Programming and Data Structures with Python

Lecture 17, 22 November 2021

Using numpy

- Arrays and lists
- Arrays are "homogenous" with regular structure
- Lists are flexible

Load numpy

```
import numpy as np
```

```
a = np.array([1,2,3])
a
```

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

```
b = np.array(range(10))
b
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
c = np.array([[0,1],[2,3]])
c
```

```
array([[0, 1],
       [2, 3]])
```

Indexing and slicing

```
a = np.arange(10)**3
a
```

```
array([  0,   1,   8,  27,  64, 125, 216, 343, 512, 729])
```

```
a[2], a[2:5]
```

```
(8, array([ 8, 27, 64]))
```

```
a[:6:2] = -1000  # equivalent to a[0:6:2] = -1000
a
```

```
array([-1000,     1, -1000,    27, -1000,   125,   216,   343,   512,
         729])
```

```
def f(x,y):
    return(10*x +  y)
```

```
f(5,7)
```

```
57
```

```
b = np.fromfunction(f,(5,4),dtype=int)
b
```

```
array([[ 0,  1,  2,  3],
       [10, 11, 12, 13],
       [20, 21, 22, 23],
       [30, 31, 32, 33],
       [40, 41, 42, 43]])
```

```
b[2,3]  # Not b[2][3]
```

```
23
```

```
b[0:5, 1] # each row in the second column of b
```

```
array([ 1, 11, 21, 31, 41])
```

```
b[ : ,1]  # equivalent to the previous example
```

```
array([ 1, 11, 21, 31, 41])
```

```
b[1:3, :]  # each column in the second and third row of b
```

```
array([[10, 11, 12, 13],
       [20, 21, 22, 23]])
```

```
b[1:4,1:3]
```

```
array([[11, 12],
       [21, 22],
       [31, 32]])
```

Iterating over elements

```
print(b)
```

```
[[ 0  1  2  3]
 [10 11 12 13]
 [20 21 22 23]
 [30 31 32 33]
 [40 41 42 43]]
```

```
for row in b:
    print(row)
```

```
[0 1 2 3]
[10 11 12 13]
[20 21 22 23]
[30 31 32 33]
[40 41 42 43]
```

```
for element in b.flat:
    print(element,end=' ')
```

```
0 1 2 3 10 11 12 13 20 21 22 23 30 31 32 33 40 41 42 43
```

Stacking arrays

```
a = np.zeros((5,7),dtype=int)
a
```

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

```
a = np.floor(10*np.random.random((2,2)))
b = np.floor(10*np.random.random((2,2)))
print(a,b)
```

```
[[2. 8.]
 [2. 7.]] [[6. 7.]
 [1. 7.]]
```

```
np.vstack((a,b))
```

```
array([[2., 8.],
       [2., 7.],
       [6., 7.],
       [1., 7.]])
```

```
c = np.floor(10*np.random.random((3,3)))
```

```
c
```

```
array([[7., 5., 2.],
       [1., 9., 9.],
       [3., 9., 8.]])
```

```
np.vstack((a,c))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-24-eb5bc5f1560c> in <module>()
----> 1 np.vstack((a,c))

<__array_function__ internals> in vstack(*args, **kwargs)

/usr/local/lib/python3.7/dist-packages/numpy/core/shape_base.py in vstack(tup)
    281     if not isinstance(arrs, list):
    282         arrs = [arrs]
--> 283     return _nx.concatenate(arrs, 0)
    284 
    285 

<__array_function__ internals> in concatenate(*args, **kwargs)

ValueError: all the input array dimensions for the concatenation axis must match exactly, but along dimension 1, the array at index 0
has size 2 and the array at index 1 has size 3
```

```
np.hstack((a,b))
```

```
array([[2., 8., 6., 7.],
       [2., 7., 1., 7.]])
```

```
np.hstack((b,c))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-27-71093adb15f8> in <module>()
----> 1 np.hstack((b,c))

<__array_function__ internals> in hstack(*args, **kwargs)

/usr/local/lib/python3.7/dist-packages/numpy/core/shape_base.py in hstack(tup)
    344         return _nx.concatenate(arrs, 0)
    345     else:
--> 346         return _nx.concatenate(arrs, 1)
    347 
    348 

<__array_function__ internals> in concatenate(*args, **kwargs)

ValueError: all the input array dimensions for the concatenation axis must match exactly, but along dimension 0, the array at index 0
has size 2 and the array at index 1 has size 3
```

SEARCH STACK OVERFLOW

Splitting arrays

```
a = np.floor(10*np.random.random((2,12)))
a
```

```
array([[9., 5., 7., 2., 3., 3., 6., 4., 0., 4., 8., 1.],
       [0., 6., 9., 5., 6., 4., 8., 7., 4., 7., 7., 1.]])
```

```
np.hsplit(a,6)
```

```
[array([[9., 5.],
        [0., 6.]]), array([[7., 2.],
        [9., 5.]]), array([[3., 3.],
        [6., 4.]]), array([[6., 4.],
        [8., 7.]]), array([[0., 4.],
        [4., 7.]]), array([[8., 1.],
        [7., 1.]])]
```

```
np.hsplit(a,(2,5,7)) # Split a after the third and the fourth column
```

```
[array([[9., 5.],
        [0., 6.]]), array([[7., 2., 3.],
        [9., 5., 6.]]), array([[3., 6.],
        [4., 8.]]), array([[4., 0., 4., 8., 1.],
        [7., 4., 7., 7., 1.]])]
```

```
np.vsplit(a,2) # Split a vertically
```

```
[array([[9., 5., 7., 2., 3., 3., 6., 4., 0., 4., 8., 1.]]),
 array([[0., 6., 9., 5., 6., 4., 8., 7., 4., 7., 7., 1.]])]
```

Copy and view

```
c = a.copy()  # Creates a disjoint copy of the array
d = a.view()  # Creates another link to the same array
```

```
a, c, d
```

```
(array([[9., 5., 7., 2., 3., 3., 6., 4., 0., 4., 8., 1.],
        [0., 6., 9., 5., 6., 4., 8., 7., 4., 7., 7., 1.]]),
 array([[9., 5., 7., 2., 3., 3., 6., 4., 0., 4., 8., 1.],
        [0., 6., 9., 5., 6., 4., 8., 7., 4., 7., 7., 1.]]),
 array([[9., 5., 7., 2., 3., 3., 6., 4., 0., 4., 8., 1.],
        [0., 6., 9., 5., 6., 4., 8., 7., 4., 7., 7., 1.]]))
```

```
c[0,4] = 88
```

```
a, c, d
```

```
(array([[9., 5., 7., 2., 3., 3., 6., 4., 0., 4., 8., 1.],
        [0., 6., 9., 5., 6., 4., 8., 7., 4., 7., 7., 1.]]),
 array([[ 9.,  5.,  7.,  2., 88.,  3.,  6.,  4.,  0.,  4.,  8.,  1.],
        [ 0.,  6.,  9.,  5.,  6.,  4.,  8.,  7.,  4.,  7.,  7.,  1.]]),
 array([[9., 5., 7., 2., 3., 3., 6., 4., 0., 4., 8., 1.],
        [0., 6., 9., 5., 6., 4., 8., 7., 4., 7., 7., 1.]]))
```

```
d[0,5] = 66
```

```
a, c, d
```

```
(array([[ 9.,  5.,  7.,  2.,  3., 66.,  6.,  4.,  0.,  4.,  8.,  1.],
        [ 0.,  6.,  9.,  5.,  6.,  4.,  8.,  7.,  4.,  7.,  7.,  1.]]),
 array([[ 9.,  5.,  7.,  2., 88.,  3.,  6.,  4.,  0.,  4.,  8.,  1.],
        [ 0.,  6.,  9.,  5.,  6.,  4.,  8.,  7.,  4.,  7.,  7.,  1.]]),
 array([[ 9.,  5.,  7.,  2.,  3., 66.,  6.,  4.,  0.,  4.,  8.,  1.],
        [ 0.,  6.,  9.,  5.,  6.,  4.,  8.,  7.,  4.,  7.,  7.,  1.]]))
```

Use `base` to check whether two arrays have same underlying elements

```
c.base is a, d.base is a
```

```
(False, True)
```

Can reshape arrays with same base without affecting the shape of the other

```
d.shape = 4,6
```

```
a,c,d
```

```
(array([[ 9.,  5.,  7.,  2.,  3., 66.,  6.,  4.,  0.,  4.,  8.,  1.],
        [ 0.,  6.,  9.,  5.,  6.,  4.,  8.,  7.,  4.,  7.,  7.,  1.]]),
 array([[ 9.,  5.,  7.,  2., 88.,  3.,  6.,  4.,  0.,  4.,  8.,  1.],
        [ 0.,  6.,  9.,  5.,  6.,  4.,  8.,  7.,  4.,  7.,  7.,  1.]]),
 array([[ 9.,  5.,  7.,  2.,  3., 66.],
        [ 6.,  4.,  0.,  4.,  8.,  1.],
        [ 0.,  6.,  9.,  5.,  6.,  4.],
        [ 8.,  7.,  4.,  7.,  7.,  1.]]))
```

```
d[2,4] = 99
```

```
a, c, d
```

```
(array([[ 9.,  5.,  7.,  2.,  3., 66.,  6.,  4.,  0.,  4.,  8.,  1.],
        [ 0.,  6.,  9.,  5., 99.,  4.,  8.,  7.,  4.,  7.,  7.,  1.]]),
 array([[ 9.,  5.,  7.,  2., 88.,  3.,  6.,  4.,  0.,  4.,  8.,  1.],
        [ 0.,  6.,  9.,  5.,  6.,  4.,  8.,  7.,  4.,  7.,  7.,  1.]]),
 array([[ 9.,  5.,  7.,  2.,  3., 66.],
        [ 6.,  4.,  0.,  4.,  8.,  1.],
        [ 0.,  6.,  9.,  5., 99.,  4.],
        [ 8.,  7.,  4.,  7.,  7.,  1.]]))
```

```
a = np.array([[1,2],[3,4]])
b = np.array([[5,6],[7,8]])
```

```
a,b
```

```
(array([[1, 2],
        [3, 4]]), array([[5, 6],
        [7, 8]]))
```

```
a+b  # Pointwise addition
```

```
array([[ 6,  8],
       [10, 12]])
```

```
a*b  # Pointwise multiplication
```

```
array([[ 5, 12],
       [21, 32]])
```

```
np.matmul(a,b)  # Normal matrix multiplication
```

```
array([[19, 22],
       [43, 50]])
```

```
a.T   # Transpose
```

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

```
np.linalg.inv(a)  # Matrix inverse
```

```
array([[-2. ,  1. ],
       [ 1.5, -0.5]])
```

```
np.matmul(a,np.linalg.inv(a))  # a a^-1
```

```
array([[1.0000000e+00, 0.0000000e+00],
       [8.8817842e-16, 1.0000000e+00]])
```

Pandas (Python and data analysis)

- Built on top of numpy

Series and data frames

- Numpy defines homogeneous n-dimensional arrays
- Data science works with tables: 2-dimensional arrays
- Pandas has two fundamental data structures
  - Series : A column of data
  - Data Frame : A table of data

Key difference

- Numpy indices are always [0..n-1] in each dimension
- Pandas allows more flexible “named” indices for rows and columns
  - Dictionary vs list

Load pandas

- Don't need to import numpy unless one is separately using numpy arrays

```
import pandas as pd
```

Create a series

- Convert a sequence into a series (column)

```
h = ('AA', '2012-02-01', 100, 10.2)
s = pd.Series(h)
type(s)
```

```
pandas.core.series.Series
```

```
s
```

```
0            AA
1    2012-02-01
2           100
3          10.2
dtype: object
```

- Convert a dictionary to a series
- Keys become "row indices"

```
d = {'name' : 'IBM', 'date' :'2010-09-08', 'shares' : 100, 'price' : 10.2}
ds = pd.Series(d)
type(ds)
```

```
pandas.core.series.Series
```

```
ds
```

```
name             IBM
date      2010-09-08
shares           100
price           10.2
dtype: object
```

Creating an index

```
f = ['FB', '2001-08-02', 90, 3.2]
fs = pd.Series(f, index = ['name','date', 'shares', 'price'])
```

```
fs
```

```
name              FB
date      2001-08-02
shares            90
price            3.2
dtype: object
```

Accessing elements

- Use named index, or position
- Use slices, sublists

```
fs['shares']
```

```
90
```

```
fs[0]
```

```
'FB'
```

```
fs[0:2]
```

```
name            FB
date    2001-08-02
dtype: object
```

```
fs[[0,2]]
```

```
name      FB
shares    90
dtype: object
```

```
fs['name':'price']
```

```
name              FB
date      2001-08-02
shares            90
price            3.2
dtype: object
```

Data frames

- A table is a sequence of columns
- A data frame is a sequence of series

```
data1 = [ ['AA', 'IBM', 'GOOG'],
          ['2001-12-01', '2012-02-10', '2010-04-09'],
          [100, 30, 90],
          [12.3, 10.3, 32.2]
        ]
df1 = pd.DataFrame(data1)
```

```
df1
```

	0	1	2
0	AA	IBM	GOOG
1	2001-12-01	2012-02-10	2010-04-09
2	100	30	90
3	12.3	10.3	32.2

```
data2 = {'name' : ['AA', 'IBM', 'GOOG'],
        'date' : ['2001-12-01', '2012-02-10', '2010-04-09'],
        'shares' : [100, 30, 90],
        'price' : [12.3, 10.3, 32.2]
}
df2 = pd.DataFrame(data2)
```

```
df2
```

	name	date	shares	price
0	AA	2001-12-01	100	12.3
1	IBM	2012-02-10	30	10.3
2	GOOG	2010-04-09	90	32.2