CME 193: Introduction to Scientific Python
Lecture 2: Data Structures

Austin Benson

Dan Frank

Institute for Computational and Mathematical Engineering (ICME)

April 9, 2013
Administrivia

Lists and tuples

Strings

Dictionaries

Functions
Homework

- Homework 1 is due now.
- Homework 2 has been posted.
- Need to get NumPy and SciPy working by Lecture 3
Today: Data types

After knowing a few basic built-in Python data structures, you can write powerful code.

Data structures covered today:

- Lists and tuples
- Strings
- Dictionaries
Administrivia

Lists and tuples

Strings

Dictionaries

Functions
Lists

Lists store a sequence of data which supports indexing.

We saw lists in the polynomial evaluation example from last lecture.
Lists

\[
l = [2, 4, 6, 8]
\]

\[
\text{print } l[3] \ # \text{ prints } '8' \\
\text{print } l[-1] \ # \text{ prints } '8'
\]

\[
\text{print } l[4] \ # \text{ error} \\
l.\text{append}(10) \\
\text{print } l[4] \ # \text{ prints } '10'
\]

Negative indexing!
Lists

We can add lists together:

\[
\begin{align*}
l_1 &= [2, 4, 6, 8, 10] \\
l_2 &= [3, 5] \\
l_3 &= l_1 + l_2 \\
\text{# addition: } l_3 \text{ is now } [2, 4, 6, 8, 10, 3, 5] \\
l_3.pop() \# \text{ removes 5 from } l_3
\end{align*}
\]
Lists

We can also manipulate slices of a list:

\[
l = [2, 4, 6, 8]
\]

\[
\text{print } l[0:2] \# \text{ prints } [2, 4]
\]

\[
l[1:3] = [7, 7] \# l \text{ is now } [2, 7, 7, 8]
\]

\[
\text{print } l[2:] \# \text{ prints } [7, 8]
\]

\[
l *= 2
\]

\[
\text{print } l[3:6] \# \text{ prints } [8, 2, 7]
\]
Python provides many helpful built-in functions:

```
l = [2, 4, 6, 8, 10, 12]

len(l) # 6: number of elements
max(l) # 12
min(l) # 2

# more advanced if you are interested:
filter(lambda(x): x % 4, l)
```
Lists do not have to be homogeneous and they can be nested:

```python
l = [2, [3, 5, 6], 'orange', 6, 'blue']
print(l[2]) # prints 'orange'
```
The `for` and `in` operators can be used to iterate over and find elements in a list:

```python
l = [2, 4, 'orange', 6, 'blue']

for elmt in l:
    print elmt

if 'blue' in l and 'red' not in l:
    print 'hi' # this will be printed
```
enumerate is convenient for keeping track of the index

```
squares = [0, 1, 4, 9, 16, 25]

for i, val in enumerate(squares):
    print i, val

# cleaner and more concise than:
i = 0
for val in squares:
    print i, val
    i = i + 1
```
List comprehensions form new lists by manipulating old ones

```python
vals = [1, 2, 3, 5, 7, 9, 10]

double_vals = [2 * v for v in vals]
```
Incorporate an if statement:

```python
vals = [1, 2, 3, 5, 7, 9, 10]
# Only include doubles for values divisible by 5
double_vals5 = [2 * v for v in vals if v % 5 == 0]
```
List comprehensions

Nested comprehension:

```python
x_pts = [-1, 0, 2]
y_pts = [2, 4]

xy_pts = [[x, y] for x in x_pts for y in y_pts]
# [[-1, 2], [-1, 4], [0, 2], [0, 4], [2, 2], [2, 4]]
```
Tuples

Tuples are similar to lists but they are immutable (cannot be modified)

You can use tuples to enforce structure in your code
Tuples

\[
p1 = (\text{'start'}, 1.2, -3.0, 17.222) \\
p2 = (\text{'end'}, -7.3, 0.0, -0.0001) \\
\]

\[
p1[3] = 17.2 \# \text{ error!} \\
\]

\[
\text{print } p2[2] \# \text{ prints '0.0'}
\]

# unpacking
\[
type1, x1, y1, z1 = p1 \\
type2, x2, y2, z2 = p2 \\
\]

\[
\text{print } x1 - x2 \# \text{ prints '8.5'}
\]
Administrivia

Lists and tuples

Strings

Dictionaries

Functions
Strings

Properties of strings are similar to those of lists (indexing, slicing)

There are many built-in string commands that make string manipulations easy (we are already saw arithmetic on strings)
Strings

str = 'hello, world!'

print str[1]  # prints 'e'
print str[-1]  # prints '!'  
print str[7:12]  # prints 'world'

str += '!!!1!'
Strings

Parsing a vector:

vec = ’[12.4, 3, 4, 7.22]’

# strip away the brackets
vec = vec.lstrip(’[’)
vec = vec.rstrip(’]’)

# form an array by splitting on comma
nums = vec.split(’,’)

# go from string to floating point
nums = [float(n) for n in nums]
Parsing a vector (one-liner):

```
vec = '12.4, 3, 4, 7.22'
nums = [float(n) for n in vec.strip('[[]]').split(',')]```
Strings

A few more useful functions:

```python
str = 'Hello, World!'

len(str) # 13
str = str.lower() # 'hello, world!'  
str = ' '.join(['Hello', 'World', '!'])
# Hello World !
```
Administrivia

Lists and tuples

Strings

Dictionaries

Functions
Dictionaries are maps from a set of keys to a set of values.

Dictionaries are also called “associative arrays”
Dictionaries

\[ K = \{ \text{keys} \} , \ V = \{ \text{values} \} . \ D : K \rightarrow V : \]

\[ k \xrightarrow{D} v_k \in V \]

In Python, you form \( D \) by a series of insertions of tuples \((k, v_k) \in K \times V\). It is “fast” to compute \( D(k) \).
Example:

```python
import math

p = (1.2, -40.0, 2*math.pi)

point = {}  # form an empty dictionary

point['x'] = p[0]
point['y'] = p[1]
point['z'] = p[2]
point['r'] = math.sqrt(sum([v ** 2 for v in p]))
point['theta'] = math.acos(point['z'] / point['r'])
point['phi'] = math.atan(point['y'] / point['x'])
```
Cleaner initialization:

```python
import math

p = (1.2, -40.0, 2*math.pi)

# Create dictionary with keys
point = {'x': p[0], 'y': p[1], 'z': p[2],
         'r': math.sqrt(sum([v ** 2 for v in p]))}  
point['theta'] = math.acos(point['z'] / point['r'])
point['phi'] = math.atan(point['y'] / point['x'])
```
Dictionaries

Accessing, removing, and overwriting keys:

```python
# access
magnitude = point['r']
x = point['rho']  # error!

# overwrite
point['r'] = 5.13
point['r'] = 6.23

# remove key-value pair
del point['theta']
```
Dictionaries

in and for:

```python
# print all keys
for key in point:
    print key

# check if a key is there
if 'theta' not in point:
    print 'missing theta!'
```
Administrivia

Lists and tuples

Strings

Dictionaries

Functions
Functions:

- Last time: functions are used to organize programs into coherent pieces
- Today:
  - Specifically learn Python functions
  - What is a lambda?
Discrete Fourier Transform (DFT)

def is used to define a function

```python
import cmath # complex math library

def dftk(x, k):
    c = -1j * 2 * cmath.pi * k / len(x)
    Xk = 0
    for n, xn in enumerate(x):
        Xk += xn * cmath.exp(c * n)
    return Xk

X3 = dftk([1, 2, 0.1, -1.1, 5], 3)
```

$x$ and $k$ are the arguments to the function
We can provide default argument values:

```python
import cmath

def dftk(x, k=0):
    c = -1j * 2 * cmath.pi * k / len(x)
    Xk = 0
    for n, xn in enumerate(x):
        Xk += xn * cmath.exp(c * n)
    return Xk

X0 = dftk([1, 2, 0.1, -1.1, 5])
X3 = dftk([1, 2, 0.1, -1.1, 5], 3)
```
import cmath

def dftk(x, k=0, all=False):
    if all:
        return [dftk(x, k) for k in range(len(x))]
    c = -1j * 2 * cmath.pi * k / len(x)
    Xk = 0
    for n, xn in enumerate(x):
        Xk += xn * cmath.exp(c * n)
    return Xk

X = dftk([1, 2, 0.1, -1.1, 5], all=True)
In Python, we can pass functions as objects:

```python
def square(x):
    return x ** 2

def cube(x):
    return x ** 3

def operate(f, y):
    return f(y)

print operate(square, 4) # prints '16'
print operate(cube, 4) # prints '64'
```
Lambdas

Sometimes it is more convenient to not declare functions:

```python
def operate(f, y):
    return f(y)

print operate(lambda(x): x ** 2, 4) # prints '16'
print operate(lambda(x): x ** 3, 4) # prints '64'

square_plus_cube = lambda(x): x ** 2 + x ** 3
print operate(square_plus_cube, 4) # prints '80'
```

These in-line function definitions are called lambdas or anonymous functions.
id_dept_pairs = [(8283, 'Aero/Astro'),
                 (3456, 'CS'),
                 (7888, 'Math')]

# Sort by id number
print sorted(id_dept_pairs,
            key=lambda pair: pair[0])
# [(3456, 'CS'), (7888, 'Math'),
#   (8283, 'Aero/Astro')]

# Sort by department alphabetically
print sorted(id_dept_pairs,
            key=lambda pair: pair[1])
# [(8283, 'Aero/Astro'), (3456, 'CS'),
#   (7888, 'Math')]
The import statement

We have seen the import statement in a number of examples. The import statement is used to load a library.

```python
import math
# code is in lambda2.py
import lambda2

print math.pi

print lambda2.operate(lambda2.square_plus_cube, 4)
```
We can import a library with a different name using as.

```python
import math
import lambda2 as l2

print math.pi

print l2.operate(l2.square_plus_cube, 4)
```
The import statement

We can import a library directly with no namespace.

```python
import math
from lambda2 import *

print math.pi

print operate(square_plus_cube, 4)
```
Assignment 2 is posted on the course web site (due Thursday 4/11).

Next time:
1. File I/O
2. Classes and object-oriented Python
3. Intro to NumPy