CME 193: Introduction to Scientific Python

Lecture 4: Strings, Filo I/O, Numpy

Sven Schmit

`stanford.edu/~schmit/cme193`

Contents

- Strings
- File I/O
- Numpy
- Exercises

Strings

Let's quickly go over *strings*.

- Strings hold a sequence of characters.
- Strings are *immutable*
- We can slice strings just like lists and tuples
- Between quotes or triple quotes

Everything can be turned into a string!

We can turn anything in Python into a string using `str`.

This includes dictionaries, lists, tuples, etc.

String formatting

- Special characters: \n, \t, \b, etc
- Add variables: %s, %f, %e, %g, %d, or use `format`

```
fl = 0.23
wo = 'Hello'
inte = 12

print "s: {} \t f: {:0.1f} \n i: {}".format(wo, fl, inte)
# s: Hello     f: 0.2
#  i: 12
```

Split

To split a string, for example, into seperate words, we can use `split()`

```
text = 'Hello, world!\n How are you?'
text.split()
# ['Hello,', 'world!', 'How', 'are', 'you?']
```

Split

What if we have a comma seperated file with numbers seperated by commas?

```
numbers = '1, 3, 2, 5'
numbers.split()
# ['1,', '3,', '2,', '5']

numbers.split(', ')
# ['1', '3', '2', '5']

[int(i) for i in numbers.split(', ')]
# [1, 3, 2, 5]
```

Use the optional argument in `split()` to use a custom seperator.

What to use for a tab seperated file?

Split

What if we have a comma seperated file with numbers seperated by commas?

```
numbers = '1, 3, 2, 5'
numbers.split()
# ['1,', '3,', '2,', '5']

numbers.split(', ')
# ['1', '3', '2', '5']

[int(i) for i in numbers.split(', ')]
# [1, 3, 2, 5]
```

Use the optional argument in `split()` to use a custom seperator.

What to use for a tab seperated file?

UPPER and lowercase

There are a bunch of useful string functions, such as `.lower()` and `.upper()` that turn your string in lower- and uppercase.

Note: To quickly find all functions for a string, we can use `dir`

```
text = 'hello'
dir(text)
```

join

Another handy function: `join`.

We can use join to create a string from a list.

```
words = ['hello', 'world']
' '.join(words)

''.join(words)
# 'helloworld'

' '.join(words)
# 'hello world'

', '.join(words)
# 'hello, world'
```

Contents

File I/O

How to read from and write to disk.

The file object

- Interaction with the file system is pretty straightforward in Python.
- Done using *file objects*
- We can instantiate a file object using `open` or `file`

Opening a file

```
f = open(filename, option)
```

- filename: path and filename
- option:

r	read file
w	write to file
a	append to file

We need to close a file after we are done: `f.close()`

with open() as f

Very useful way to open, read/write and close file:

```
with open('data/text_file.txt', 'r') as f:
    print f.read()
```

Reading files

read()	Read entire line (or first n characters, if supplied)
readline()	Reads a single line per call
readlines()	Returns a list with lines (splits at newline)

Another fast option to read a file

```
with open('f.txt', 'r') as f:
    for line in f:
        print line
```

Reading files

read()	Read entire line (or first n characters, if supplied)
readline()	Reads a single line per call
readlines()	Returns a list with lines (splits at newline)

Another fast option to read a file

```
with open('f.txt', 'r') as f:
    for line in f:
        print line
```

Writing to file

Use `write()` to write to a file

```
with open(filename, 'w') as f:
    f.write("Hello, %s!\n" % "John")
```

Contents

Numpy

- NumPy is the fundamental package for scientific computing with Python
- N-dimensional array object
- Linear algebra, Fourier transform, random number capabilities
- Building block for other packages (e.g. Scipy)
- Open source

import numpy as np

```
import numpy as np

array = np.array([[1, 2, 3], [4, 5, 6]])
floatarray = np.array([1, 2, 3], float)
```

Slicing works as usual.

More basics

```
np.arange(0, 1, 0.1)
# array([ 0. ,  0.2,  0.4,  0.6,  0.8])

np.linspace(0, 2*np.pi, 100)
# array([ 0.,  0.785,  1.570,  2.356,  3.141])

A = np.zeros((2,3))
# array([[ 0.,  0.,  0.],
#        [ 0.,  0.,  0.]])
# np.ones, np.diag
A.shape
# (2, 3)
```

More basics

```
np.random.random((2,3))
# array([[ 0.78084261,  0.64328818,  0.55380341],
#        [ 0.24611092,  0.37011213,  0.83313416]])

a = np.random.normal(loc=1.0, scale=2.0, size=(2,2))
# array([[ 2.87799514,  0.6284259 ],
#        [ 3.10683164,  2.05324587]])

np.savetxt("a_out.txt", a)
# save to file
b = np.loadtxt("a_out.txt")
# read from file
```

Array attributes

```
a = np.arange(10).reshape((2,5))

a.ndim    # 2 dimension
a.shape   # (2, 5) shape of array
a.size    # 10 # of elements
a.T       # transpose
a.dtype   # data type
```

Basic operations

Arithmetic operators on arrays apply *elementwise*

```
a = np.arange(4)
b = np.array([2, 3, 2, 4])

a * b  # array([ 0,  3,  4, 12])
b - a  # array([2, 2, 0, 1])

c = [2, 3, 4, 5]
a * c  # array([ 0,  3,  8, 15])
```

Also, we can use shorthand operators such as += and *=.

More operations

```
a = np.random.random((2,3))
# array([[ 0.23471,  0.69461,  0.99414 ],
#        [ 0.19549,  0.23025,  0.59429]])
a.sum()
a.sum(axis=0)  # column sum
a.cumsum()
a.cumsum(axis=1)  # cumulative row sum
a.min()
a.max(axis=0)
```

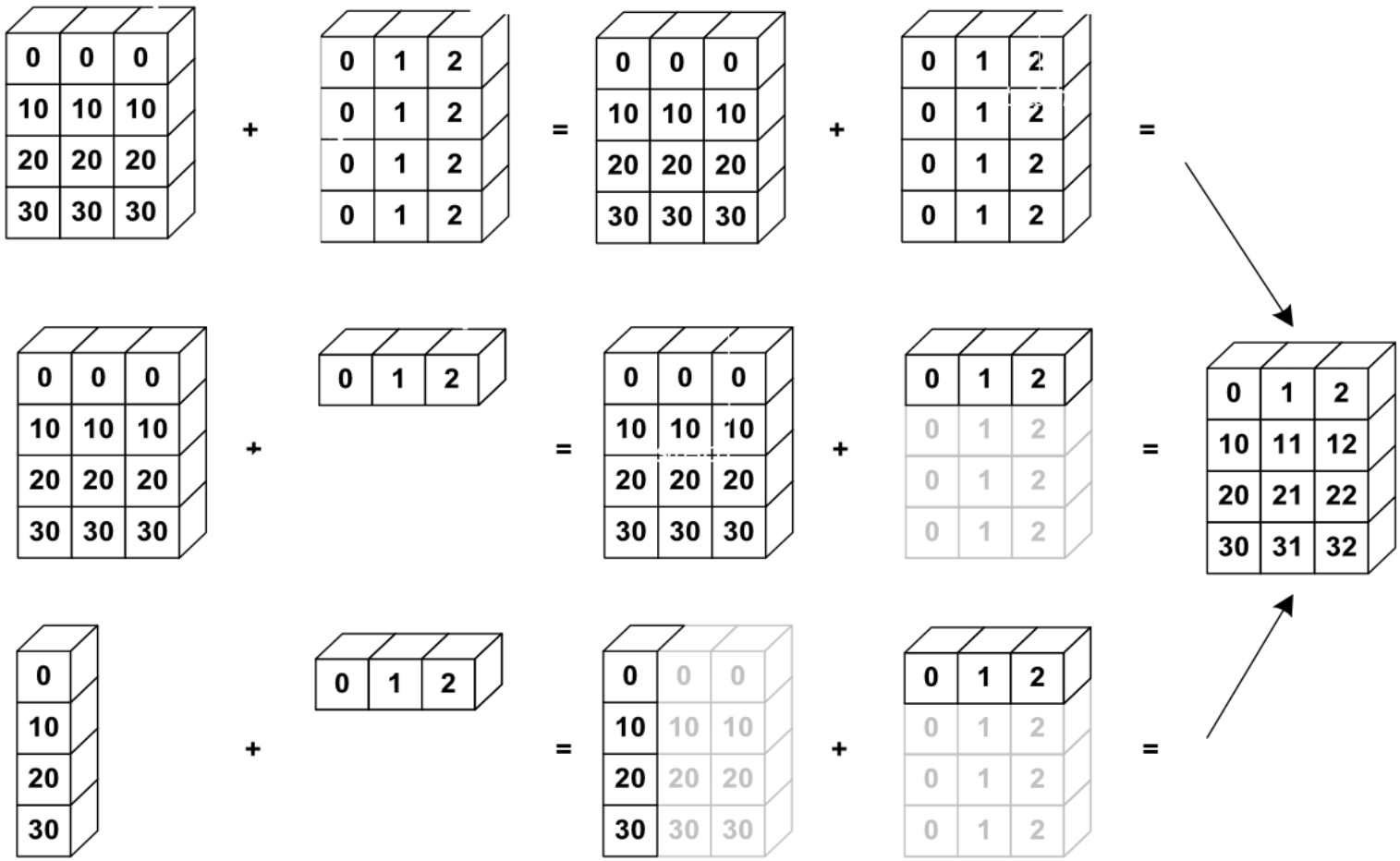
Many Numpy (and Scipy) functions are created to operate along axes.

Array broadcasting

When operating on two arrays, numpy compares shapes. Two dimensions are compatible when

1. They are of equal size
2. One of them is 1

Array broadcasting



Slicing arrays

```
a = np.random.random((4,5))

a[2, :]
# third row, all columns
a[1:3]
# 2nd, 3rd row, all columns
a[:, 2:4]
# all rows, columns 3 and 4
```

Iterating over arrays

- Iterating over multidimensional arrays is done with respect to the first axis: `for row in A`
- Looping over all elements: `for element in A.flat`

Reshaping

We can reshape an array using `reshape`. Total size must remain the same.

We can resize an array using `resize`, this will append zeros, or remove elements. Hence, size can change. Note the first dimension has 'priority'.

Try it!

Reshaping

We can reshape an array using `reshape`. Total size must remain the same.

We can resize an array using `resize`, this will append zeros, or remove elements. Hence, size can change. Note the first dimension has 'priority'.

Try it!

Reshaping

We can reshape an array using `reshape`. Total size must remain the same.

We can resize an array using `resize`, this will append zeros, or remove elements. Hence, size can change. Note the first dimension has 'priority'.

Try it!

Matrix operations

```
import numpy.linalg
```

`dot(A, B)`	Dot product
`eye(3)`	Identity matrix
`trace(A)`	Trace
`column_stack((A,B))`	Stack column wise
`row_stack((A,B,A))`	Stack row wise

Linear algebra

```
import numpy.linalg
```

`qr`	Computes the QR decomposition
`cholesky`	Computes the Cholesky decomposition
`inv(A)`	Inverse
`solve(A,b)`	Solves $Ax = b$ for A full rank
`lstsq(A,b)`	Solves $\arg\min_x \|Ax - b\|_2$
`eig(A)`	Eigenvalue decomposition
`eig(A)`	Eigenvalue decomposition for symmetric or hermitian
`eigvals(A)`	Computes eigenvalues.
`svd(A, full)`	Singular value decomposition
`pinv(A)`	Computes pseudo-inverse of A

Fourier transform

```
import numpy.fft
```

- `fft` 1-dimensional DFT
- `fft2` 2-dimensional DFT
- `fftn` N-dimensional DFT
- `ifft` 1-dimensional inverse DFT (etc.)
- `rfft` Real DFT (1-dim)
- `ifft` Imaginary DFT (1-dim)

Random sampling

```
import numpy.random
```

`rand(d0,d1,...,dn)`	Random values in a given shape
`randn(d0, d1, ...,dn)`	Random standard normal
`randint(lo, hi, size)`	Random integers [lo, hi)
`choice(a, size, repl, p)`	Sample from a
`shuffle(a)`	Permutation (in-place)
`permutation(a)`	Permutation (new array)

Distributions in random

```
import numpy.random
```

The list of distributions to sample from is quite long, and includes

- `beta`
- `binomial`
- `chisquare`
- `exponential`
- `dirichlet`
- `gamma`
- `laplace`
- `lognormal`
- `pareto`
- `poisson`
- `power`

Contents

- Strings
- File I/O
- Numpy
- Exercises

Exercises