UC Berkeley Extension

Python for Data Analysis and Scientific Computing

X433.3 (2 semester units in COMPSCI)

Instructor Alexander I. Iliev, Ph.D.

Course Content Outline

- **Introduction to Python®**
- Python - pros and cons
- Installing the environment with core packages
- Python modules, packages and scientific blocks
- Working with the shell, IPyton and the editor HW1

- **Basic language specifics 1/2**
- Basic arithmetic operations, assignment operators, data types, containers
- Control flow (if/elif/else)
- Conditional expressions
- Iterative programming (for/continue/while/break)
- Functions: definition, return values, local vs. global variables

- **Basic language specifics 2/2**
- Classes / Functions (cont.): objects, methods, passing by value and reference
- Scripts, modules, packages
- I/O interaction with files
- Standard library
- Exceptions

- **NumPy 1/3**
- Why NumPy?
- Data type objects
- NumPy arrays
- Indexing and slicing of arrays HW2

- **Matplotlib**
- What is Matplotlib?
- Basic plotting
- Tools: title, labels, legend, axis, points, subplots, etc.
- Advanced plotting: scatter, pie, bar, 3D plots, etc. HW3

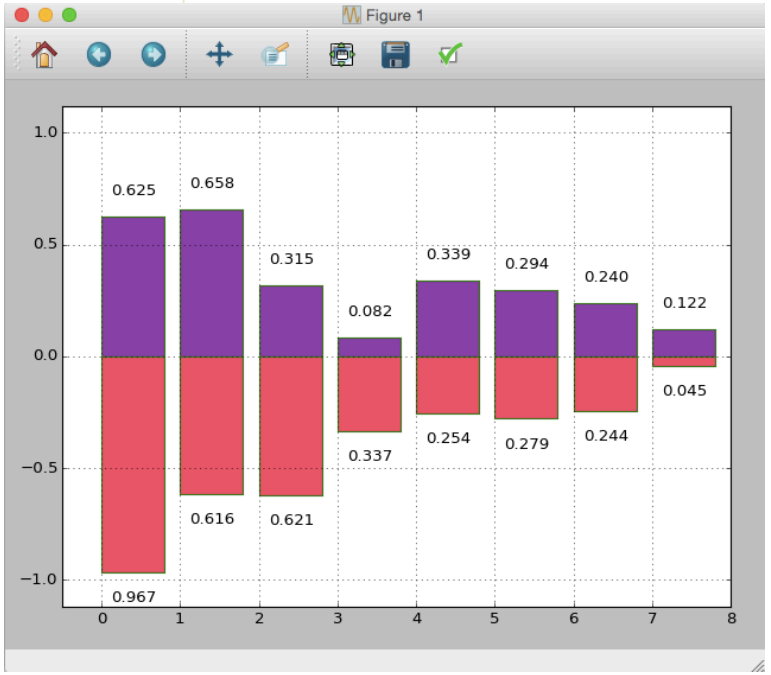
Advanced plotting

- Advanced plotting
  - There are many different types of 2-D and 3-D plotting choices in Matplotlib
  - Matplotlib provides a huge variety of plots aiming at specific visualization of results
  - Some of these types are:
    - Bar plots
    - Scatter plots
    - Imshow
    - Histogram
    - Pie charts
    - Contour plots
    - Polar axis
    - 3-D plots
    - *Text plots*
    - *Grids*
    - *Quiver plots*

Advanced plotting

- Advanced plotting: *bar plot*

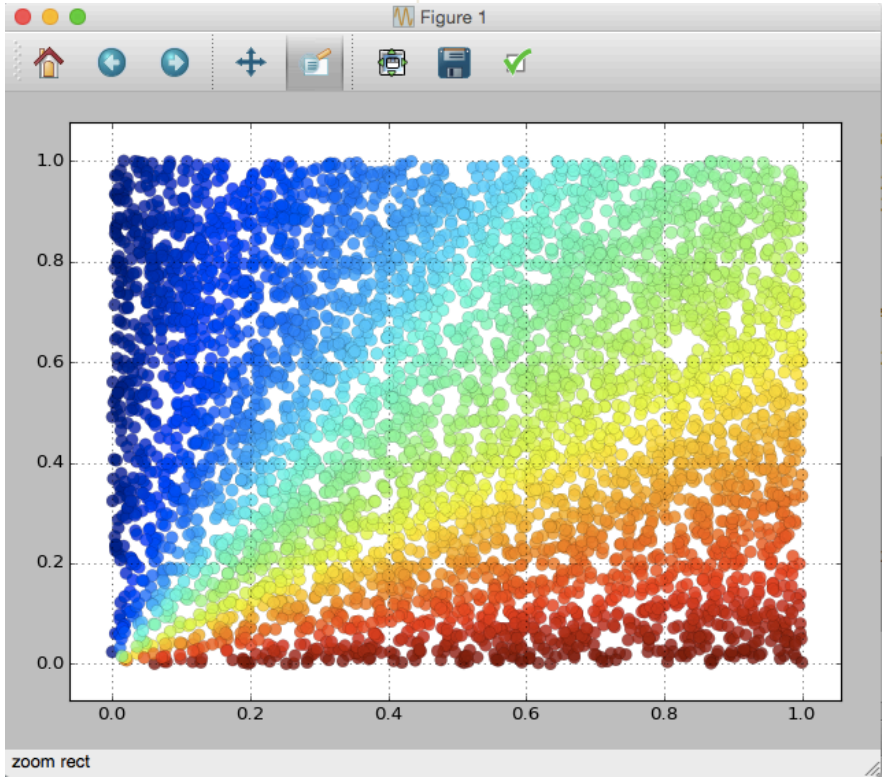
```
## Advanced plotting:
# Bar plot:
import pylab as plb

k = 8
x = plb.arange(k)
y1 = plb.rand(k) * (1 - x / k)
y2 = plb.rand(k) * (1 - x / k)
plb.axes([0.075, 0.075 , .88, .88])

plb.bar(x, +y1, facecolor='#9922aa', edgecolor='green')
plb.bar(x, -y2, facecolor='#ff3366', edgecolor='green')

for a, b in zip(x, y1):
    plb.text(a+0.41, b+0.08, '%.3f' % b, ha='center', va='bottom')
for a, b in zip(x, y2):
    plb.text(a+0.41, -b-0.08, '%.3f' % b, ha='center', va= 'top')

plb.xlim(-.5, k), plb.ylim(-1.12, +1.12)
plb.grid(True)
plb.show()
```



Advanced plotting

- Advanced plotting: *scatter plot*

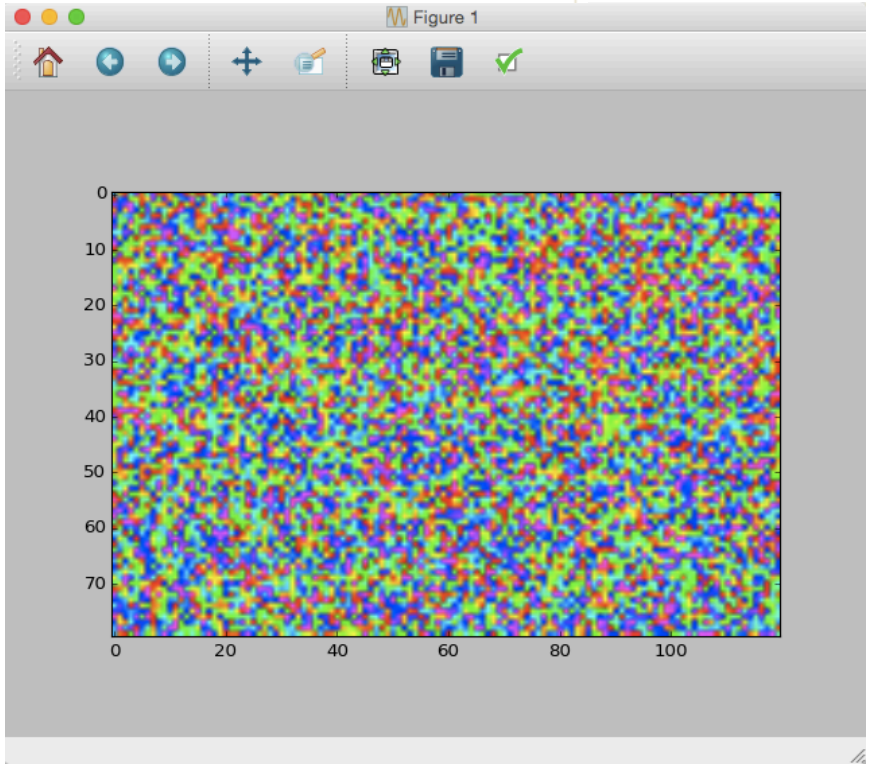
```
312  ## Scatter plot:
313  import pylab as plb
314
315  x = plb.rand(1,2,1500)
316  y = plb.rand(1,2,1500)
317  plb.axes([0.075, 0.075 , .88, .88])
318
319  plb.cla()   # clear the current axis
320  plb.scatter(x, y, s=65, alpha=.75, linewidth=.125,
321      c=plb.arctan2(x, y))
322
323  plb.grid(True)
324  plb.xlim(-0.085,1.085), plb.ylim(-0.085,1.085)
325  plb.pause(1)
```



Advanced plotting

- Advanced plotting: *image plot*

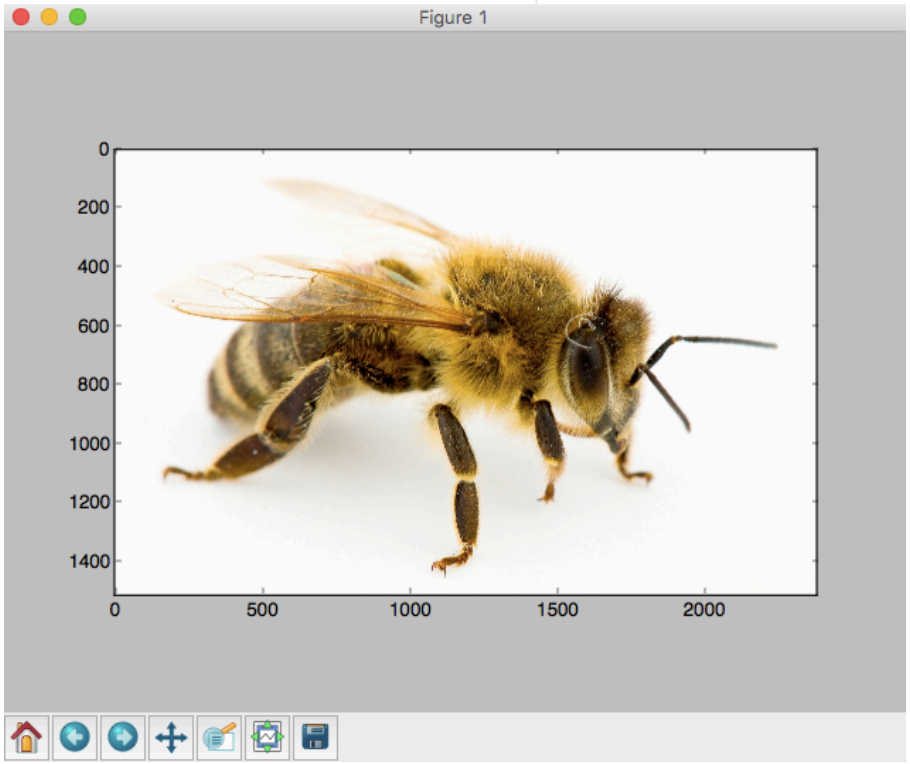
```
358 ## Image 1/2:
359 plb.cla()
360 array = plb.random((80, 120))
361 plb.imshow(array, cmap=plb.cm.gist_rainbow) # with a specific colormap
362 plb.pause(1)
```



Advanced plotting

- Advanced plotting: *image plot*

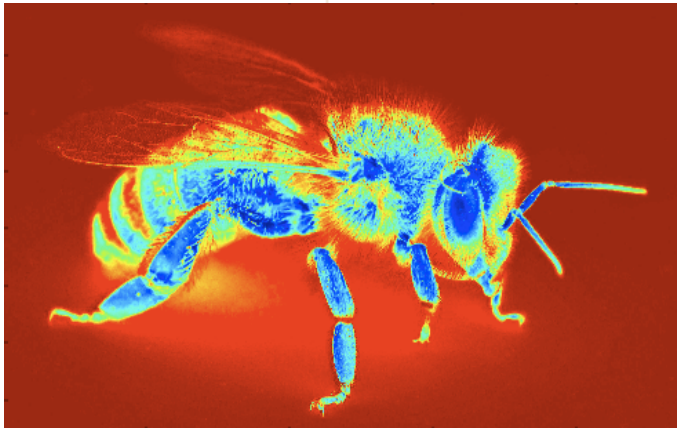
```
364  ## Image 2/2:
365  import matplotlib.image as img
366  import matplotlib.pyplot as plt
367
368  image = img.imread('files/lecture5/bee.jpg')
369  plt.imshow(image)
370  plt.pause(1)
```



Advanced plotting

- Advanced plotting: *image plot 1/3*

```
372 # luminosity display using 1-channel only (no RGB color).
373 # A default colormap (lookup tabel = LUT) is applied called 'jet':
374 luminosity = image[:,:,0]
375 plt.imshow(luminosity)
376 plt.pause(5)
377
378 # Other colormaps can be:
379 plt.imshow(luminosity, cmap="hot")
380 plt.pause(5)
381 plt.imshow(luminosity, cmap="spectral")
382 plt.pause(5)
```



Advanced plotting

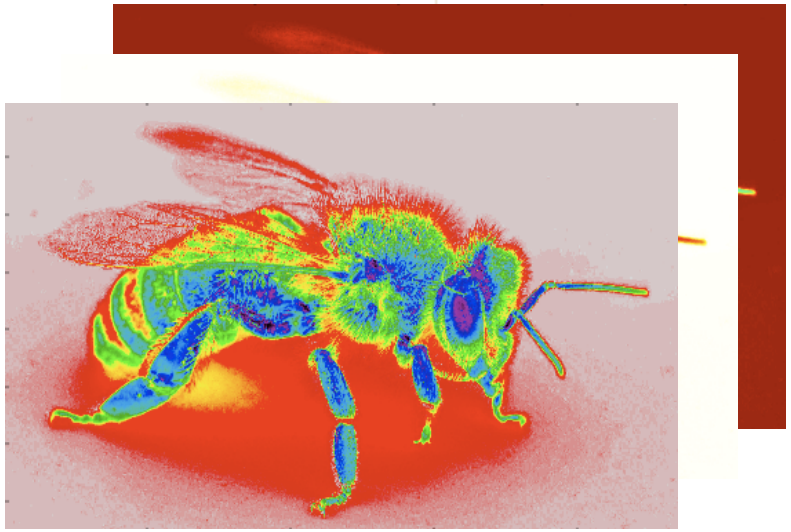
- Advanced plotting: *image plot 2/3*

```
372 # luminosity display using 1-channel only (no RGB color).
373 # A default colormap (lookup tabel = LUT) is applied called 'jet':
374 luminosity = image[:,:,0]
375 plt.imshow(luminosity)
376 plt.pause(5)
377
378 # Other colormaps can be:
379 plt.imshow(luminosity, cmap="hot")
380 plt.pause(5)
381 plt.imshow(luminosity, cmap="spectral")
382 plt.pause(5)
```

Advanced plotting

- Advanced plotting: *image plot 3/3*

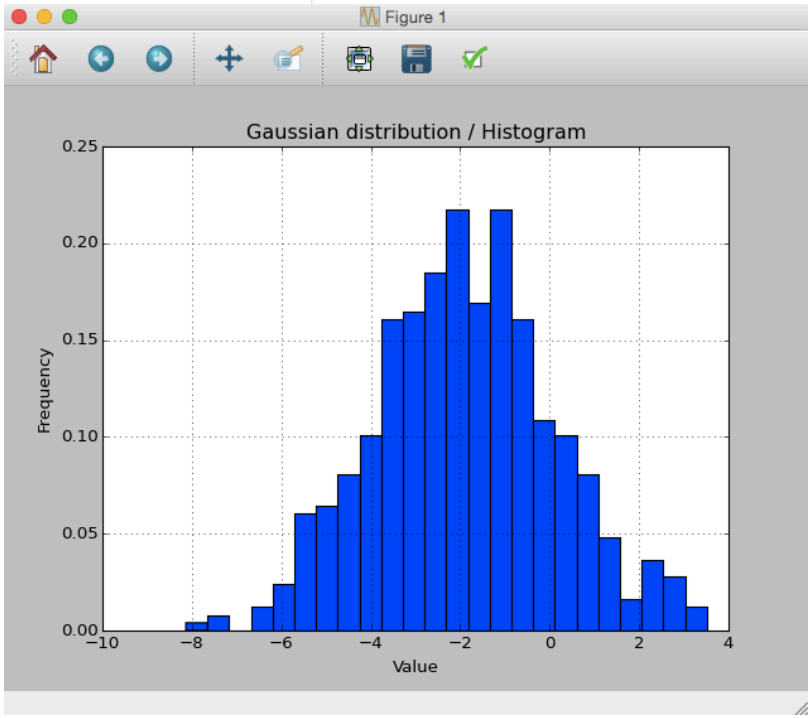
```
372 # luminosity display using 1-channel only (no RGB color).
373 # A default colormap (lookup tabel = LUT) is applied called 'jet':
374 luminosity = image[:,:,0]
375 plt.imshow(luminosity)
376 plt.pause(5)
377
378 # Other colormaps can be:
379 plt.imshow(luminosity, cmap="hot")
380 plt.pause(5)
381 plt.imshow(luminosity, cmap="spectral")
382 plt.pause(5)
```



Advanced plotting

- Advanced plotting: *histogram plot 1/2*

```
285 # Histogram 1/2:
286 import pylab as plb
287
288 plb.figure(1)
289 gaus_dist = plb.normal(-2,2,size=512) # create a random floating point vector
290
291 # plot the histogram with specific: bin number
292 plb.hist(gaus_dist, normed=True, bins=24) # default: bins=10, color='blue'
293
294 plb.title("Gaussian distribution / Histogram")
295 plb.xlabel("Value")
296 plb.ylabel("Frequency")
297 plb.grid(True)
298 plb.show()
```



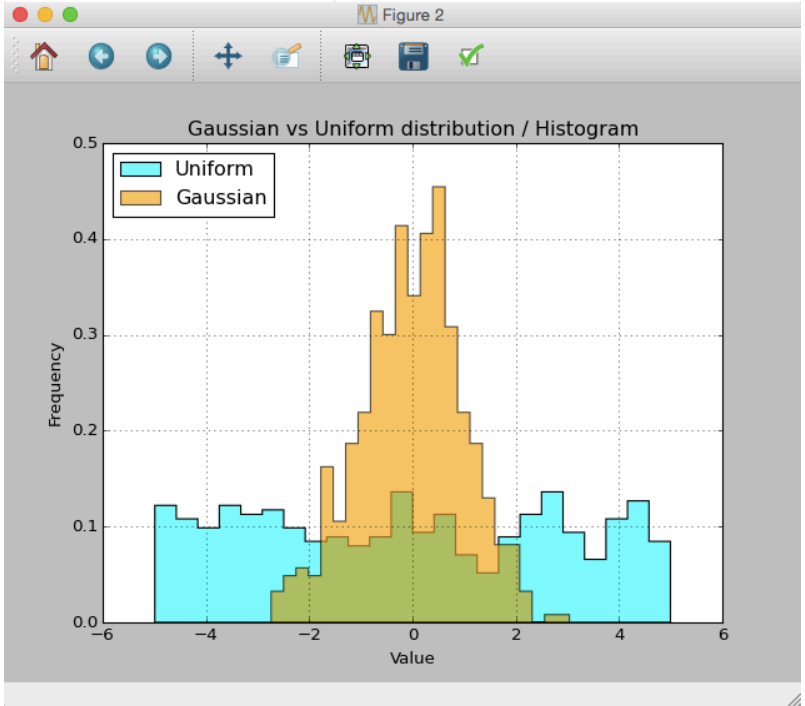
Histogram is great for visualizing statistical distribution of a set of variables in a given pool of samples, divided into classes called bins

Advanced plotting

- Advanced plotting: *histogram plot 2/2*

```
300 # Histogram 2/2:
301 plb.figure(2)
302 gaus_dist = plb.normal(size=512)
303 unif_dist = plb.uniform(-5,5,size=512) # create uniform distribution vector
304
305 # plot the histogram with specific: bin number, color, transparency, label
306 plb.hist(unif_dist, bins=24, histtype='stepfilled',
307             normed=True, color='cyan', label='Uniform')
308 plb.hist(gaus_dist, bins=24, histtype='stepfilled',
309             normed=True, color='orange', label='Gaussian', alpha=0.65)
310
311 plb.legend(loc='upper left')
312 plb.title("Gaussian vs Uniform distribution / Histogram")
313 plb.xlabel("Value")
314 plb.ylabel("Frequency")
315 plb.grid(True)
316 plb.show()
```

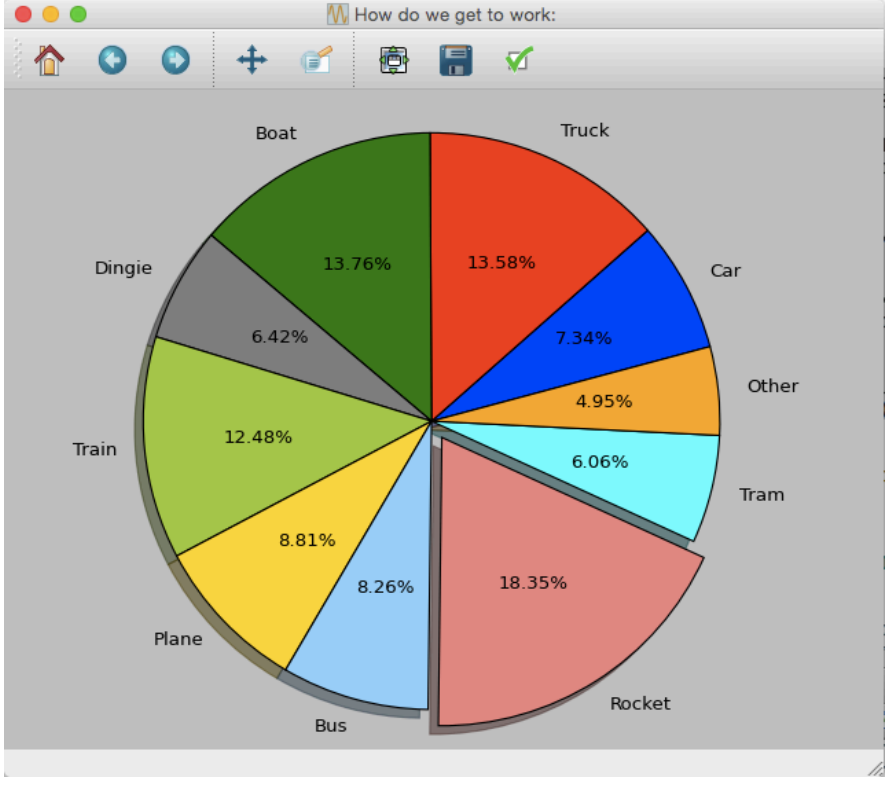
Histogram is great for visualizing statistical distribution of a set of variables in a given pool of samples, divided into classes called bins



Advanced plotting

- Advanced plotting: *pie chart*

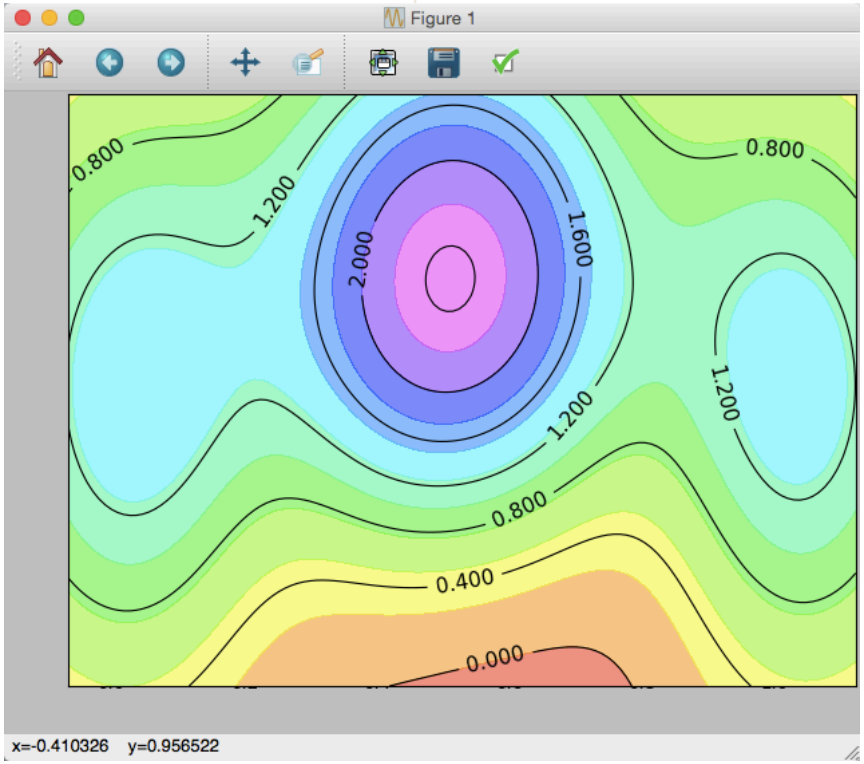
```
318 # Pie chart:
319 plb.figure('How do we get to work:')
320 plb.axes([0.035, 0.035, 0.9, 0.9])
321
322 l = 'Car', 'Truck', 'Boat', 'Dingie', 'Train', 'Plane', 'Bus', 'Rocket', 'Tram', 'Other'
323 b = plb.round_(plb.random(10), decimals=2)
324 c = ['blue', 'red', 'green', 'gray', 'yellowgreen',
325      'gold', 'lightskyblue', 'lightcoral', 'cyan', 'orange']
326 e = (0, 0, 0, 0, 0, 0, 0, 0.05, 0, 0) # 'explode' the 8th slice only - 'Rocket'
327
328 plb.cla()
329 plb.pie(b, explode = e, labels=l, colors=c, radius=.75,
330         autopct='%1.2f%%', shadow=True, startangle=15)
331
332 # we set the aspect ratio to 'equal' so the pie is drawn in a circle
333 plb.axis('equal')
334 plb.xticks(()); plb.yticks(())
335 plb.show()
```



Advanced plotting

- Advanced plotting: *contour plot*

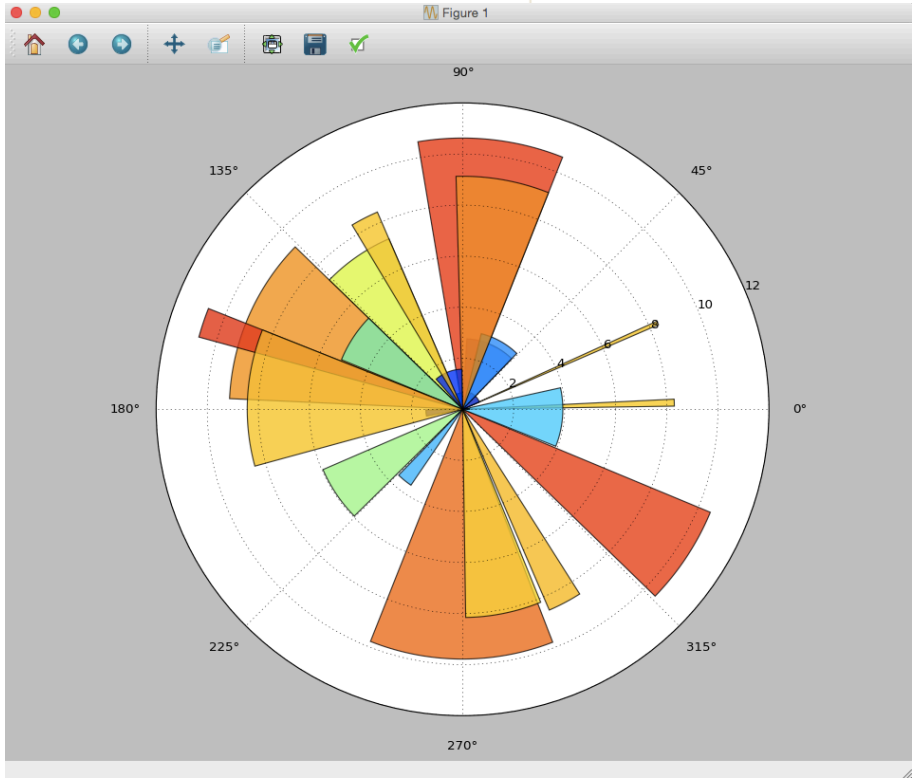
```
337 # Contour plot:
338 import pylab as plb
339
340 def f(x,y):
341     return (2 - x/3 + x**6 + 2.125*y) * plb.exp(-x**2 -y**2)
342
343 n = 128
344 x = plb.linspace(-2, 2, n)
345 y = plb.linspace(-1, 1, n)
346 A,B = plb.meshgrid(x, y)
347
348 plb.cla()
349 plb.axes([0.075, 0.075, 0.92, 0.92])
350
351 plb.contourf(A, B, f(A, B), 12, alpha=.50,
352     cmap=plb.cm.gist_rainbow)
353 C = plb.contour(A, B, f(A, B), 8, colors='black',
354     linewidth=.65)
355
356 plb.clabel(C, inline=1, fontsize=14)
357 plb.xticks(()); plb.yticks(())
358 plb.show()
```



Advanced plotting

- Advanced plotting: *polar plot*

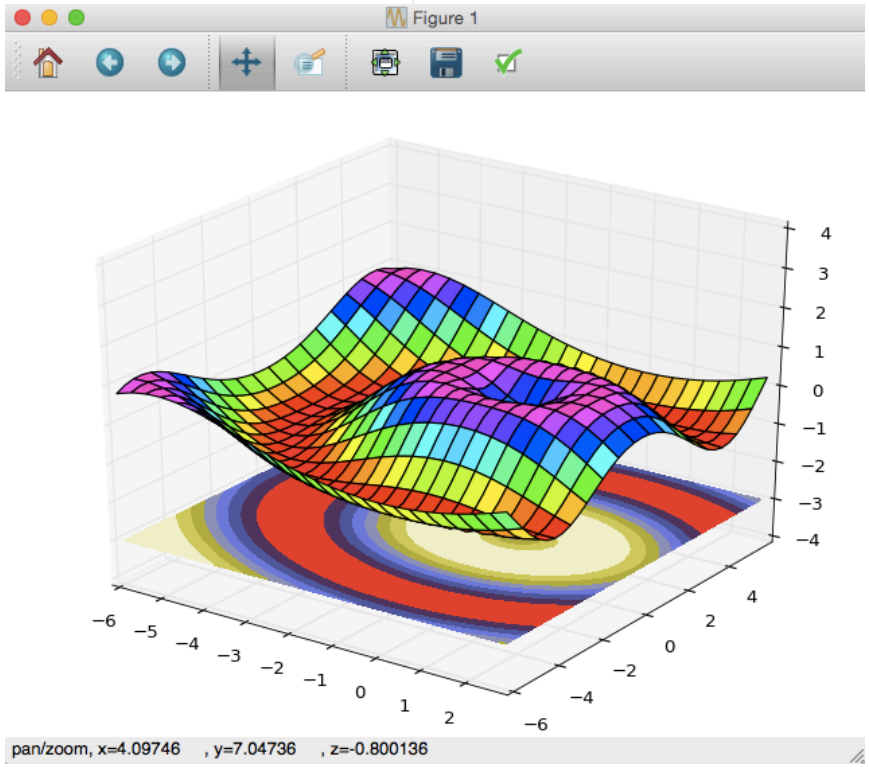
```
360  # Polar plot:
361  import pylab as plb
362
363  a = plb.axes([0.065, 0.065, 0.88, 0.88], polar=True)
364
365  q = 24
366  t = plb.arange(0.015, 3*plb.pi, 3 * plb.pi / q)
367  rad = 12 * plb.rand(q)
368  w = plb.pi / 4 * plb.rand(q)
369  ba = plb.bar(t, rad, width = w)
370
371  for r,bar in zip(rad, ba):
372      bar.set_facecolor(plb.cm.jet(r/12.))
373      bar.set_alpha(0.75)
374
375  plb.show()
```



Advanced plotting

- Advanced plotting: *3-D plot*

```
377 # 3-D plot:
378 import pylab as plb
379 from mpl_toolkits.mplot3d import Axes3D
380
381 ax = Axes3D(plb.figure())
382 x = plb.arange(-6, 3, 0.35)
383 y = plb.arange(-6, 6, 0.35)
384 x, y = plb.meshgrid(x, y)
385 k = plb.sqrt(x**2 + y**2)
386 z = plb.sin(k)
387
388 ax.plot_surface(x, y, z, rstride=2, cstride=1,
389                 cmap=plb.cm.gist_rainbow)
390 ax.contourf(x, y, z, zdir='z', offset=-3,
391                 cmap=plb.cm.gist_stern)
392 ax.set_zlim(-4, 4)
393
394 plb.show()
```



Course Content Outline

- **NumPy 2/3**
- Array operations
- Reductions
- Broadcasting
- Array: shaping, reshaping, flattening, resizing, dimension changing
- Data sorting — Midterm

- **NumPy 3/3**
- Type casting
- Masking data
- Organizing arrays
- Loading data files
- Dealing with polynomials
- Good coding practices — Project proposal due

- **Scipy 1/2**
- What is Scipy?
- Working with files
- Algebraic operations
- The Fast Fourier Transform
- Signal Processing — HW4

- **Scipy 2/2**
- Interpolation
- Statistics
- Optimization

- **Project**
- Project Presentation — Final Project

Array operations

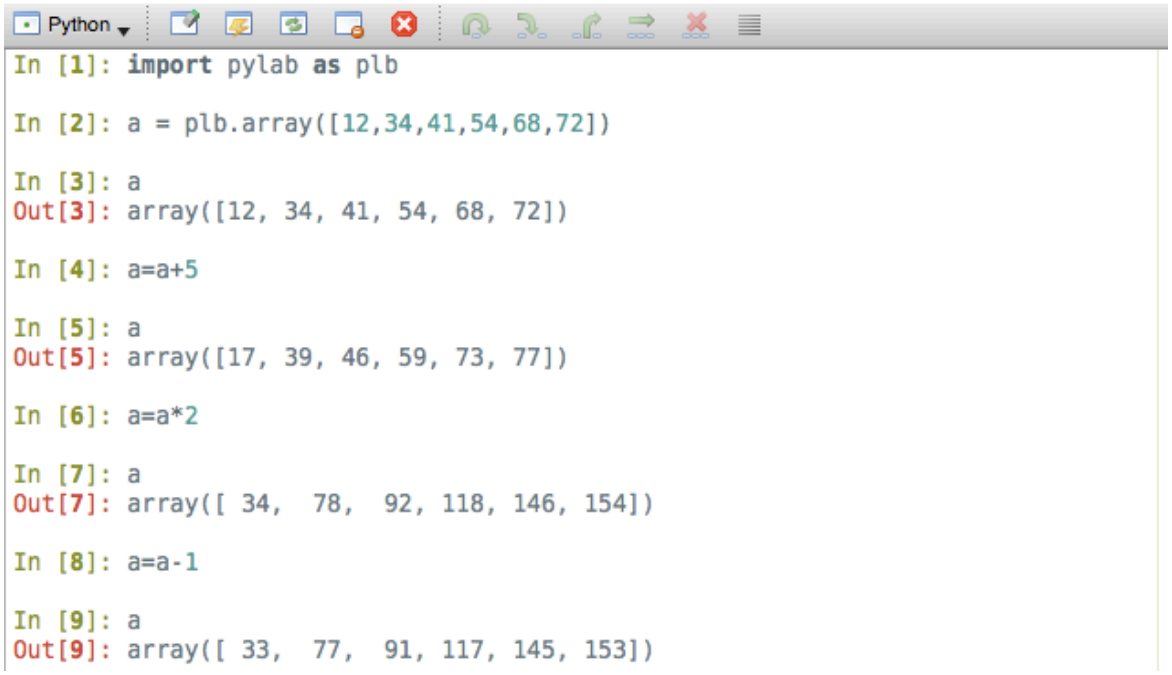
- Arrays operations

```
import numpy as np
        x = np.array([[1,2],[3,4]], dtype=np.float64)
        y = np.array([[5,6],[7,8]], dtype=np.float64)

# Elementwise sum; both produce the array # [[ 6.0 8.0] # [10.0 12.0]]
print(x + y)
print(np.add(x, y))

# Elementwise difference; both produce the array # [[-4.0 -4.0] # [-4.0 -4.0]]
print(x - y)
print(np.subtract(x, y))

# Elementwise product; both produce the array # [[ 5.0 12.0] # [21.0 32.0]]
print(x * y)
print(np.multiply(x, y))

# Elementwise division; both produce the array # [[ 0.2 0.33333333] # [ 0.42857143 0.5 ]]
print(x / y)
print(np.divide(x, y))

# Elementwise square root; produces the array # [[ 1. 1.41421356] # [ 1.73205081 2. ]]
print(np.sqrt(x))
```

Array operations

- Array operations
  - basic operations of arrays with scalars are very simple and intuitive
  - they performed on element by element basis

Examples:



Array operations

- Array operations
  - As in any language we need to be aware of NumPy's precedence:
    - ( ) – has the highest precedence order. The inner most bracket expression has the highest precedence
    - F (args...) – Function calls
    - x[ind:ind] – Slicing
    - x[index] – Subscription
    - x.attribute – Attribute reference
    - ** – power, exponentiation
    - ~x – bitwise not
    - +x, -x – positive and negative signs
    - *, /, % – multiplication, division, remainder
    - +, - – addition, subtraction
    - <<, >> – bitwise shift
    - & – bitwise and
    - ^ – bitwise xor (exclusive *or*, meaning both values must be mutually exclusive)
    - | – bitwise or
    - in, is, not in, is not, <, <=, >, >=, <>, !=, == – comparison, membership and identity tests
    - not x – boolean NOT
    - and – boolean AND
    - or – boolean OR

    conversions: bin(x) – int to bin, int('0011001', 2) – bin to int

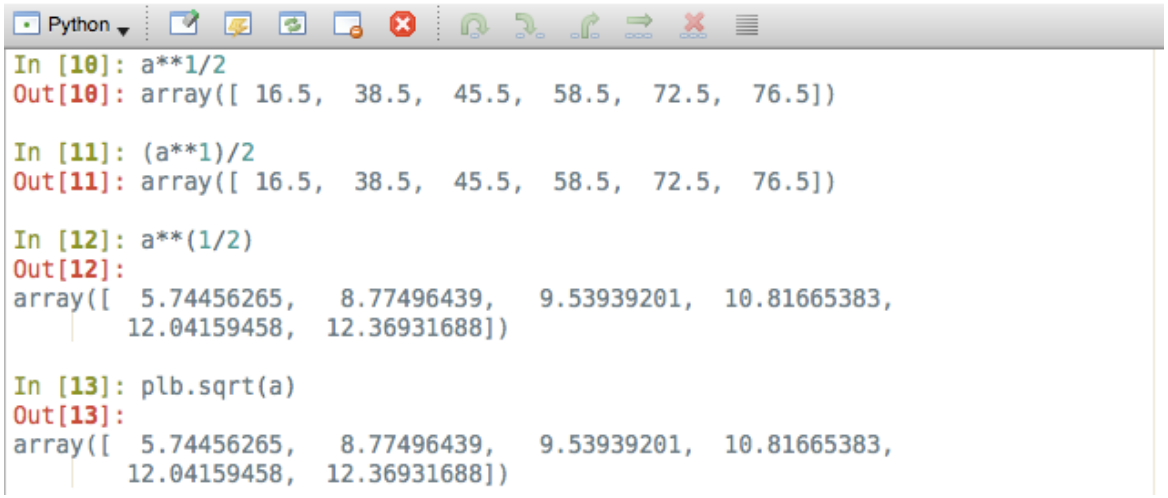
  Examples:

  e & f | z == (e & f) | z e | f & z == e | (f & z) , where z = e & f

Array operations

- Array operations
  - associativity:
    - (12 – 32) + 41 = 21 → left-associative
    - 12 – (32 + 41) = - 61 → right-associative

Examples:



Array operations

- Array operations
  - arithmetic operations between arrays are also performed element by element
  - NumPy operations are much faster than the ones used by basic math in Python

Examples:

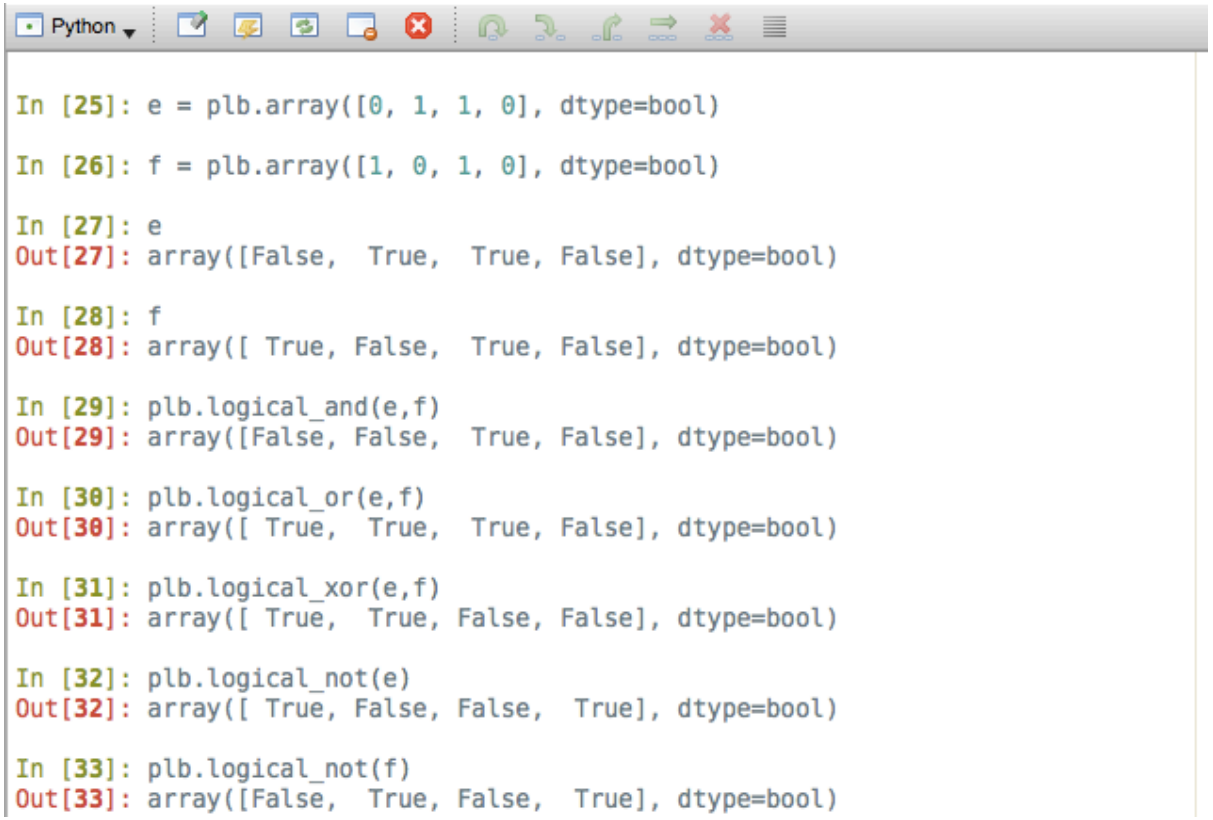
```
In [14]: b = plb.arange(6)+4

In [15]: b
Out[15]: array([4, 5, 6, 7, 8, 9])

In [16]: a-b
Out[16]: array([ 29,  72,  85, 110, 137, 144])

In [17]: a/b
Out[17]:
array([  8.25      ,  15.4       ,  15.16666667,  16.71428571,
        18.125     ,  17.        ])

In [18]: c = plb.ones(6)

In [19]: c=(a+b)**2-plb.sqrt(c)

In [20]: c
Out[20]: array([  1368.,   6723.,   9408.,  15375.,  23408.,  26243.])

In [21]: d = plb.arange(5)

In [22]: plb.size(a)
Out[22]: 6

In [23]: plb.size(d)
Out[23]: 5

In [24]: a-d
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-24-ba0eb01b3f36> in <module>()
----> 1 a-d

ValueError: operands could not be broadcast together with shapes (6,) (5,)
```

when performing operations between arrays they must -> be of the same shape

Array operations

- Array operations
  - logical operations are built in NumPy

Examples:



Array operations

- Array operations
  - arrays in NumPy can be compared in a element by element basis easily:

Examples:

```
In [34]: e
Out[34]: array([False,  True,  True, False], dtype=bool)

In [35]: f
Out[35]: array([ True, False,  True, False], dtype=bool)

In [36]: e == f
Out[36]: array([False, False,  True,  True], dtype=bool)

In [37]: e > f
Out[37]: array([False,  True, False, False], dtype=bool)

In [38]: e < f
Out[38]: array([ True, False, False, False], dtype=bool)
```

Array operations

- Array operations
  - arrays in NumPy can also be compared as whole vectors:

Examples:

```
Python
In [39]: g = e

In [40]: h = plb.array([0, 1, 1, 0], dtype=bool)

In [41]: e
Out[41]: array([False,  True,  True, False], dtype=bool)

In [42]: f
Out[42]: array([ True, False,  True, False], dtype=bool)

In [43]: g
Out[43]: array([False,  True,  True, False], dtype=bool)

In [44]: h
Out[44]: array([False,  True,  True, False], dtype=bool)

In [45]: plb.array_equal(e,f)
Out[45]: False

In [46]: plb.array_equal(e,g)
Out[46]: True

In [47]: plb.array_equiv(e,g)
Out[47]: True

In [48]: plb.array_equiv(g,h)
Out[48]: True

In [49]: id(e)
Out[49]: 4543434224

In [50]: id(g)
Out[50]: 4543434224

In [51]: id(h)
Out[51]: 4582111072
```

Array operations

- Array operations
  - one can take the *sin, cos, tan, log, exp* or *exp2* of an array easily in NumPy:

Examples:

```
Python

In [52]: i = plb.arange(5)-2

In [53]: i
Out[53]: array([-2, -1,  0,  1,  2])

In [54]: plb.sin(i)
Out[54]: array([-0.90929743, -0.84147098,  0.        ,  0.84147098,  0.90929743])

In [55]: plb.cos(i)
Out[55]: array([-0.41614684,  0.54030231,  1.        ,  0.54030231, -0.41614684])

In [56]: plb.tan(i)
Out[56]: array([ 2.18503986, -1.55740772,  0.        ,  1.55740772, -2.18503986])

In [57]: plb.log(i)
__main__:1: RuntimeWarning: divide by zero encountered in log
__main__:1: RuntimeWarning: invalid value encountered in log
Out[57]: array([        nan,         nan,        -inf,  0.        ,  0.69314718])

In [58]: plb.exp(i)
Out[58]: array([ 0.13533528,  0.36787944,  1.        ,  2.71828183,  7.3890561 ])

In [59]: plb.exp2(i)
Out[59]: array([ 0.25,  0.5 ,  1.  ,  2.  ,  4.  ])
```

be aware of your data to avoid this ->

Array operations

- Array operations
  - Transpose an array is in NumPy is performed like this:

Examples:

```
In [60]: j = plb.tril(plb.ones((5,5)),-2) # Lower triangle of an array

In [61]: j
Out[61]:
array([[ 0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.],
       [ 1.,  0.,  0.,  0.,  0.],
       [ 1.,  1.,  0.,  0.,  0.],
       [ 1.,  1.,  1.,  0.,  0.]])

In [62]: j.T
Out[62]:
array([[ 0.,  0.,  1.,  1.,  1.],
       [ 0.,  0.,  0.,  1.,  1.],
       [ 0.,  0.,  0.,  0.,  1.],
       [ 0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.]])

In [63]: k = plb.triu(plb.ones((5,5)),2) # Upper triangle of an array

In [64]: k
Out[64]:
array([[ 0.,  0.,  1.,  1.,  1.],
       [ 0.,  0.,  0.,  1.,  1.],
       [ 0.,  0.,  0.,  0.,  1.],
       [ 0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.]])

In [65]: plb.array_equal(j.T,k)
Out[65]: True
```

Reductions

- Reductions
  - reduction is an array operation in which a special set of array elements is produced based on a selection or mathematical operation
  - In NumPy there are several methods that process arrays as input and can create a new set of array elements as output based on specific requirements
  - one of the most basic methods for reduction is *sum*

Examples:

Reductions

- Reductions
  - *sum* can be performed on specific set of elements from an array such as rows and columns:

Examples:

```
Python

In [70]: n = plb.array([[12,34,41],[52,64,72]])

In [71]: n
Out[71]:
array([[12, 34, 41],
       [52, 64, 72]])

In [72]: n.sum()
Out[72]: 275

In [73]: n[:,:].sum()    # show total sum of all elements in 'n' (as above)
Out[73]: 275

In [74]: n[:,2].sum()    # shows the sum of column #3
Out[74]: 113

In [75]: n[1,:].sum()    # shows the sum of row #2
Out[75]: 188

In [76]: n.sum(axis=0)   # shows the sum of each column
Out[76]: array([ 64,  98, 113])

In [77]: n.sum(axis=1)   # shows the sum of each row
Out[77]: array([ 87, 188])

In [78]: n.cumsum()      # shows the cumulative sum in 'n'
Out[78]: array([ 12,  46,  87, 139, 203, 275])
```

Reductions

- Reductions
  - logical operations can be performed on rows and columns:

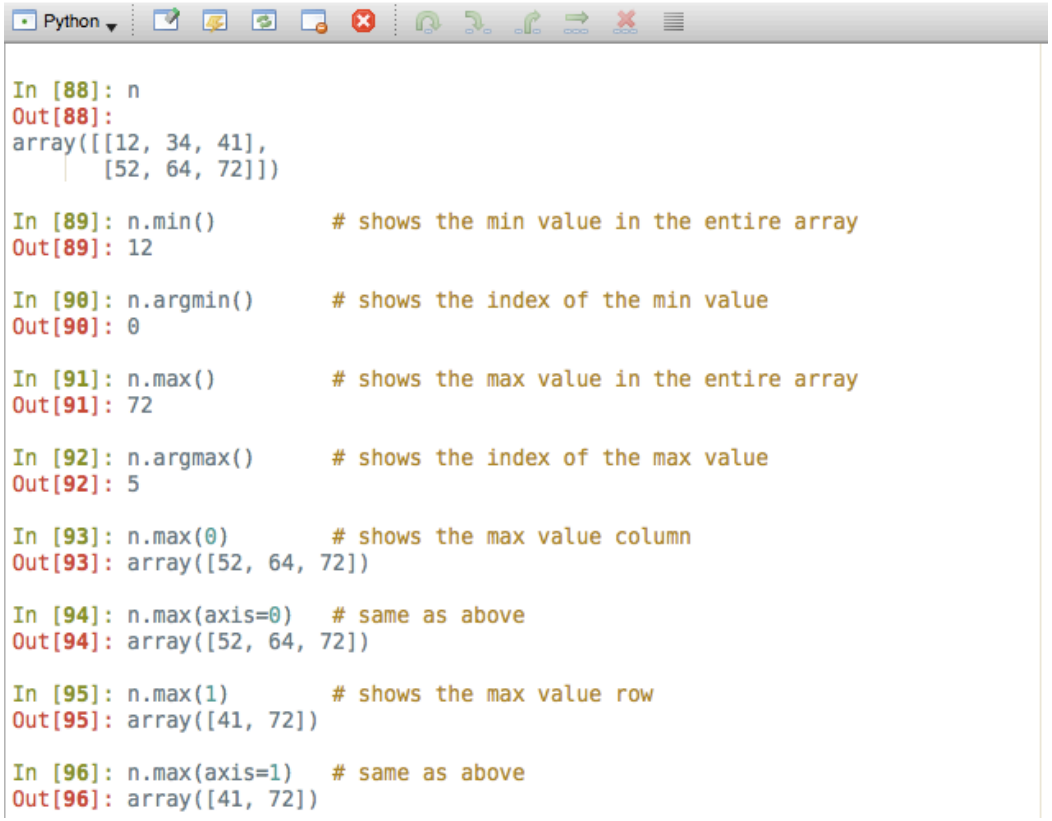
Examples:

```
Python

In [79]: n
Out[79]:
array([[12, 34, 41],
       [52, 64, 72]])

In [80]: p = plb.array([[12,34,41,0],[52,64,0,72]])

In [81]: p
Out[81]:
array([[12, 34, 41,  0],
       [52, 64,  0, 72]])

In [82]: plb.any(p != 0)     # show if there are any elements different than '0'
Out[82]: True

In [83]: plb.all(p != 0)     # show if all elements are different than '0'
Out[83]: False

In [84]: plb.any(p != n)     # show if any elements in 'p' and 'n' are different
Out[84]: True

In [85]: n!=p                # show if 'n' is different than 'p'
Out[85]: True

In [86]: n.sum()!=p.sum()    # show if total sum in 'n' is different than the one in 'p'
Out[86]: False

In [87]: n.sum()==p.sum()    # show if total sum in 'n' and 'p' is the same
Out[87]: True
```

Reductions

- Reductions
  - *min, max* extremes can be performed on specific set of elements from an array such as rows and columns:

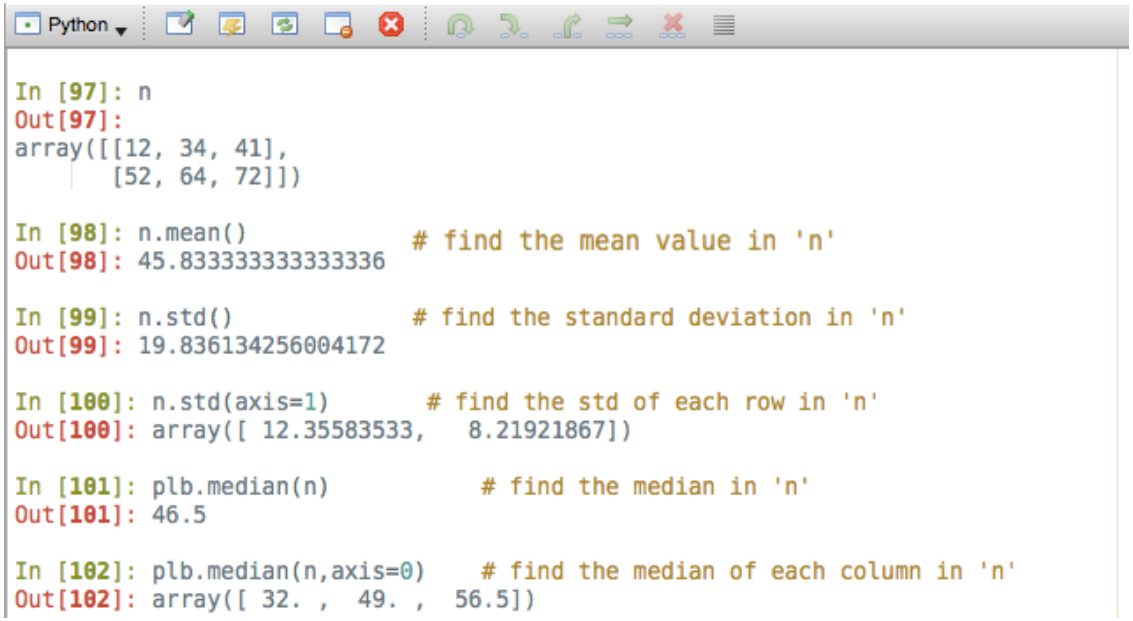
Examples:



Reductions

- Reductions
  - statistics can be performed on the arrays as well as on rows and columns:

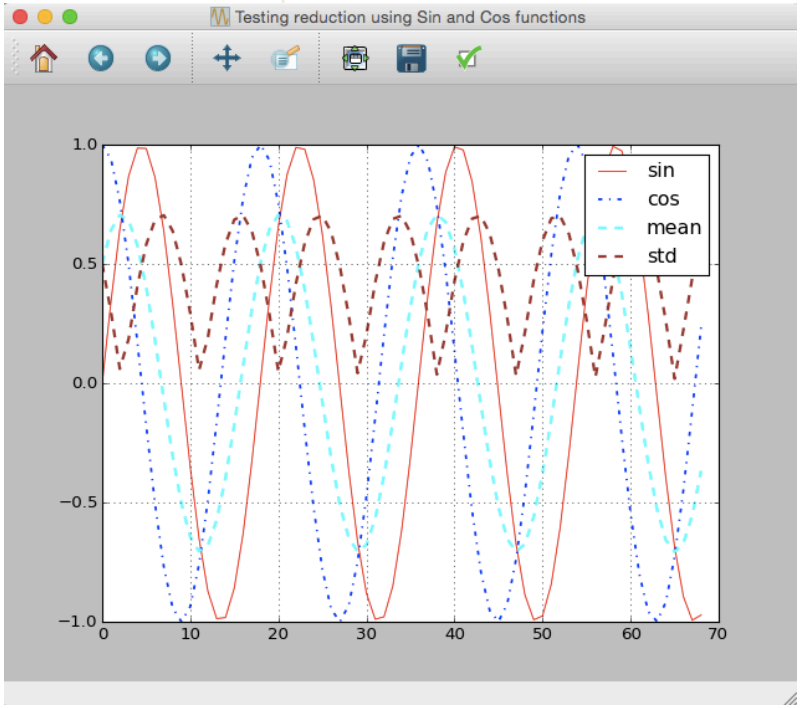
Examples:



Reductions

- Reductions
  - taking the mean and std values of *sin* vs *cos* functions:

```
134 # Example using reduction:
135 import pylab as plb
136
137 x0 = plb.arange(0, 24, 0.35)
138 x1 = plb.sin(x0)
139 x2 = plb.cos(x0)
140 fig = plb.mean([x1,x2],axis=0)
141 plb.figure('Testing reduction using Sin and Cos functions')
142 plb.plot(x1,color='r', linewidth=0.8, label='sin')
143 plb.plot(x2,color='b', linewidth=1.5, linestyle='-.',
144                    label='cos')
145
146 # plot the mean values of the sin and cos functions:
147 plb.plot(plb.mean([x1,x2],axis=0), color='cyan', linewidth=2,
148                    linestyle='--', label='mean')
149
150 # plot the std values of the sin and cos functions:
151 plb.plot(plb.std([x1,x2],axis=0), color='brown', linewidth=2,
152                    linestyle='--', label='std')
153 plb.legend(loc='upper right')
154 plb.grid(True)
155 plb.show()
```



Broadcasting

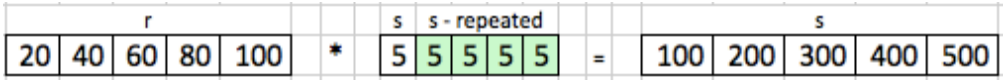
- Broadcasting
  - normally any operation on arrays will require them to be of the same size and shape, like in Matlab
  - in some cases where this rule is not met, NumPy has a way of overcoming that problem whenever possible
  - this is called Broadcasting
  - in a simple multiplication of arrays with scalars (which we have seen so far) this is exactly what NumPy is doing
  - in order to solve the problem NymPy automatically changes the shape of some arrays or scalars by repeating some values in order to complete the task

Broadcasting

- Broadcasting
  - in this example we multiply an array with a scalar and the latter changes its shape by repeating the same scalar to match the shape of the array
  - only after this rule is satisfied the multiplication can be performed

Example:

```
Python

In [156]: # Broacasting - multiplication example:

In [157]: r = plb.array([20,40,60,80,100])

In [158]: r
Out[158]: array([ 20,  40,  60,  80, 100])

In [159]: plb.shape(r)
Out[159]: (5,)

In [160]: s = 5

In [161]: s = r*s

In [162]: s
Out[162]: array([100, 200, 300, 400, 500])

In [163]: plb.shape(s)
Out[163]: (5,)
```



Broadcasting

- Broadcasting
  - in this example we add two arrays of different sizes and shapes
  - we notice that in order for the addition to be performed both shapes were changed

Example:

```
Python
In [164]: # Broacasting - addition example:

In [165]: r = plb.array([20,40,60,80,100]).reshape(5,1)

In [166]: r
Out[166]:
array([[ 20],
       [ 40],
       [ 60],
       [ 80],
       [100]])

In [167]: plb.shape(r)
Out[167]: (5, 1)

In [168]: s = plb.arange(2,4)

In [169]: s
Out[169]: array([2, 3])

In [170]: plb.shape(s)
Out[170]: (2,)

In [171]: t = r+s

In [172]: t
Out[172]:
array([[ 22,  23],
       [ 42,  43],
       [ 62,  63],
       [ 82,  83],
       [102, 103]])

In [173]: plb.shape(t)
Out[173]: (5, 2)
```

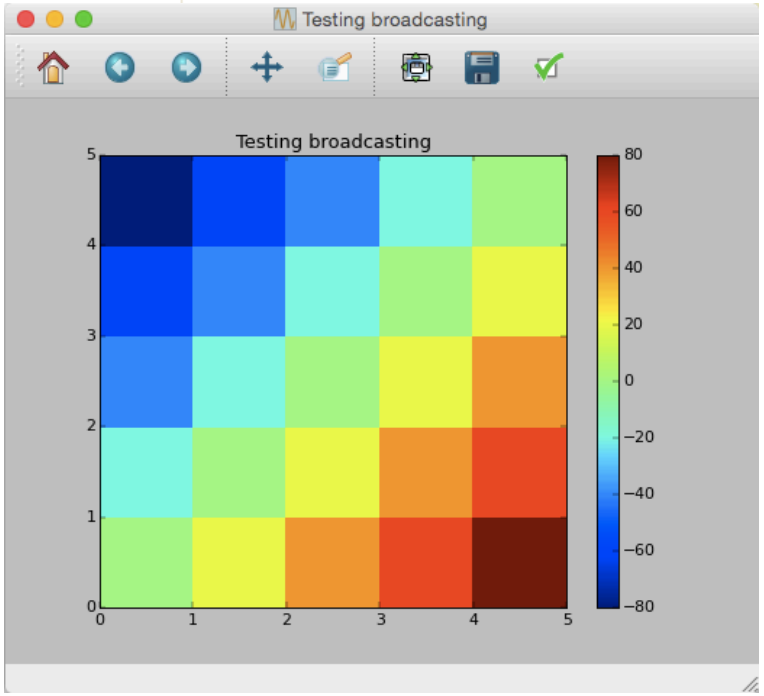
r	r - rep.							t	
20	20			2	3	s		22	23
40	40			2	3			42	43
60	60	+		2	3	s - repeated	=	62	63
80	80			2	3			82	83
100	100			2	3			102	103

Broadcasting

- Broadcasting

Example:

```
183  ## Broadcasting - example of a 2-D array or square matrix with diagonal = 0:
184  r = plb.array([20,40,60,80,100])
185  print(s)
186  r[:,plb.newaxis]    # change the shape of 'r' and add a new axis - 2-D array
187  s = r - r[:,plb.newaxis]    # create a square matrix with diagonal = 0
188  print(s)
189
190  plb.figure('Testing broadcasting', dpi=65)
191  plb.gca(title='Testing broadcasting')
192  plb.pcolormesh(s)    # chosing the printing scheme
193  plb.colorbar()       # adding a colorbar on the side for clarity
194  plb.pause(1)
```



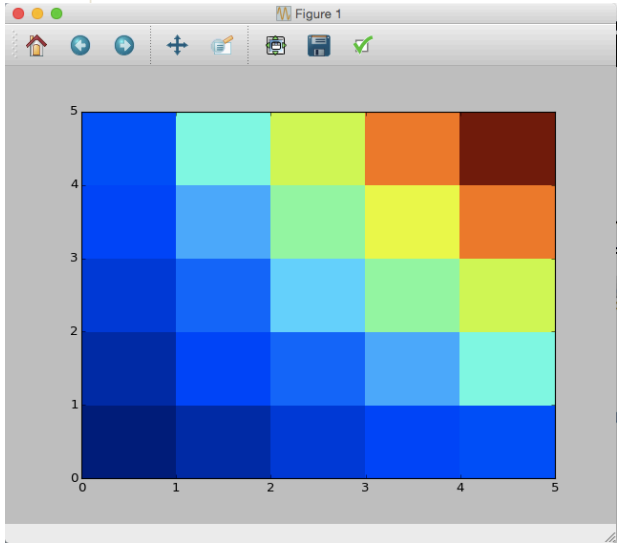
newaxis – expands the dimensions by one unit-length dimension

Broadcasting

- Broadcasting
  - the NymPy function *ogrid* assists us in making horizontal and vertical shape vector arrays in one line
  - we can also assign the x and y axis to different variables

Example:

```
In [174]: # Broadcasting - 'ogrid' example:

In [175]: plb.ogrid[1:6, 1:6]
Out[175]:
[array([[1],
        [2],
        [3],
        [4],
        [5]]), array([[1, 2, 3, 4, 5]])]

In [176]: u, v = plb.ogrid[1:6, 1:6]    # assign x,y axis to u,v variables

In [177]: u
Out[177]:
array([[1],
       [2],
       [3],
       [4],
       [5]])

In [178]: v
Out[178]: array([[1, 2, 3, 4, 5]])

In [179]: u*v
Out[179]:
array([[ 1,  2,  3,  4,  5],
       [ 2,  4,  6,  8, 10],
       [ 3,  6,  9, 12, 15],
       [ 4,  8, 12, 16, 20],
       [ 5, 10, 15, 20, 25]])

In [180]: plb.pcolormesh(u*v)
Out[180]: <matplotlib.collections.QuadMesh at 0x1059bc208>
```



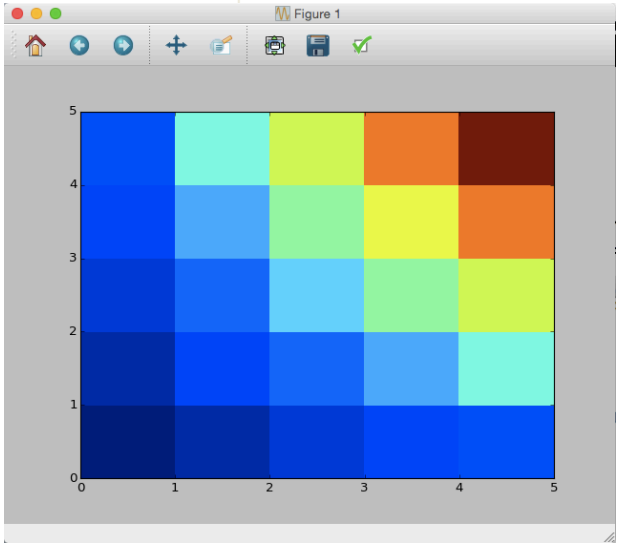
Broadcasting

- Broadcasting
  - NymPy also provides the function *mgrid*, designed specifically to provide matrices full of indices in case we can't take a full advantage from broadcasting

Example:

```
In [181]: # Broadcasting - 'mgrid' example:

In [182]: plb.mgrid[1:6, 1:6]
Out[182]:
array([[[1, 1, 1, 1, 1],
        [2, 2, 2, 2, 2],
        [3, 3, 3, 3, 3],
        [4, 4, 4, 4, 4],
        [5, 5, 5, 5, 5]],

       [[1, 2, 3, 4, 5],
        [1, 2, 3, 4, 5],
        [1, 2, 3, 4, 5],
        [1, 2, 3, 4, 5],
        [1, 2, 3, 4, 5]]])

In [183]: u, v = plb.mgrid[1:6, 1:6]

In [184]: u*v
Out[184]:
array([[ 1,  2,  3,  4,  5],
       [ 2,  4,  6,  8, 10],
       [ 3,  6,  9, 12, 15],
       [ 4,  8, 12, 16, 20],
       [ 5, 10, 15, 20, 25]])

In [185]: plb.pcolormesh(u*v)
Out[185]: <matplotlib.collections.QuadMesh at 0x105944780>
```

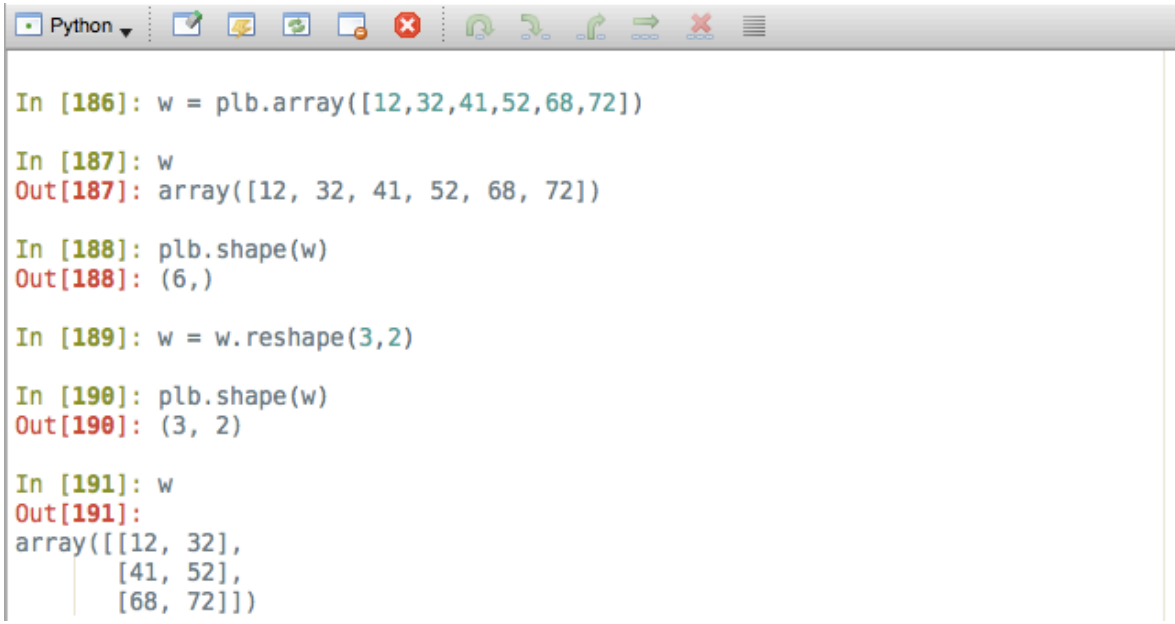
we get the same result



Arrays

- Array: shaping, reshaping, flattening, resizing, dimension changing
  - array manipulation in NumPy is supported by several built in functions such as:
    shape, reshape, ravel, resize, newaxis
  - there are several common techniques we will consider in the next few slides

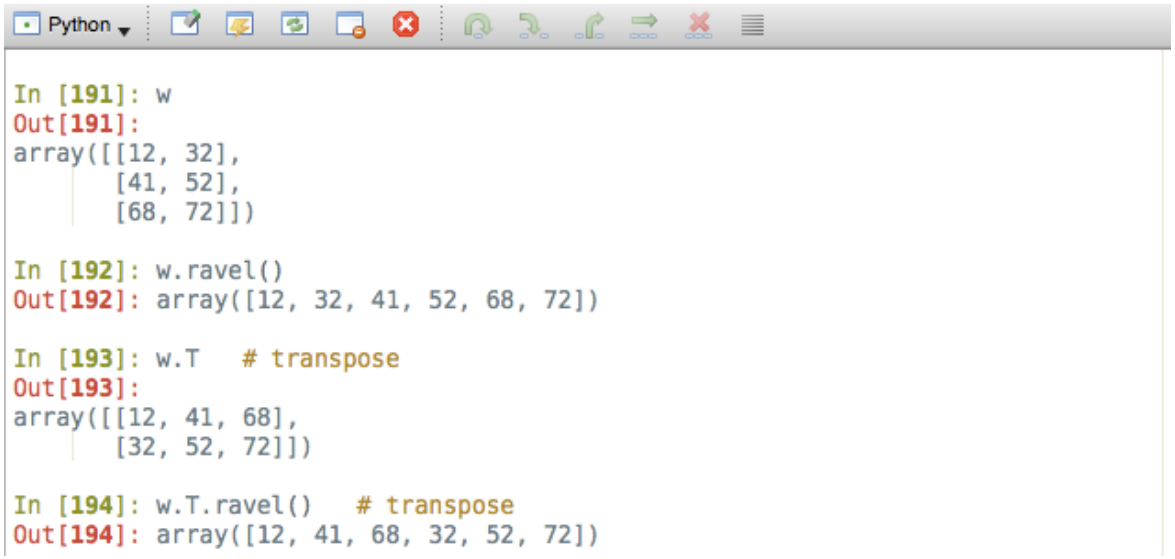
Using shape/reshape:
(we have already used them)



Arrays

- Array: shaping, reshaping, flattening, resizing, dimension changing
  - array manipulation in NumPy is supported by several built in functions such as:
    shape, reshape, ravel, resize, newaxis
  - there are several common techniques we will consider in the next few slides

Using ravel for flattening, hence representing a multidimensional array in a flat 1-D vector:



Arrays

- Array: shaping, reshaping, flattening, resizing, dimension changing
  - array manipulation in NumPy is supported by several built in functions such as:
    shape, reshape, ravel, resize, newaxis
  - there are several common techniques we will consider in the next few slides

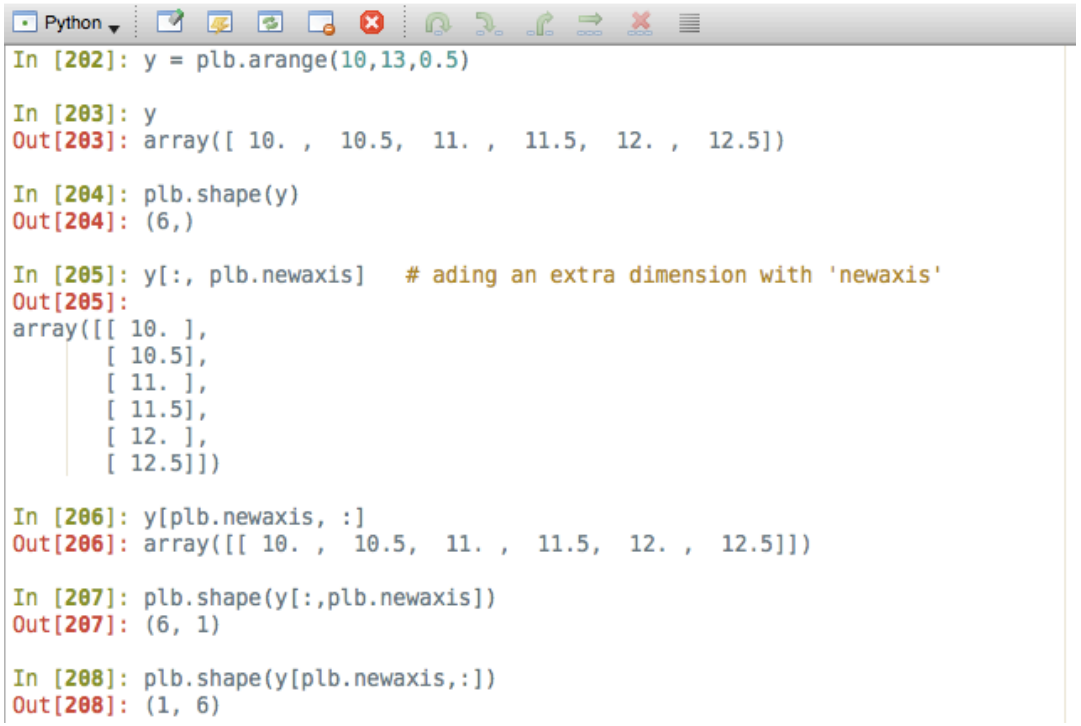
Using resize for resizing for existing array:

```
Python
In [195]: x = plb.arange(5,11,1)

In [196]: x
Out[196]: array([ 5,  6,  7,  8,  9, 10])

In [197]: x.resize(2,3)

In [198]: x
Out[198]:
array([[ 5,  6,  7],
       [ 8,  9, 10]])

In [199]: x.resize(8)
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-199-935ffac0ea7b> in <module>()
----> 1 x.resize(8)

ValueError: cannot resize an array that references or is referenced
by another array in this way.  Use the resize function

In [200]: x.resize(6)

In [201]: x
Out[201]: array([ 5,  6,  7,  8,  9, 10])
```

Arrays

- Array: shaping, reshaping, flattening, resizing, dimension changing
  - array manipulation in NumPy is supported by several built in functions such as:
    shape, reshape, ravel, resize, newaxis
  - there are several common techniques we will consider in the next few slides

Using newaxis for adding an extra dimension on an existing array:



Arrays

- Array: shaping, reshaping, flattening, resizing, dimension changing
  - array manipulation in NumPy is supported by several built in functions such as:
    shape, reshape, ravel, resize, newaxis
  - there are several common techniques we will consider in the next few slides

Changing and shifting dimensions on an existing array:

```
Python
In [209]: z = plb.arange(12,15,0.5)

In [210]: z
Out[210]: array([ 12. ,  12.5,  13. ,  13.5,  14. ,  14.5])

In [211]: plb.shape(z)
Out[211]: (6,)

In [212]: z = z.reshape(2,3)

In [213]: z
Out[213]:
array([[ 12. ,  12.5,  13. ],
       [ 13.5,  14. ,  14.5]])

In [214]: z[1,2]
Out[214]: 14.5

In [215]: z = z.T

In [216]: z
Out[216]:
array([[ 12. ,  13.5],
       [ 12.5,  14. ],
       [ 13. ,  14.5]])

In [217]: z[2,1]
Out[217]: 14.5
```

Data sorting

- Data sorting
  - array data can be sorted on rows and columns basis

Example:

```
In [218]: a = plb.rand(12)*10 # create a matrix with 12 random

In [219]:                     # elements bound between 0 and 10

In [220]: a = plb.round_(a).reshape(4,3)  # round all values and reshape the array

In [221]: a = a.astype(int)   # type cast all elements from 'float64' to 'int'

In [222]: a
Out[222]:
array([[ 3,  0,  6],
       [ 4,  6,  4],
       [10,  1,  9],
       [ 1,  6,  9]])

In [223]: a.sort(axis=0)  # sort each column (same as 'a.sort(0)')

In [224]: a
Out[224]:
array([[ 1,  0,  4],
       [ 3,  1,  6],
       [ 4,  6,  9],
       [10,  6,  9]])

In [225]: a.sort(1)       # sort each row (same as 'a.sort()' and 'a.sort(axis=1)')

In [226]: a
Out[226]:
array([[ 0,  1,  4],
       [ 1,  3,  6],
       [ 4,  6,  9],
       [ 6,  9, 10]])
```

Data sorting

- Data sorting
  - sorting indices can be returned without sorting the array by calling the argsort function

Example:

```
In [227]: a = plb.rand(12)*10 # create a matrix with 12 random

In [228]:                     # elements bound between 0 and 10

In [229]: a = plb.round_(a).reshape(4,3)  # round all values and reshape the array

In [230]: a = a.astype(int)   # type cast all elements from 'float64' to 'int'

In [231]: a
Out[231]:
array([[ 4,  1,  7],
       [ 4, 10,  1],
       [ 0,  7,  6],
       [ 1,  0,  3]])

In [232]: a.argsort()
Out[232]:
array([[1, 0, 2],
       [2, 0, 1],
       [0, 2, 1],
       [1, 0, 2]])

In [233]: a
Out[233]:
array([[ 4,  1,  7],
       [ 4, 10,  1],
       [ 0,  7,  6],
       [ 1,  0,  3]])
```