40, 40, 40, 40, 40, 39, 40, 40, 40, 40, 40, 39, 40, 40, 40, 40, 40, 39, 40, 40, 40, 40, 40, 39, 40, 40, 40, 40, 40, 39, 40, 40, 40, 40, 40, 39, 40, 40, 40, 40, 40, 39, 40, 40, 40, 40, 40, 39, 40, 40, 40, 40, 40, 39, 40, 40, 40, 40, 40, 39, 40, 40, 40, 40, 40, 39, 40, 40, 40, 40, 40, 39, 40, 40, 40, 40, 40, 39, 40, 40, 40, 40, 40, 39, 40, 40, 40, 40, 40, 39, 40, 40, 40, 40, 40, 39, 40, 40, 40, 40, 40, 39, 40, 40, 40, 40, 40, 39, 40, 40, 40, 40, 40, 39, 40, 40, 40, 40, 40, 39, 40, 40, 40, 40, 40, 39, 40, 40, 40, 40, 40, 39, 40, 40, 40, 40, 40, 39, 40, 40, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38]
How to turn a list of lists into a numpy matrix

```python
>>> import numpy
>>> dataMatrix = numpy.matrix(collection)
>>> dataMatrix
matrix([[40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 37, 40, 40, 40,
        40, 40, 37, 40, 39, 40, 40, 40, 39, 40, 35, 40, 40, 37, 40,
        40, 40, 35, 39, 40, 40, 40, 39, 39, 39, 39, 39, 38, 39, 40, 39],
        [40, 40, 40, 39, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 39,
        40, 40, 40, 40, 40, 39, 40, 38, 40, 38, 39, 38, 40, 40, 40, 38,
        39, 39, 40, 40, 40, 40, 40, 39, 39, 39, 39, 39, 37, 40, 33, 40, 36],
        [40, 39, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40,
        40, 39, 40, 40, 40, 38, 40, 40, 40, 40, 40, 40, 40, 40, 33, 38, 38,
        38, 38, 38, 38, 38, 38, 39, 38, 38, 39, 38, 38, 38, 38, 35, 36, 38],
        [40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40,
        39, 40, 38, 40, 39, 39, 40, 40, 35, 40, 48, 40, 36, 39, 40, 40, 40,
        40, 40, 37, 40, 39, 33, 40, 40, 38, 40, 38, 40, 35, 40, 40, 40, 36, 39],
        [38, 38, 38, 38, 38, 38, 38, 40, 40, 35, 40, 40, 40, 40, 40, 40,
        40, 38, 40, 38, 40, 38, 40, 40, 39, 40, 39, 40, 39, 38, 36, 39, 37,
        38, 38, 38, 38, 38, 38, 39, 38, 39, 38, 39, 38, 39, 36, 31, 36, 36],
        [40, 39, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40,
        40, 36, 40, 36, 40, 36, 36, 40, 36, 36, 40, 36, 40, 36, 40, 36, 40,
        39, 36, 39, 39, 38, 38, 39, 39, 38, 39, 39, 39, 39, 36, 39, 36, 38],
        [40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40,
        40, 40, 40, 40, 40, 39, 40, 40, 40, 40, 38, 40, 40, 40, 39, 39, 39,
        39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39],
        [39, 36, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40,
        40, 39, 40, 40, 40, 36, 40, 40, 40, 40, 38, 40, 40, 40, 40, 40, 40,
        40, 40, 40, 40, 40, 36, 40, 37, 35, 32, 35, 35, 37, 33, 37, 36],
        [40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40,
        40, 39, 40, 40, 40, 36, 40, 40, 40, 40, 38, 40, 40, 40, 40, 40, 40,
        40, 40, 40, 40, 40, 36, 40, 37, 35, 32, 35, 35, 37, 33, 37, 36],
        [40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40,
        36, 40, 40, 40, 40, 40, 40, 40, 40, 40, 38, 40, 40, 40, 40, 40, 40,
        40, 39, 40, 40, 40, 36, 40, 40, 40, 40, 38, 40, 40, 40, 40, 40, 40,
        40, 40, 40, 40, 40, 36, 40, 37, 35, 32, 35, 35, 37, 33, 37, 36],
        [40, 40, 39, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40,
        40, 36, 40, 36, 40, 36, 36, 40, 36, 36, 40, 36, 40, 36, 40, 36, 40,
        39, 36, 39, 39, 38, 38, 39, 39, 38, 39, 39, 39, 39, 36, 39, 36, 38],
        [40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40,
        40, 40, 40, 40, 40, 39, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40,
        40, 40, 40, 40, 40, 36, 40, 37, 35, 32, 35, 35, 37, 33, 37, 36],
        [39, 36, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40,
        40, 37, 40, 39, 35, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40,
        39, 38, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40,
        31, 36, 36, 35, 35, 35, 36, 34, 33, 33, 34, 33, 33, 32, 33]])])
```
And how to generate a matrix of just means

```python
>>> dataMeans = dataMatrix.mean(axis = 1)
>>> dataMeans
matrix([[ 39.4 ],
        [ 39.36],
        [ 38.88],
        [ 39.26],
        [ 37.74],
        [ 38.42],
        [ 38.7 ],
        [ 38.68],
        [ 38.2 ],
        [ 36.58]])
>>> dataMeans = dataMeans.transpose()
>>> dataMeans
matrix([[ 39.4 , 39.36, 38.88, 39.26, 37.74, 38.42, 38.7 , 38.68,
        38.2 , 36.58]])
```
A function to get a matrix of qualities

```python
readLengthMinimum = 100
analysisLength = 50

def getQualityMeanMatrix(samFile, readLengthMinimum, analysisLength, analysisPositionStart = 0, dataLimit = False):
    assert readLengthMinimum >= analysisLength + analysisPositionStart, "Analysis length must be shorter than or equal to read length minimum."
    import samReader2
    import numpy
    samLines = samReader2.readSAMFileLines(samFile)
    qualityList = []
    for line in samLines:
        if len(line.quality.qualityArray) >= readLengthMinimum:
            qualityList.append(line.quality.qualityArray[analysisPositionStart:analysisLength + analysisPositionStart])
        if dataLimit:
            if len(qualityList) >= dataLimit:
                break
    qualityMatrix = numpy.matrix(qualityList)
    meanMatrix = qualityMatrix.mean(axis = 1)
    return meanMatrix.transpose()
```
A function to make a histogram

```python
def makeHistogram(meanMatrix):
    import matplotlib.pyplot as plt
    plt.hist(meanMatrix, bins = 41)
    plt.title("Average quality score in the first 50 bases")
    plt.xlabel("Phred score")
    plt.ylabel("Frequency")
    #plt.show()
    plt.savefig("qualityByRead.png")

qualityMeanMatrix = getQualityMeanMatrix("sampleData.sam", 100, 50)
print("Mean matrix shape:")
print(qualityMeanMatrix.shape)
makeHistogram(qualityMeanMatrix)
```
Result?

C:\Users\mweinstein\Documents\pythonClass2>python histogramMaker.py
Read 500172 lines
Mean matrix shape:
(500000, 1)