Numerical Python

NumPy

Announcements

quiz: `quiz10` due on Tues 22/10

lab: `lab` is going to Sports Meet. No Lab

hw: `hw05` due Wed 23/10

exam: `exam02` coming...

Objectives

A. Understand `NumPy` arrays as a new container type.
B. Use `NumPy` arrays to store and operate on multidimensional data.

Dictreader

Question

We have this file **drinks.txt**

```
Item,Normal,Professor,Student
Tea,16,10,11
Coffee,18,12,13
Latte,22,15,16
Chocolate Milk,20,12,5
```

How do we read it?

Files/DictReader(...)

```
from csv import DictReader
DictReader(myfile, fieldnames=[...]
(optional), delimiter=',' (optional))
```

> *myfile = file* that you return with `open(...)`
> `fieldnames` = [...] Optional as the values in the first row of the *myfile* is used. If supplied, values in the first row will be treated as part of the data
> `delimiter` = ',' Optional. Default delimiter = ','

This reader does not give you what you see in a file.

Files/DictReader(...)

```
from csv import DictReader
myfile = open( 'drinks.csv' )
thisHasData = DictReader(myfile)

for banana in thisHasData:
    print(banana)

myfile.close()
```

Files/DictReader(...)

```
from csv import DictReader
myfile = open( 'drinks.csv' )
thisHasData = DictReader(myfile)

for banana in thisHasData:
    print(banana)

myfile.close()
```

Ans: (Many OrderedDict)

```
OrderedDict([('Item', 'Tea'), ('Normal', '16'),
             ('Professor', '10'), ('Student', '11')])
OrderedDict([('Item', 'Coffee'), ('Normal', '18'),
             ('Professor', '12'), ('Student', '13')])
OrderedDict([('Item', 'Latte'), ('Normal', '22'),
             ('Professor', '15'), ('Student', '16')])
OrderedDict([('Item', 'Chocolate Milk'), ('Normal',
             ('Professor', '12'), ('Student', '5')])
```

dictionaries Recap

Question

```
d = { 'red':1, 'green':2, 'blue':3 }
for n in d:
    print( n )
```

What does this code print?

A The values of `d`.

B The keys of `d`.

C The key–value pairs of `d`.

Question

```
d = { 'red':1, 'green':2, 'blue':3 }
for n in d:
    print( n )
```

What does this code print?

A The values of `d`.

B The keys of `n`. ⋆⋆⋆

C The key–value pairs of `d`.

So how do you access value?

Question

```
d = { 'red':1, 'green':2, 'blue':3 }
for n in d:
    print( n )
```

What does this code print?

A The values of `d`.

B The keys of `n`. ⋆⋆⋆

C The key–value pairs of `d`.

So how do you access value?

```
d[ n ]
```

Question

```
d1st = { 'red':1, 'green':1 }
d2nd = { }
dK1 = list(d1st.keys())
for n in range(2):
    d2nd['C'] = n
    d2nd['E'] = n
    d1st[dK1[n]] = d2nd

print(d1st)
```

What does this code print?

A {'red': {'C': 1, 'E': 1}, 'green': {'C': 1, 'E': 1}}

B {'red': {'C': 0, 'E': 0}, 'green': {'C': 1, 'E': 1}}

C error

Question

```
d1st = { 'red':1, 'green':1 }
d2nd = { }
dK1 = list(d1st.keys())
for n in range(2):
    d2nd['C'] = n
    d2nd['E'] = n
    d1st[dK1[n]] = d2nd

print(d1st)
```

What does this code print?

A {'red': {'C': 1, 'E': 1}, 'green': {'C': 1, 'E': 1}} ★★★

B {'red': {'C': 0, 'E': 0}, 'green': {'C': 1, 'E': 1}}

C error

So how do you get B?

Question

```
d1st = { 'red':1, 'green':1 }
d2nd = { }
dK1 = list(d1st.keys())
for n in range(2):
    d2nd['C'] = n
    d2nd['E'] = n
    d1st[dK1[n]] = d2nd

print(d1st)
```

What does this code print?

A {'red': {'C': 1, 'E': 1}, 'green': {'C': 1, 'E': 1}} ★★★

B {'red': {'C': 0, 'E': 0}, 'green': {'C': 1, 'E': 1}}

C error

So how do you get B?

Move `d2nd = { }` into for loop

The problem

```
mydata = [ 4.5, 6.0, 1.2, 5.4 ]
from math import sin
sin(mydata)
```

The problem

```
mydata = [ 4.5, 6.0, 1.2, 5.4 ]
from math import sin
sin(mydata)
```

Error! Why doesn't this work?

`list` can contain any type!

Also operators don't do what we "want":

```
mydata * 2  # doesn't double values!
```

The problem

```
mydata = [ 4.5, 6.0, 1.2, 5.4 ]
from math import sin
sin(mydata)
```

Error! Why doesn't this work?

`list` can contain any type!

Also operators don't do what we "want":

```
mydata * 2  # doesn't double values!
```

ans:

```
[4.5, 6.0, 1.2, 5.4, 4.5, 6.0, 1.2, 5.4]
```

Numpy Arrays

numpy

```
import numpy
import numpy as np  # rename it, it's easier
```

numpy

```
import numpy
import numpy as np  # rename it, it's easier
```

`numpy` provides arrays and mathematical functions.

```
data = np.array( [ 4.5, 6.0, 1.2, 5.4 ] )
data * 2
```

numpy

```
import numpy
import numpy as np  # rename it, it's easier
```

`numpy` provides arrays and mathematical functions.

```
data = np.array( [ 4.5, 6.0, 1.2, 5.4 ] )
data * 2
```

ans:

```
array([ 9. , 12. ,  2.4, 10.8])
```

Different from the normal list in Python!!!

numpy

```
>>> x = np.array( [ [ 1,2 ], [ 3,4 ] ] )
#look at how many []?
```

numpy

```
>>> x = np.array( [ [ 1,2 ], [ 3,4 ] ] )
#look at how many []?

array([[1, 2],
       [3, 4]])


>>> x.shape
```

numpy

```
>>> x = np.array( [ [ 1,2 ], [ 3,4 ] ] )
#look at how many []?

array([[1, 2],
       [3, 4]])


>>> x.shape
```

(2,2)

```
>>> x.dtype
```

numpy

```
>>> x = np.array( [ [ 1,2 ], [ 3,4 ] ] )
#look at how many []?
```

```
array([[1, 2],
       [3, 4]])
```

```
>>> x.shape
```

(2,2)

```
>>> x.dtype
```

dtype('int32')

```
>>> x = np.array( [ [ 1,2 ], [ 3,4 ] ]
                        ,dtype=np.float64 )
```

numpy

```
>>> x = np.array( [ [ 1,2 ], [ 3,4 ] ] )
#look at how many []?

array([[1, 2],
       [3, 4]])


>>> x.shape
```

(2,2)

```
>>> x.dtype
```

dtype('int32')

```
>>> x = np.array( [ [ 1,2 ], [ 3,4 ] ]
                        ,dtype=np.float64 )

array([[1., 2.],
       [3., 4.]])
```

numpy

```
>>> x.T
```

numpy

```
>>> x.T

array([[1, 3],
       [2, 4]])


>>> x * x  # element-wise, not matrix-like!
```

numpy

```
>>> x.T

array([[1, 3],
       [2, 4]])


>>> x * x  # element-wise, not matrix-like!

array([[ 1,  4],
       [ 9, 16]])


>>> x + 2
```

numpy

```
>>> x.T

array([[1, 3],
       [2, 4]])


>>> x * x  # element-wise, not matrix-like!

array([[ 1,  4],
       [ 9, 16]])


>>> x + 2

array([[ 3,  4],
       [ 5, 6]])
```

numpy

```
>>> np.sqrt(x)
```

numpy

```
>>> np.sqrt(x)
array([[1, 1.414],
       [1.732, 2]])

>>> np.sin(x)
```

numpy

```
>>> np.sqrt(x)
array([[1, 1.414],
       [1.732, 2]])

>>> np.sin(x)
array([[0.841,  0.909],
       [0.141, -0.756]])
```

numpy

```
>>> np.zeros( ( 3,3 ) )
```

numpy

```
>>> np.zeros( ( 3,3 ) )
```

Ans:

```
array([[0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.]])
```

```
>>> np.ones( ( 4,2 ) )
```

numpy

```
>>> np.zeros( ( 3,3 ) )
```

Ans:

```
array([[0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.]])
```

```
>>> np.ones( ( 4,2 ) )
```

Ans:

```
array([[1., 1.],
       [1., 1.],
       [1., 1.],
       [1., 1.]])
```

numpy

```
>>> np.eye( 4 )
```

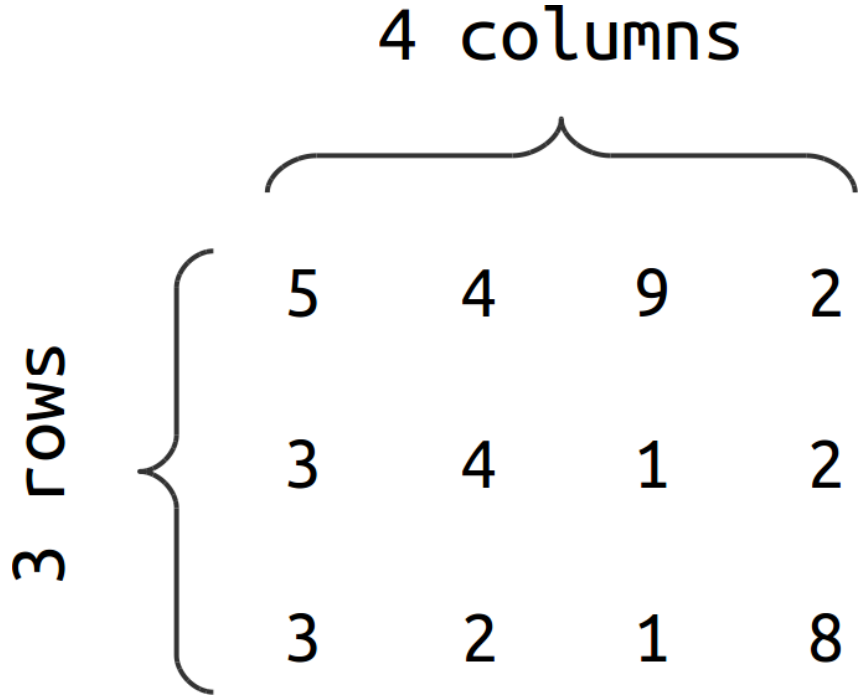
numpy

```
>>> np.eye( 4 )
```

Ans:

```
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

Indexing arrays



`numpy` indexes by

`array[row][col]` or

`array[ row,col ]`

numpy

```
>>> x[ :,1 ] # element [1] of all the rows

>>> x[ 1,: ] # all the elements of the row [1]

>>> x.tolist() # convert to a python list

>>> x.sort(i)
#sort by column if i=0
#sort by row if i=1 or nothing

>>> x.argsort(i)
#sort by column if i=0, by row if i=1 or nothing
#calculate if the original matrix is sorted
#where the original elements will be
#in the sorted matrix
#But the matrix is NOT sorted
```

numpy

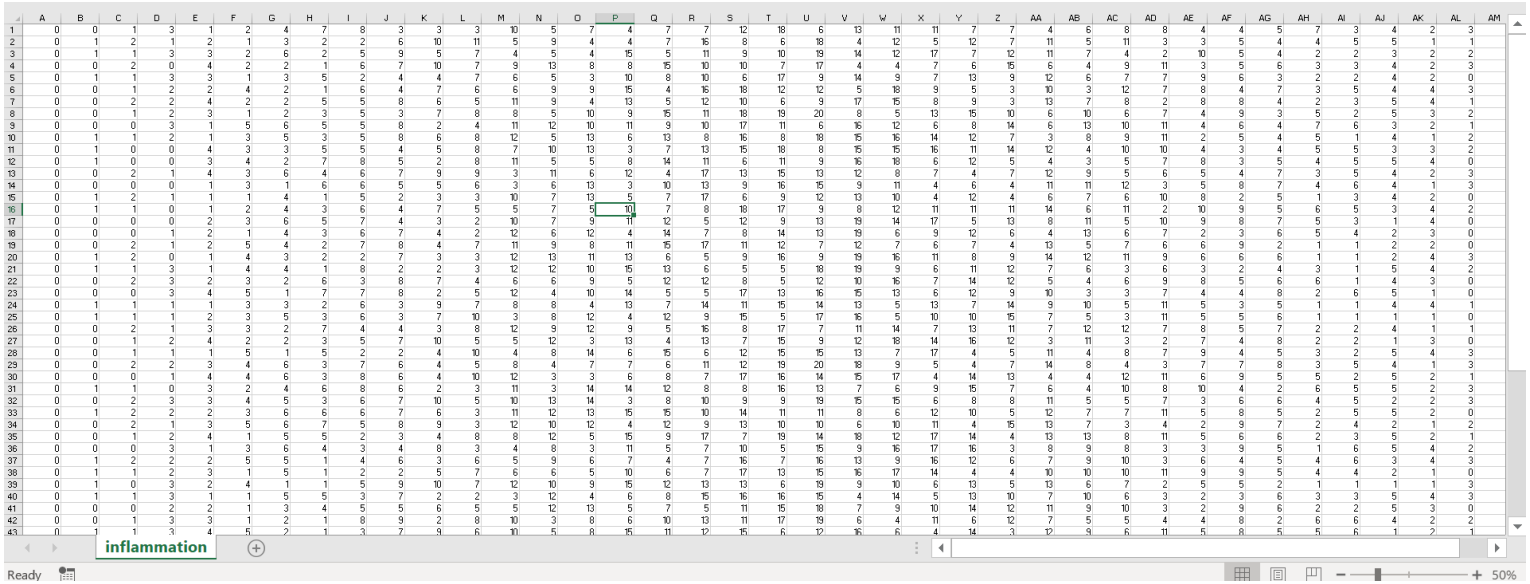
```
>>> x.max(i)
#max by column if i=0, by row if i=1
#max of everything in x if nothing

>>> x.min(i)
#min by column if i=0, by row if i=1 or nothing
#min of everything in x if nothing

>>> x.mean(i)
#mean by column if i=0, by row if i=1 or nothing
#mean of everything in x if nothing
```

numpy: load .csv type files

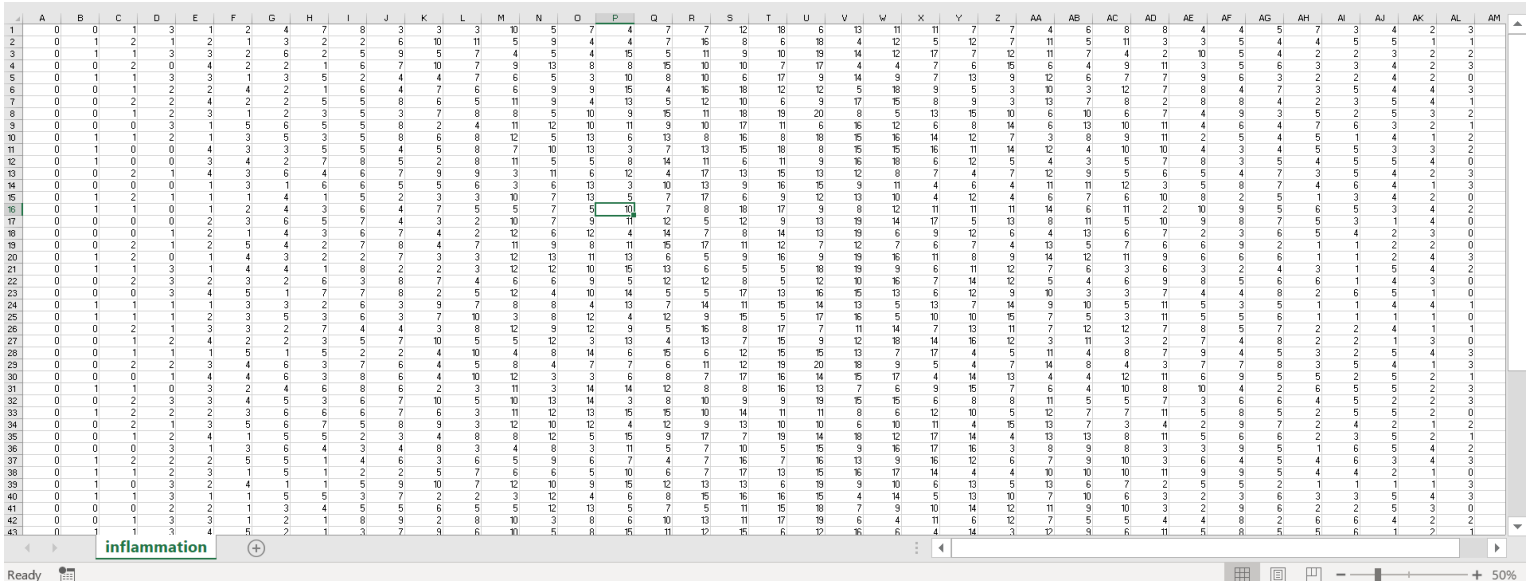
Consider a data set containing patient inflammation records for 60 patients over 40 days, contained in `inflammation.csv`.




```
data = np.loadtxt( './data/inflammation.csv',
                   delimiter=',' )
print( data.shape )
```

numpy: load .csv type files

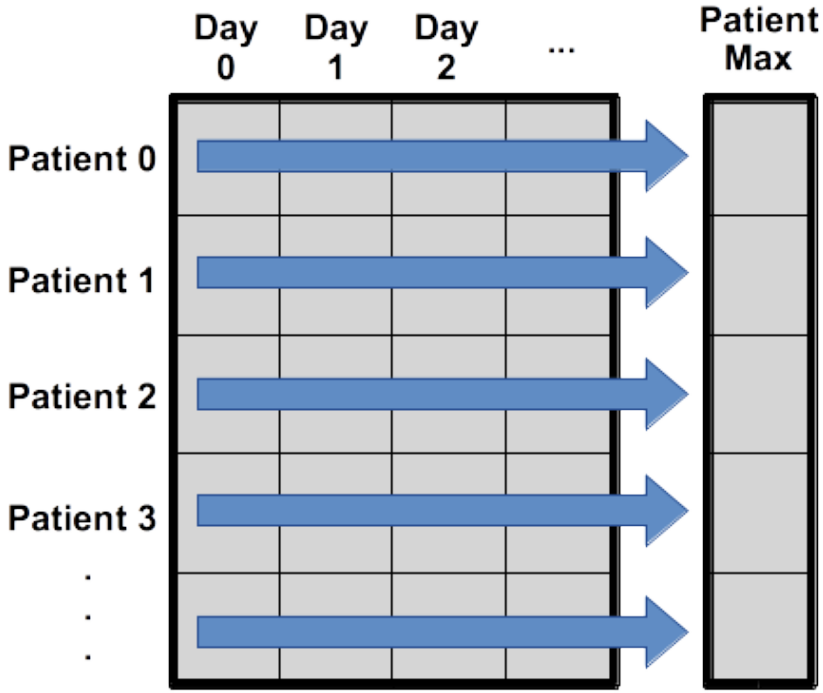
Consider a data set containing patient inflammation records for 60 patients over 40 days, contained in `inflammation.csv`.




```
data = np.loadtxt( './data/inflammation.csv',
                   delimiter=',' )
print( data.shape )
```

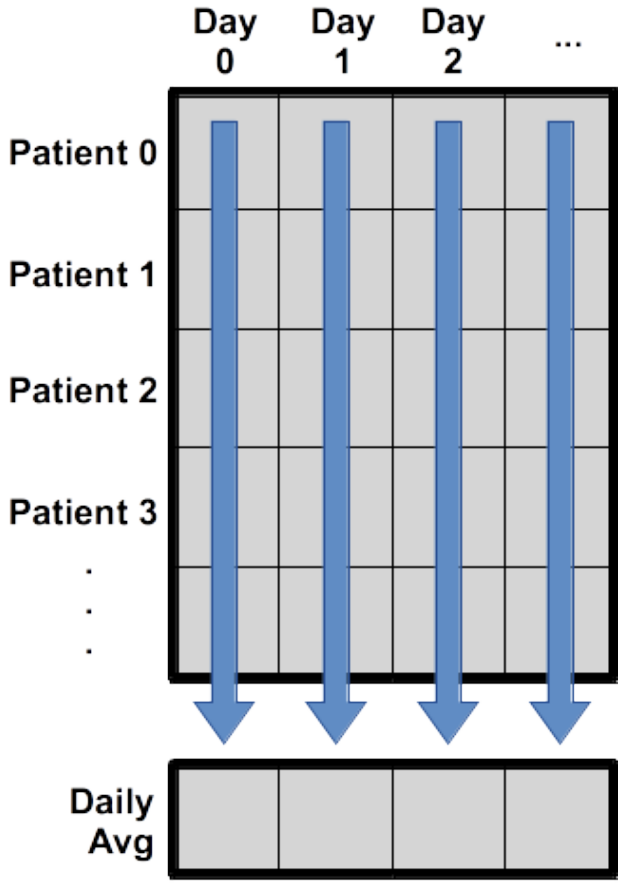
Ans: (60,40)

numpy



Max for each patient

`data.max(axis=1)`



Average for each day

`data.mean(axis=0)`

Axes can be a bit tricky; test them if you need to.

Question

```
import numpy as np
x = np.array( [ 5,1,3 ] )
x *= 2
```

What is the value of `x`?

A `[ 10,2,6 ]`

B `array( [ 10,2,6 ] )`

C `[ 5,1,3,5,1,3 ]`

D `array( [ [ 5,1,3 ], [ 5,1,3 ] ] )`

Question

```
import numpy as np
x = np.array( [ 5,1,3 ] )
x *= 2
```

What is the value of `x`?

A `[ 10,2,6 ]`

B `array( [ 10,2,6 ] )` *******

C `[ 5,1,3,5,1,3 ]`

D `array( [[ 5,1,3 ], [ 5,1,3 ]] )`

Question

```
import numpy as np
x =  np.array( [ 1 ] * 2 )
x += 1
```

What is the final value of `x`?

A `array( [2] )`

B `array( [ 1,1,1 ] )`

C `array( [ 2,2 ] )`

D `array( [ 3 ] )`

Question

```
import numpy as np
x =  np.array( [ 1 ] * 2 )
x += 1
```

What is the final value of `x`?

A `array( [2] )`

B `array( [ 1,1,1 ] )`

C `array( [ 2,2 ] )` ******

D `array( [ 3 ] )`

Data types

`numpy` supports many possible data types:

- `bool`
- `int16`, `int32`, `int64`
- `float16`, `float32`, `float64`
- `complex64`, `complex128`

Data types

`numpy` supports many possible data types:

- `bool`
- `int16`, `int32`, `int64`
- `float16`, `float32`, `float64`
- `complex64`, `complex128`

You frequently don't need to specify the type.

For the most part, stick with `bool`, `int64`, and `float64` (most accurate).

Specify (and query) with `dtype` method:

```
a = [ 3,2,4 ]
x = np.array( a,dtype=np.float64 )
x.dtype
```

linspace

```
>>> w = np.linspace( 0,10,51 )
array([ 0. ,  0.2,  0.4,  0.6,  0.8,  1. ,  1.2,
1.4,  1.6,  1.8,  2. , 2.2,  2.4,  2.6,  2.8,
3. ,  3.2,  3.4,  3.6,  3.8,  4. ,  4.2, 4.4,
4.6,  4.8,  5. ,  5.2,  5.4,  5.6,  5.8,  6. ,
6.2,  6.4, 6.6,  6.8,  7. ,  7.2,  7.4,  7.6,
7.8,  8. ,  8.2,  8.4,  8.6, 8.8,  9. ,  9.2,
9.4,  9.6,  9.8, 10. ])
```

`np.linspace( start, finish, n)`

Produce arrays from `start` to `finish` of `n` points (*not* spacing!).

Excellent for grids and coordinates.

Summary

Summary

A. Numpy and its mathematics library
B. Convert `list` to `numpy array`
C. `import numpy as np` and np.methods
D. `np.linspace(s,d,x)`