Numpy Labs

Array creation routines
Ones and zeros

In [1]:

```
import numpy as np
```

Create a new array of 2*2 integers, without initializing entries.

In [27]:

```
np.empty([2,2], int)
```

Out[27]:

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

Let X = np.array([1,2,3], [4,5,6], np.int32). Create a new array with the same shape and type as X.

In [32]:

```
X = np.array([[1,2,3], [4,5,6]], np.int32)
np.empty_like(X)
```

Out[32]:

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

Create a 3-D array with ones on the diagonal and zeros elsewhere.

In [33]:

```
np.eye(3)
```

Out[33]:

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

In [35]:

```
np.identity(3)
```

Out[35]:

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

Create a new array of 3*2 float numbers, filled with ones.

In [36]:

```
np.ones([3,2], float)
```

Out[36]:

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

Let x = np.arange(4, dtype=np.int64). Create an array of ones with the same shape and type as X.

In [59]:

```
x = np.arange(4, dtype=np.int64)
np.ones_like(x)
```

Out[59]:

```
array([1, 1, 1, 1], dtype=int64)
```

Create a new array of 3*2 float numbers, filled with zeros.

In [45]:

```
np.zeros((3,2), float)
```

Out[45]:

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

Let x = np.arange(4, dtype=np.int64). Create an array of zeros with the same shape and type as X.

In [58]:

```
x = np.arange(4, dtype=np.int64)
np.zeros_like(x)
```

Out[58]:

```
array([0, 0, 0, 0], dtype=int64)
```

Create a new array of 2*5 uints, filled with 6.

In [49]:

```
np.full((2, 5), 6, dtype=np.uint)
```

Out[49]:

```
array([[6, 6, 6, 6, 6],
       [6, 6, 6, 6, 6]], dtype=uint32)
```

In [50]:

```
np.ones([2, 5], dtype=np.uint) * 6
```

Out[50]:

```
array([[6, 6, 6, 6, 6],
       [6, 6, 6, 6, 6]], dtype=uint32)
```

Let x = np.arange(4, dtype=np.int64). Create an array of 6's with the same shape and type as X.

In [79]:

```
x = np.arange(4, dtype=np.int64)
np.full_like(x, 6)
```

Out[79]:

```
array([6, 6, 6, 6], dtype=int64)
```

In [81]:

```
np.ones_like(x) * 6
```

Out[81]:

```
array([6, 6, 6, 6], dtype=int64)
```

From existing data

Create an array of [1, 2, 3].

In [53]:

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

Out[53]:

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

Let x = [1, 2]. Convert it into an array.

In [60]:

```
x = [1,2]
np.asarray(x)
```

Out[60]:

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

Let X = np.array([[1, 2], [3, 4]]). Convert it into a matrix.

In [62]:

```
X = np.array([[1, 2], [3, 4]])
np.asmatrix(X)
```

Out[62]:

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

Let x = [1, 2]. Conver it into an array of `float`.

In [63]:

```
x = [1, 2]
np.asfarray(x)
```

Out[63]:

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

In [64]:

```
np.asarray(x, float)
```

Out[64]:

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

Let x = np.array([30]). Convert it into scalar of its single element, i.e. 30.

In [67]:

```
x = np.array([30])
np.asscalar(x)
```

Out[67]:

```
30
```

In [68]:

```
x[0]
```

Out[68]:

```
30
```

Let x = np.array([1, 2, 3]). Create a array copy of x, which has a different id from x.

In [76]:

```
x = np.array([1, 2, 3])
y = np.copy(x)
print id(x), x
print id(y), y
70140352 [1 2 3]
70140752 [1 2 3]
```

Numerical ranges

Create an array of 2, 4, 6, 8, ..., 100.

In [85]:

```
np.arange(2, 101, 2)
```

Out[85]:

```
array([  2,   4,   6,   8,  10,  12,  14,  16,  18,  20,  22,  24,  26
,
        28,  30,  32,  34,  36,  38,  40,  42,  44,  46,  48,  50,  52
,
        54,  56,  58,  60,  62,  64,  66,  68,  70,  72,  74,  76,  78
,
        80,  82,  84,  86,  88,  90,  92,  94,  96,  98, 100])
```

Create a 1-D array of 50 evenly spaced elements between 3. and 10., inclusive.

In [86]:

```
np.linspace(3., 10, 50)
```

Out[86]:

```
array([  3.        ,   3.14285714,   3.28571429,   3.42857143,
         3.57142857,   3.71428571,   3.85714286,   4.        ,
         4.14285714,   4.28571429,   4.42857143,   4.57142857,
         4.71428571,   4.85714286,   5.        ,   5.14285714,
         5.28571429,   5.42857143,   5.57142857,   5.71428571,
         5.85714286,   6.        ,   6.14285714,   6.28571429,
         6.42857143,   6.57142857,   6.71428571,   6.85714286,
         7.        ,   7.14285714,   7.28571429,   7.42857143,
         7.57142857,   7.71428571,   7.85714286,   8.        ,
         8.14285714,   8.28571429,   8.42857143,   8.57142857,
         8.71428571,   8.85714286,   9.        ,   9.14285714,
         9.28571429,   9.42857143,   9.57142857,   9.71428571,
         9.85714286,  10.        ])
```

Create a 1-D array of 50 element spaced evenly on a log scale between 3. and 10., exclusive.

In [88]:

```
np.logspace(3., 10., 50, endpoint=False)
```

Out[88]:

```
array([  1.00000000e+03,   1.38038426e+03,   1.90546072e+03,
         2.63026799e+03,   3.63078055e+03,   5.01187234e+03,
         6.91830971e+03,   9.54992586e+03,   1.31825674e+04,
         1.81970086e+04,   2.51188643e+04,   3.46736850e+04,
```

```
         4.78630092e+04,   6.60693448e+04,   9.12010839e+04,
         1.25892541e+05,   1.73780083e+05,   2.39883292e+05,
         3.31131121e+05,   4.57088190e+05,   6.30957344e+05,
         8.70963590e+05,   1.20226443e+06,   1.65958691e+06,
         2.29086765e+06,   3.16227766e+06,   4.36515832e+06,
         6.02559586e+06,   8.31763771e+06,   1.14815362e+07,
         1.58489319e+07,   2.18776162e+07,   3.01995172e+07,
         4.16869383e+07,   5.75439937e+07,   7.94328235e+07,
         1.09647820e+08,   1.51356125e+08,   2.08929613e+08,
         2.88403150e+08,   3.98107171e+08,   5.49540874e+08,
         7.58577575e+08,   1.04712855e+09,   1.44543977e+09,
         1.99526231e+09,   2.75422870e+09,   3.80189396e+09,
         5.24807460e+09,   7.24435960e+09])
```

Building matrices

Let X = np.array([[0, 1, 2, 3], [4, 5, 6, 7], [8, 9, 10, 11]]). Get the diagonal of X, that is, [0, 5, 10].

In [93]:

```
X = np.array([[ 0, 1, 2, 3], [ 4, 5, 6, 7], [ 8, 9, 10, 11]])
np.diag(X)
```

Out[93]:

```
array([ 0,  5, 10])
```

In [94]:

```
X.diagonal()
```

Out[94]:

```
array([ 0,  5, 10])
```

Create a 2-D array whose diagonal equals [1, 2, 3, 4] and 0's elsewhere.

In [95]:

```
np.diagflat([1, 2, 3, 4])
```

Out[95]:

```
array([[1, 0, 0, 0],
       [0, 2, 0, 0],
       [0, 0, 3, 0],
       [0, 0, 0, 4]])
```

Create an array which looks like below. array([[0., 0., 0., 0., 0.], [1., 0., 0., 0., 0.], [1., 1., 0., 0., 0.]])

In [97]:

```
np.tri(3, 5, -1)
```

Out[97]:

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

Create an array which looks like below. array([[0, 0, 0], [4, 0, 0], [7, 8, 0], [10, 11, 12]])

In [101]:

```
np.tril(np.arange(1, 13).reshape(4, 3), -1)
```

Out[101]:

```
array([[ 0,  0,  0],
       [ 4,  0,  0],
       [ 7,  8,  0],
       [10, 11, 12]])
```

Create an array which looks like below. array([[1, 2, 3], [4, 5, 6], [0, 8, 9], [0, 0, 12]])

In [102]:

```
np.triu(np.arange(1, 13).reshape(4, 3), -1)
```

Out[102]:

```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 0,  8,  9],
       [ 0,  0, 12]])
```

Array manipulation routines

In [1]:

```
import numpy as np
```

In [2]:

```
np.__version__
```

Out[2]:

```
'1.11.2'
```

Q1. Let x be a ndarray [10, 10, 3] with all elements set to one. Reshape x so that the size of the second dimension equals 150.

In [5]:

```
x = np.ones([10, 10, 3])
out = np.reshape(x, [-1, 150])
print out
assert np.allclose(out, np.ones([10, 10, 3]).reshape([-1, 150]))
[[ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.
1.
   1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.
1.
   1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.
1.
   1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.
1.
   1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.
1.
   1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.
1.
   1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.
1.
```

```
   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.
1.
   1.   1.   1.   1.   1.   1.]
 [ 1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.
1.
   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.
1.
   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.
1.
   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.
1.
   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.
1.
   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.
1.
   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.
1.
   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.   1.
1.
   1.   1.   1.   1.   1.   1.]]
```

Q2. Let x be array [[1, 2, 3], [4, 5, 6]]. Convert it to [1 4 2 5 3 6].

In [22]:

```
x = np.array([[1, 2, 3], [4, 5, 6]])
out1 = np.ravel(x, order='F')
out2 = x.flatten(order="F")
assert np.allclose(out1, out2)
print out1
```

```
[1 4 2 5 3 6]
```

Q3. Let x be array [[1, 2, 3], [4, 5, 6]]. Get the 5th element.

In [23]:

```
x = np.array([[1, 2, 3], [4, 5, 6]])
out1 = x.flat[4]
out2 = np.ravel(x)[4]
assert np.allclose(out1, out2)
print out1
```

```
5
```

Q4. Let x be an arbitrary 3-D array of shape (3, 4, 5). Permute the dimensions of x such that the new shape will be (4,3,5).

In [36]:

```
x = np.zeros((3, 4, 5))
out1 = np.swapaxes(x, 1, 0)
out2 = x.transpose([1, 0, 2])
assert out1.shape == out2.shape
print out1.shape
```

```
(4L, 3L, 5L)
```

Q5. Let x be an arbitrary 2-D array of shape (3, 4). Permute the dimensions of x such that the new shape will be (4,3).

In [38]:

```
x = np.zeros((3, 4))
out1 = np.swapaxes(x, 1, 0)
out2 = x.transpose()
out3 = x.T
assert out1.shape == out2.shape == out3.shape
print out1.shape
```

```
(4L, 3L)
```

Q5. Let x be an arbitrary 2-D array of shape (3, 4). Insert a nex axis such that the new shape will be (3, 1, 4).

In [42]:

```
x = np.zeros((3, 4))
print np.expand_dims(x, axis=1).shape
```

```
(3L, 1L, 4L)
```

Q6. Let x be an arbitrary 3-D array of shape (3, 4, 1). Remove a single-dimensional entries such that the new shape will be (3, 4).

In [43]:

```
x = np.zeros((3, 4, 1))
print np.squeeze(x).shape
```

```
(3L, 4L)
```

Q7. Lex x be an array
[[1 2 3]
[4 5 6].

and y be an array
[[7 8 9]
[10 11 12]].
Concatenate x and y so that a new array looks like
[[1, 2, 3, 7, 8, 9],
[4, 5, 6, 10, 11, 12]].

In [31]:

```
x = np.array([[1, 2, 3], [4, 5, 6]])
y = np.array([[7, 8, 9], [10, 11, 12]])
out1 = np.concatenate((x, y), 1)
out2 = np.hstack((x, y))
assert np.allclose(out1, out2)
print out2
```

```
[[ 1  2  3  7  8  9]
 [ 4  5  6 10 11 12]]
```

Q8. Lex x be an array
[[1 2 3]
[4 5 6].

and y be an array
[[7 8 9]
[10 11 12]].
Concatenate x and y so that a new array looks like
[[1 2 3]
[4 5 6]

[7 8 9]
[10 11 12]]

In [38]:

```
x = np.array([[1, 2, 3], [4, 5, 6]])
y = np.array([[7, 8, 9], [10, 11, 12]])
out1 = np.concatenate((x, y), 0)
out2 = np.vstack((x, y))
assert np.allclose(out1, out2)
print out2
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
```

Q8. Let x be an array [1 2 3] and y be [4 5 6]. Convert it to [[1, 4], [2, 5], [3, 6]].

In [9]:

```
x = np.array((1,2,3))
y = np.array((4,5,6))
out1 = np.column_stack((x, y))
out2 = np.squeeze(np.dstack((x, y)))
out3 = np.vstack((x, y)).T
assert np.allclose(out1, out2)
assert np.allclose(out2, out3)
print out1
```

Out[9]:

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

Q9. Let x be an array [[1],[2],[3]] and y be [[4], [5], [6]]. Convert x to [[[1, 4]], [[2, 5]], [[3, 6]]].

In [34]:

```
x = np.array([[1],[2],[3]])
y = np.array([[4],[5],[6]])
out = np.dstack((x, y))
print out
[[[1 4]]

 [[2 5]]

 [[3 6]]]
```

Q10. Let x be an array [1, 2, 3, ..., 9]. Split x into 3 arrays, each of which has 4, 2, and 3 elements in the original order.

In [62]:

```
x = np.arange(1, 10)
print np.split(x, [4, 6])
[array([1, 2, 3, 4]), array([5, 6]), array([7, 8, 9])]
```

Q11. Let x be an array
[[[0., 1., 2., 3.],
[4., 5., 6., 7.]],
[[8., 9., 10., 11.],
[12., 13., 14., 15.]]].
Split it into two such that the first array looks like
[[[0., 1., 2.],
[4., 5., 6.]],
[[8., 9., 10.],
[12., 13., 14.]]].
and the second one look like:
[[[3.],
[7.]],
[[11.],
[15.]]].

In [72]:

```
x = np.arange(16).reshape(2, 2, 4)
out1 = np.split(x, [3],axis=2)
out2 = np.dsplit(x, [3])
assert np.allclose(out1[0], out2[0])
assert np.allclose(out1[1], out2[1])
print out1
[array([[[ 0,  1,  2],
        [ 4,  5,  6]],

       [[ 8,  9, 10],
        [12, 13, 14]]]), array([[[ 3],
        [ 7]],

       [[11],
        [15]]])]
```

Q12. Let x be an array
[[0., 1., 2., 3.],
[4., 5., 6., 7.],
[8., 9., 10., 11.],
[12., 13., 14., 15.]].
Split it into two arrays along the second axis.

In [74]:

```
x = np.arange(16).reshape((4, 4))
out1 = np.hsplit(x, 2)
out2 = np.split(x, 2, 1)
assert np.allclose(out1[0], out2[0])
assert np.allclose(out1[1], out2[1])
print out1
[array([[ 0,  1],
       [ 4,  5],
       [ 8,  9],
       [12, 13]]), array([[ 2,  3],
       [ 6,  7],
       [10, 11],
```

```
        [14, 15]])]
```

Q13. Let x be an array [[0., 1., 2., 3.], [4., 5., 6., 7.], [8., 9., 10., 11.], [12., 13., 14., 15.]]. Split it into two arrays along the first axis.

In [75]:

```
x = np.arange(16).reshape((4, 4))
out1 = np.vsplit(x, 2)
out2 = np.split(x, 2, 0)
assert np.allclose(out1[0], out2[0])
assert np.allclose(out1[1], out2[1])
print out1
[array([[0, 1, 2, 3],
       [4, 5, 6, 7]]), array([[ 8,  9, 10, 11],
       [12, 13, 14, 15]])]
```

Q14. Let x be an array [0, 1, 2]. Convert it to [[0, 1, 2, 0, 1, 2], [0, 1, 2, 0, 1, 2]].

In [93]:

```
x = np.array([0, 1, 2])
out1 = np.tile(x, [2, 2])
out2 = np.resize(x, [2, 6])
assert np.allclose(out1, out2)
print out1
[[0 1 2 0 1 2]
 [0 1 2 0 1 2]]
```

Q15. Let x be an array [0, 1, 2]. Convert it to [0, 0, 1, 1, 2, 2].

In [83]:

```
x = np.array([0, 1, 2])
print np.repeat(x, 2)
[0 0 1 1 2 2]
```

Q16. Let x be an array [0, 0, 0, 1, 2, 3, 0, 2, 1, 0]. remove the leading the trailing zeros.

In [105]:

```
x = np.array((0, 0, 0, 1, 2, 3, 0, 2, 1, 0))
out = np.trim_zeros(x)
print out
[1 2 3 0 2 1]
```

Q17. Let x be an array [2, 2, 1, 5, 4, 5, 1, 2, 3]. Get two arrays of unique elements and their counts.

In [107]:

```
x = np.array([2, 2, 1, 5, 4, 5, 1, 2, 3])
u, indices = np.unique(x, return_counts=True)
print u, indices
```

```
[1 2 3 4 5] [2 3 1 1 2]
```

Q18. Lex x be an array
[[1 2]
[3 4]].
Flip x along the second axis.

In [120]:

```
x = np.array([[1,2], [3,4]])
out1 = np.fliplr(x)
out2 = x[:, ::-1]
assert np.allclose(out1, out2)
print out1
[[2 1]
 [4 3]]
```

Q19. Lex x be an array
[[1 2]
[3 4]].
Flip x along the first axis.

In [121]:

```
x = np.array([[1,2], [3,4]])
out1 = np.flipud(x)
out2 = x[::-1, :]
assert np.allclose(out1, out2)
print out1
[[3 4]
 [1 2]]
```

Q20. Lex x be an array
[[1 2]
[3 4]].
Rotate x 90 degrees counter-clockwise.

In [122]:

```
x = np.array([[1,2], [3,4]])
out = np.rot90(x)
print out
[[2 4]
 [1 3]]
```

Q21 Lex x be an array
[[1 2 3 4]
[5 6 7 8]].
Shift elements one step to right along the second axis.

In [126]:

```
x = np.arange(1, 9).reshape([2, 4])
print np.roll(x, 1, axis=1)
[[4 1 2 3]
 [8 5 6 7]]
```

String operations

In [1]:

```
from __future__ import print_function
import numpy as np
```

In [3]:

```
np.__version__
```

Out[3]:

```
'1.11.3'
```

Q1. Concatenate x1 and x2.

In [4]:

```
x1 = np.array(['Hello', 'Say'], dtype=np.str)
x2 = np.array([' world', ' something'], dtype=np.str)
out = np.char.add(x1, x2)
print(out)
['Hello world' 'Say something']
```

Q2. Repeat x three time element-wise.

In [5]:

```
x = np.array(['Hello ', 'Say '], dtype=np.str)
out = np.char.multiply(x, 3)
print(out)
['Hello Hello Hello ' 'Say Say Say ']
```

Q3-1. Capitalize the first letter of x element-wise.
Q3-2. Lowercase x element-wise.
Q3-3. Uppercase x element-wise.
Q3-4. Swapcase x element-wise.
Q3-5. Title-case x element-wise.

In [6]:

```
x = np.array(['heLLo woRLd', 'Say sOmething'], dtype=np.str)
capitalized = np.char.capitalize(x)
lowered = np.char.lower(x)
uppered = np.char.upper(x)
swapcased = np.char.swapcase(x)
titlecased = np.char.title(x)
print("capitalized =", capitalized)
print("lowered =", lowered)
print("uppered =", uppered)
print("swapcased =", swapcased)
print("titlecased =", titlecased)
capitalized = ['Hello world' 'Say something']
lowered = ['hello world' 'say something']
uppered = ['HELLO WORLD' 'SAY SOMETHING']
swapcased = ['HEllO WOrlD' 'sAY SoMETHING']
titlecased = ['Hello World' 'Say Something']
```

Q4. Make the length of each element 20 and the string centered / left-justified / right-justified with paddings of _.

In [7]:

```
x = np.array(['hello world', 'say something'], dtype=np.str)
centered = np.char.center(x, 20, fillchar='_')
```

```
left = np.char.ljust(x, 20, fillchar='_')
right = np.char.rjust(x, 20, fillchar='_')

print("centered =", centered)
print("left =", left)
print("right =", right)
```

```
centered = ['____hello world_____' '___say something____']
left = ['hello world_________' 'say something_______']
right = ['_________hello world' '_______say something']
```

Q5. Encode x in cp500 and decode it again.

In [8]:

```
x = np.array(['hello world', 'say something'], dtype=np.str)
encoded = np.char.encode(x, 'cp500')
decoded = np.char.decode(encoded,'cp500')
print("encoded =", encoded)
print("decoded =", decoded)
```

```
encoded = [b'\x88\x85\x93\x93\x96@\xa6\x96\x99\x93\x84'
 b'\xa2\x81\xa8@\xa2\x96\x94\x85\xa3\x88\x89\x95\x87']
decoded = ['hello world' 'say something']
```

Q6. Insert a space between characters of x.

In [9]:

```
x = np.array(['hello world', 'say something'], dtype=np.str)
out = np.char.join(" ", x)
print(out)
```

```
['h e l l o   w o r l d' 's a y   s o m e t h i n g']
```

Q7-1. Remove the leading and trailing whitespaces of x element-wise.
Q7-2. Remove the leading whitespaces of x element-wise.
Q7-3. Remove the trailing whitespaces of x element-wise.

In [10]:

```
x = np.array(['   hello world   ', '\tsay something\n'], dtype=np.str)
stripped = np.char.strip(x)
lstripped = np.char.lstrip(x)
rstripped = np.char.rstrip(x)
print("stripped =", stripped)
print("lstripped =", lstripped)
print("rstripped =", rstripped)
```

```
stripped = ['hello world' 'say something']
lstripped = ['hello world   ' 'say something\n']
rstripped = ['   hello world' '\tsay something']
```

Q8. Split the element of x with spaces.

In [11]:

```
x = np.array(['Hello my name is John'], dtype=np.str)
out = np.char.split(x)
print(out)
```

```
[['Hello', 'my', 'name', 'is', 'John']]
```

Q9. Split the element of x to multiple lines.

In [12]:

```
x = np.array(['Hello\nmy name is John'], dtype=np.str)
out = np.char.splitlines(x)
print(out)
```

```
[['Hello', 'my name is John']]
```

Q10. Make x a numeric string of 4 digits with zeros on its left.

In [13]:

```
x = np.array(['34'], dtype=np.str)
out = np.char.zfill(x, 4)
print(out)
```

```
['0034']
```

Q11. Replace "John" with "Jim" in x.

In [14]:

```
x = np.array(['Hello nmy name is John'], dtype=np.str)
out = np.char.replace(x, "John", "Jim")
print(out)
```

```
['Hello nmy name is Jim']
```

Comparison

Q12. Return x1 == x2, element-wise.

In [15]:

```
x1 = np.array(['Hello', 'my', 'name', 'is', 'John'], dtype=np.str)
x2 = np.array(['Hello', 'my', 'name', 'is', 'Jim'], dtype=np.str)
out = np.char.equal(x1, x2)
print(out)
```

```
[ True  True  True  True False]
```

Q13. Return x1 != x2, element-wise.

In [16]:

```
x1 = np.array(['Hello', 'my', 'name', 'is', 'John'], dtype=np.str)
x2 = np.array(['Hello', 'my', 'name', 'is', 'Jim'], dtype=np.str)
out = np.char.not_equal(x1, x2)
print(out)
```

```
[False False False False  True]
```

String information

Q14. Count the number of "l" in x, element-wise.

In [17]:

```
x = np.array(['Hello', 'my', 'name', 'is', 'Lily'], dtype=np.str)
out = np.char.count(x, "l")
print(out)
```

```
[2 0 0 0 1]
```

Q15. Count the lowest index of "l" in x, element-wise.

In [18]:

```
x = np.array(['Hello', 'my', 'name', 'is', 'Lily'], dtype=np.str)
out = np.char.find(x, "l")
print(out)

# compare
```

```
# print(np.char.index(x, "l"))
# => This raises an error!
[ 2 -1 -1 -1  2]
```

Q16-1. Check if each element of x is composed of digits only.
Q16-2. Check if each element of x is composed of lower case letters only.
Q16-3. Check if each element of x is composed of upper case letters only.

In [19]:

```
x = np.array(['Hello', 'I', 'am', '20', 'years', 'old'], dtype=np.str)
out1 = np.char.isdigit(x)
out2 = np.char.islower(x)
out3 = np.char.isupper(x)
print("Digits only =", out1)
print("Lower cases only =", out2)
print("Upper cases only =", out3)
Digits only = [False False False  True False False]
Lower cases only = [False False  True False  True  True]
Upper cases only = [False  True False False False False]
```

Q17. Check if each element of x starts with "hi".

In [20]:

```
x = np.array(['he', 'his', 'him', 'his'], dtype=np.str)
out = np.char.startswith(x, "hi")
print(out)
[False  True  True  True]
```

Input and Output

In [1]:

```
from __future__ import print_function
import numpy as np
```

In [15]:

```
np.__version__
```

Out[15]:

```
'1.12.0'
```

In [16]:

```
from datetime import date
print(date.today())
2017-04-01
```

NumPy binary files (NPY, NPZ)

Q1. Save x into `temp.npy` and load it.

In [4]:

```
x = np.arange(10)
np.save('temp.npy', x) # Actually you can omit the extension. If so, i
t will be added automatically.
```

```python
# Check if there exists the 'temp.npy' file.
import os
if os.path.exists('temp.npy'):
    x2 = np.load('temp.npy')
    print(np.array_equal(x, x2))
```

```
True
```

Q2. Save x and y into a single file 'temp.npz' and load it.

In [5]:

```python
x = np.arange(10)
y = np.arange(11, 20)
np.savez('temp.npz', x=x, y=y)
# np.savez_compressed('temp.npz', x=x, y=y) # If you want to save x an
d y into a single file in compressed .npz format.
with np.load('temp.npz') as data:
    x2 = data['x']
    y2 = data['y']
    print(np.array_equal(x, x2))
    print(np.array_equal(y, y2))
```

```
True
True
```

Text files

Q3. Save x to 'temp.txt' in string format and load it.

In [6]:

```python
x = np.arange(10).reshape(2, 5)
header = 'num1 num2 num3 num4 num5'
np.savetxt('temp.txt', x, fmt="%d", header=header)
np.loadtxt('temp.txt')
```

Out[6]:

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

Q4. Save `x`, `y`, and `z` to 'temp.txt' in string format line by line, then load it.

In [7]:

```python
x = np.arange(10)
y = np.arange(11, 21)
z = np.arange(22, 32)
np.savetxt('temp.txt', (x, y, z), fmt='%d')
np.loadtxt('temp.txt')
```

Out[7]:

```
array([[  0.,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.],
       [ 11.,  12.,  13.,  14.,  15.,  16.,  17.,  18.,  19.,  20.],
       [ 22.,  23.,  24.,  25.,  26.,  27.,  28.,  29.,  30.,  31.]])
```

Q5. Convert `x` into bytes, and load it as array.

In [8]:

```python
x = np.array([1, 2, 3, 4])
```

```
x_bytes = x.tostring() # Don't be misled by the function name. What it really does is it returns bytes.
x2 = np.fromstring(x_bytes, dtype=x.dtype) # returns a 1-D array even if x is not.
print(np.array_equal(x, x2))
```

```
True
```

Q6. Convert `a` into an ndarray and then convert it into a list again.

In [9]:

```
a = [[1, 2], [3, 4]]
x = np.array(a)
a2 = x.tolist()
print(a == a2)
```

```
True
```

String formatting¶

Q7. Convert `x` to a string, and revert it.

In [10]:

```
x = np.arange(10).reshape(2,5)
x_str = np.array_str(x)
print(x_str, "\n", type(x_str))
x_str = x_str.replace("[", "") # [] must be stripped
x_str = x_str.replace("]", "")
x2 = np.fromstring(x_str, dtype=x.dtype, sep=" ").reshape(x.shape)
assert np.array_equal(x, x2)
```

```
[[0 1 2 3 4]
 [5 6 7 8 9]]
 <class 'str'>
```

Text formatting options

Q8. Print `x` such that all elements are displayed with precision=1, no suppress.

In [11]:

```
x = np.random.uniform(size=[10,100])
np.set_printoptions(precision=1, threshold=np.nan, suppress=True)
print(x)
```

```
[[ 0.5  0.   0.8  0.2  0.3  0.2  0.2  1.   0.4  0.8  0.6  0.2  0.5  0.
1
   0.4  0.1  0.9  0.6  0.1  0.5  0.8  0.8  0.8  0.   0.6  0.8  0.4  0.
3
   0.8  0.2  0.7  0.7  0.2  1.   0.8  0.1  0.2  0.1  0.3  0.1  0.5  0.
9
   0.6  0.9  0.6  0.5  0.8  0.3  0.3  0.5  0.1  0.6  0.1  0.3  0.6  0.
2
   0.4  0.8  0.6  0.4  0.2  0.6  0.   0.3  0.8  0.5  0.7  0.9  0.8  0.
6
   0.9  0.8  0.4  0.4  0.7  0.8  0.   0.1  0.5  0.4  0.7  1.   0.1  0.
2
   0.6  0.3  0.9  0.1  0.6  0.4  0.3  0.8  0.3  0.6  0.6  0.3  1.   0.
2
```

```
   0.9  0.2]
 [ 0.9  0.2  0.4  0.9  0.5  0.6  0.1  0.7  0.   0.   0.1  0.8  0.8  1.
0.2
   0.8  0.3  0.2  1.   0.6  1.   0.3  0.4  0.4  0.7  0.5  0.4  0.8  0.
5
   0.9  0.3  0.5  0.7  0.4  0.2  0.3  0.9  0.   0.6  0.8  0.3  0.5  0.
2
   0.3  0.   0.6  0.5  0.2  0.5  0.8  0.2  0.8  0.   0.9  0.   0.7  0.
1
   0.4  0.2  0.5  0.6  0.2  0.6  0.1  0.1  0.   0.5  0.9  0.4  0.5  0.
8
   0.5  0.1  0.7  0.   1.   0.5  0.4  0.2  0.   1.   0.4  0.1  0.7  0.
7
   0.4  0.8  0.4  0.6  0.6  0.5  0.8  0.8  0.2  0.2  0.3  0.2  0.5  0.
9
   0.5]
 [ 0.3  0.6  0.4  0.5  0.5  0.   0.7  0.1  0.   0.9  0.5  0.7  0.6  0.
3
   0.9  0.5  0.1  0.4  0.1  0.9  0.8  0.6  0.8  0.8  0.1  0.4  0.9  0.
1  1.
   0.7  0.4  0.3  0.8  0.3  0.8  0.8  0.2  0.7  0.2  0.8  0.3  0.9  0.
1
   0.9  0.2  0.8  0.9  0.6  0.1  0.3  0.4  1.   0.1  0.7  0.3  0.9  0.
3
   0.5  0.9  0.   0.6  0.   0.8  0.1  0.9  0.   0.8  0.6  0.5  0.5  0.
2  1.
   0.4  0.   0.2  0.   0.9  0.9  0.8  0.2  0.7  0.3  0.2  0.1  1.   0.
4
   0.5  0.4  0.8  0.8  0.8  0.7  0.6  0.4  0.7  0.6  0.5  0.8  0.7  0.
6]
 [ 0.2  0.6  0.9  0.7  0.1  0.1  1.   0.5  0.8  0.3  1.   0.4  0.1  0.
5
   0.6  0.8  0.8  0.8  0.1  1.   0.8  0.   0.7  0.6  0.8  0.2  0.5  0.
9
   0.4  0.8  0.7  0.2  0.8  0.6  0.9  0.6  0.9  0.8  0.9  1.   0.6  0.
6
   0.7  0.1  0.5  0.3  0.   0.8  0.   0.5  0.8  0.3  0.8  0.7  0.1  0.
5
   0.2  0.1  0.7  0.   0.   0.6  0.   0.8  0.7  0.1  0.4  0.1  0.2  0.
1
   0.9  0.6  0.9  0.3  0.4  0.9  0.2  0.6  0.8  0.9  0.6  0.8  0.5  0.
1
   0.6  1.   0.   0.7  0.7  0.4  0.1  0.9  0.4  0.1  0.7  0.6  0.3  0.
9
   0.3  0.5]
 [ 0.9  0.3  0.1  0.1  0.2  0.4  0.3  0.5  0.2  0.   0.5  0.4  0.5  0.
3
```

```
   0.6  1.   0.1  0.7  0.6  0.2  0.3  0.3  0.1  0.5  0.6  0.   0.6  0.
7
   0.6  0.4  0.2  0.6  0.1  0.9  0.9  0.1  0.9  0.1  0.6  0.6  0.   0.
1
   0.6  0.4  0.3  0.1  0.9  0.8  0.1  0.2  0.8  0.4  0.7  0.8  0.6  0.
4
   0.9  0.3  0.6  0.7  0.4  0.8  0.3  0.   0.   0.9  0.3  0.3  0.8  0.
5
   0.8  1.   0.2  0.6  0.6  0.2  0.2  0.2  0.4  0.6  0.6  0.4  0.4  0.
8
   0.2  0.5  0.7  0.7  0.1  0.9  0.5  0.6  0.3  0.3  0.6  0.8  0.6  0.
8
   0.4  0.3]
 [ 0.3  1.   0.6  0.9  0.6  1.   0.7  0.9  0.4  0.3  0.9  0.9  0.3  0.
8
   0.3  0.6  0.7  0.3  0.1  0.1  0.4  0.3  0.6  0.5  0.1  0.6  0.1  0.
5
   0.9  0.5  0.5  0.6  0.4  0.4  0.3  1.   0.6  0.6  0.3  0.1  0.4  0.
7
   0.7  0.1  0.5  0.1  0.3  0.1  0.6  0.7  0.   0.1  0.2  0.4  0.1  0.
4
   0.7  0.3  0.2  0.9  0.5  0.   0.4  0.9  1.   0.4  0.   0.2  0.3  0.
9
   0.3  0.   0.8  0.9  0.8  0.6  0.4  0.5  0.   0.9  0.6  0.6  0.1  0.
6
   0.9  0.1  0.8  0.6  0.6  0.5  0.7  1.   0.5  0.3  0.3  0.4  0.6  0.
6  1.
   0.2]
 [ 0.7  0.7  0.9  0.2  0.6  0.3  0.9  0.2  0.9  0.8  0.5  0.3  0.9  0.
5  1.
   0.6  0.9  0.5  0.5  0.1  0.8  0.3  0.9  0.5  0.7  1.   0.6  0.7  0.
1
   0.7  0.9  0.4  0.8  0.9  0.4  1.   0.1  1.   0.5  0.1  0.4  0.7  1.
0.4
   0.3  0.2  0.2  0.6  0.6  0.3  0.7  0.5  0.7  0.1  0.3  0.5  1.   0.
8
   0.4  0.8  0.8  0.7  0.1  0.2  0.4  0.3  0.4  0.3  0.5  0.4  0.6  0.
3
   0.1  0.7  0.8  0.6  0.6  0.2  0.7  0.9  0.9  0.7  0.3  0.9  0.4  0.
6  0.
   0.4  0.4  0.2  0.8  0.3  0.1  0.2  0.6  0.5  0.9  0.8  0.9  0.7]
 [ 0.8  0.7  0.7  0.6  0.9  0.1  0.4  0.9  1.   0.3  0.   0.2  0.1  0.
5
   0.8  0.1  0.7  0.7  0.6  1.   0.7  1.   0.4  0.6  0.2  0.4  0.4  0.
6  0.
   0.1  1.   0.5  0.1  0.2  0.8  0.2  0.1  0.4  0.7  0.5  0.4  1.   0.
5
```

```
   0.5  0.4  0.8  0.2  0.1  0.7  0.2  0.1  0.4  0.3  0.6  0.9  0.9  0.
9
   0.9  0.1  0.1  0.   1.   0.   0.1  0.4  0.6  1.   0.4  0.9  0.3  0.
2
   0.7  0.   0.3  0.2  0.7  0.4  0.3  0.9  0.3  0.   0.5  0.2  0.3  0.
1
   0.2  0.   0.1  0.6  0.9  0.2  0.5  0.8  0.7  0.   0.4  0.8  0.8  0.
5
   0.2]
 [ 0.2  0.3  0.   0.1  0.8  0.4  0.1  0.2  0.   0.7  0.   1.   0.6  0.
7
   0.3  0.3  0.7  0.9  0.3  0.7  0.1  0.1  0.5  0.6  0.3  0.8  0.7  0.
1
   0.6  0.6  0.3  0.2  0.3  0.3  1.   0.1  0.1  0.2  0.4  0.4  0.6  0.
5
   0.7  0.7  0.2  0.   0.8  0.3  0.9  0.1  0.1  0.4  0.4  0.5  0.3  0.
9
   0.6  0.9  0.3  0.5  0.   0.4  0.8  1.   0.3  0.5  0.7  0.5  0.8  0.
7
   0.6  0.3  0.1  0.2  0.5  1.   0.9  0.5  0.6  0.6  0.2  0.8  0.6  0.
0.5
   0.6  0.8  0.5  0.8  0.8  0.9  0.7  0.9  0.5  0.2  1.   1.   0.1  0.
3
   0.3]
 [ 0.   0.3  0.4  0.7  0.2  0.9  0.2  0.3  0.6  0.8  0.4  0.7  0.3  0.
5
   0.6  0.3  0.7  0.   0.1  0.1  0.9  0.   0.7  0.7  0.1  0.6  0.6  0.
0.3
   0.5  0.9  0.3  0.1  0.3  0.1  0.9  0.6  0.3  0.3  0.4  0.4  0.2  0.
3
   0.1  0.5  0.3  0.8  0.   0.8  0.6  0.2  0.7  0.4  0.8  0.2  0.9  1.
1.
   0.7  0.9  0.1  0.2  0.   0.5  0.8  0.7  0.6  0.7  0.7  0.5  0.9  0.
2
   0.2  0.1  0.2  0.1  0.7  1.   0.6  0.3  0.9  1.   0.3  0.3  0.7  0.
9
   0.5  0.8  0.9  0.7  0.2  0.7  0.3  0.1  0.9  0.2  0.5  0.6  0.3  0.
4]]
```

Base-n representations

Q9. Convert 12 into a binary number in string format.

In [12]:

```python
out1 = np.binary_repr(12)
out2 = np.base_repr(12, base=2)
assert out1 == out2 # But out1 is better because it's much faster.
print(out1)
```

```
1100
```

Q10. Convert 12 into a hexadecimal number in string format.

In [13]:

```
np.base_repr(1100, base=16)
```

Out[13]:

```
'44C'
```

Linear algebra

In [1]:

```
import numpy as np
```

In [2]:

```
np.__version__
```

Out[2]:

```
'1.11.2'
```

Matrix and vector products

Q1. Predict the results of the following code.

In [61]:

```
x = [1,2]
y = [[4, 1], [2, 2]]
print np.dot(x, y)
print np.dot(y, x)
print np.matmul(x, y)
print np.inner(x, y)
print np.inner(y, x)
[8 5]
[6 6]
[8 5]
[6 6]
[6 6]
```

Q2. Predict the results of the following code.

In [62]:

```
x = [[1, 0], [0, 1]]
y = [[4, 1], [2, 2], [1, 1]]
print np.dot(y, x)
print np.matmul(y, x)
[[4 1]
 [2 2]
 [1 1]]
[[4 1]
 [2 2]
 [1 1]]
```

Q3. Predict the results of the following code.

In [63]:

```
x = np.array([[1, 4], [5, 6]])
```

```
y = np.array([[4, 1], [2, 2]])
print np.vdot(x, y)
print np.vdot(y, x)
print np.dot(x.flatten(), y.flatten())
print np.inner(x.flatten(), y.flatten())
print (x*y).sum()
30
30
30
30
30
```

Q4. Predict the results of the following code.

In [65]:

```
x = np.array(['a', 'b'], dtype=object)
y = np.array([1, 2])
print np.inner(x, y)
print np.inner(y, x)
print np.outer(x, y)
print np.outer(y, x)
abb
abb
[['a' 'aa']
 ['b' 'bb']]
[['a' 'b']
 ['aa' 'bb']]
```

Decompositions

Q5. Get the lower-trianglular `L` in the Cholesky decomposition of x and verify it.

In [97]:

```
x = np.array([[4, 12, -16], [12, 37, -43], [-16, -43, 98]], dtype=np.i
nt32)
L = np.linalg.cholesky(x)
print L
assert np.array_equal(np.dot(L, L.T.conjugate()), x)
[[ 2.  0.  0.]
 [ 6.  1.  0.]
 [-8.  5.  3.]]
```

Q6. Compute the qr factorization of x and verify it.

In [107]:

```
x = np.array([[12, -51, 4], [6, 167, -68], [-4, 24, -41]], dtype=np.fl
oat32)
q, r = np.linalg.qr(x)
print "q=\n", q, "\nr=\n", r
assert np.allclose(np.dot(q, r), x)
q=
[[-0.85714287  0.39428571  0.33142856]
 [-0.42857143 -0.90285712 -0.03428571]
```

```
 [ 0.2857143  -0.17142858  0.94285715]]
r=
[[ -14.  -21.   14.]
 [   0. -175.   70.]
 [   0.    0.  -35.]]
```

Q7. Factor x by Singular Value Decomposition and verify it.

In [165]:

```
x = np.array([[1, 0, 0, 0, 2], [0, 0, 3, 0, 0], [0, 0, 0, 0, 0], [0, 2
, 0, 0, 0]], dtype=np.float32)
U, s, V = np.linalg.svd(x, full_matrices=False)
print "U=\n", U, "\ns=\n", s, "\nV=\n", v
assert np.allclose(np.dot(U, np.dot(np.diag(s), V)), x)
U=
[[ 0.  1.  0.  0.]
 [ 1.  0.  0.  0.]
 [ 0.  0.  0. -1.]
 [ 0.  0.  1.  0.]]
s=
[ 3.          2.23606801  2.          0.        ]
V=
[[ 1.  0.  0.]
 [ 0.  1.  0.]
 [ 0.  0.  1.]]
```

Matrix eigenvalues

Q8. Compute the eigenvalues and right eigenvectors of x. (Name them eigenvals and eigenvecs, respectively)

In [68]:

```
x = np.diag((1, 2, 3))
eigenvals = np.linalg.eig(x)[0]
eigenvals_ = np.linalg.eigvals(x)
assert np.array_equal(eigenvals, eigenvals_)
print "eigenvalues are\n", eigenvals
eigenvecs = np.linalg.eig(x)[1]
print "eigenvectors are\n", eigenvecs
eigenvalues are
[ 1.  2.  3.]
eigenvectors are
[[ 1.  0.  0.]
 [ 0.  1.  0.]
 [ 0.  0.  1.]]
```

Q9. Predict the results of the following code.

In [69]:

```
print np.array_equal(np.dot(x, eigenvecs), eigenvals * eigenvecs)
True
```

Norms and other numbers

Q10. Calculate the Frobenius norm and the condition number of x.

In [12]:

```
x = np.arange(1, 10).reshape((3, 3))
print np.linalg.norm(x, 'fro')
print np.linalg.cond(x, 'fro')
```

```
16.8819430161
4.56177073661e+17
```

Q11. Calculate the determinant of x.

In [22]:

```
x = np.arange(1, 5).reshape((2, 2))
out1 = np.linalg.det(x)
out2 = x[0, 0] * x[1, 1] - x[0, 1] * x[1, 0]
assert np.allclose(out1, out2)
print out1
```

```
-2.0
```

Q12. Calculate the rank of x.

In [35]:

```
x = np.eye(4)
out1 = np.linalg.matrix_rank(x)
out2 = np.linalg.svd(x)[1].size
assert out1 == out2
print out1
```

```
4
```

Q13. Compute the sign and natural logarithm of the determinant of x.

In [49]:

```
x = np.arange(1, 5).reshape((2, 2))
sign, logdet = np.linalg.slogdet(x)
det = np.linalg.det(x)
assert sign == np.sign(det)
assert logdet == np.log(np.abs(det))
print sign, logdet
```

```
-1.0 0.69314718056
```

Q14. Return the sum along the diagonal of x.

In [57]:

```
x = np.eye(4)
out1 = np.trace(x)
out2 = x.diagonal().sum()
assert out1 == out2
print out1
```

```
4.0
```

Solving equations and inverting matrices

Q15. Compute the inverse of x.

In [60]:

```
x = np.array([[1., 2.], [3., 4.]])
out1 = np.linalg.inv(x)
assert np.allclose(np.dot(x, out1), np.eye(2))
```

```
print out1
[[-2.   1. ]
 [ 1.5 -0.5]]
```

In [1]:

```
from __future__ import print_function
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:

```
from datetime import date
date.today()
```

Out[2]:

```
datetime.date(2017, 11, 2)
```

In [4]:

```
np.__version__
```

Out[4]:

```
'1.13.1'
```

Complex Numbers

Q1. Return the angle of `a` in radian.

In [5]:

```
a = 1+1j
output = np.angle(a, deg=False)
print(output)
0.785398163397
```

Q2. Return the real part and imaginary part of `a`.

In [6]:

```
a = np.array([1+2j, 3+4j, 5+6j])
real = a.real
imag = a.imag
print("real part=", real)
print("imaginary part=", imag)
real part= [ 1.  3.  5.]
imaginary part= [ 2.  4.  6.]
```

Q3. Replace the real part of a with `9`, the imaginary part with `[5, 7, 9]`.

In [7]:

```
a = np.array([1+2j, 3+4j, 5+6j])
a.real = 9
a.imag = [5, 7, 9]
print(a)
[ 9.+5.j  9.+7.j  9.+9.j]
```

Q4. Return the complex conjugate of `a`.

In [8]:

```
a = 1+2j
output = np.conjugate(a)
print(output)
```

```
(1-2j)
```

Discrete Fourier Transform

Q5. Compuete the one-dimensional DFT of `a`.

In [9]:

```
a = np.exp(2j * np.pi * np.arange(8))
output = np.fft.fft(a)
print(output)
```

```
[  8.00000000e+00 -6.85802208e-15j   2.36524713e-15 +9.79717439e-16j
   9.79717439e-16 +9.79717439e-16j   4.05812251e-16 +9.79717439e-16j
   0.00000000e+00 +9.79717439e-16j  -4.05812251e-16 +9.79717439e-16j
  -9.79717439e-16 +9.79717439e-16j  -2.36524713e-15 +9.79717439e-16j]
```

Q6. Compute the one-dimensional inverse DFT of the `output` in the above question.

In [10]:

```
print("a=", a)
inversed = np.fft.ifft(output)
print("inversed=", a)
```

```
a= [ 1. +0.00000000e+00j  1. -2.44929360e-16j  1. -4.89858720e-16j
  1. -7.34788079e-16j  1. -9.79717439e-16j  1. -1.22464680e-15j
  1. -1.46957616e-15j  1. -1.71450552e-15j]
inversed= [ 1. +0.00000000e+00j  1. -2.44929360e-16j  1. -4.89858720e-16j
  1. -7.34788079e-16j  1. -9.79717439e-16j  1. -1.22464680e-15j
  1. -1.46957616e-15j  1. -1.71450552e-15j]
```

Q7. Compute the one-dimensional discrete Fourier Transform for real input `a`.

In [11]:

```
a = [0, 1, 0, 0]
output = np.fft.rfft(a)
print(output)
assert output.size==len(a)//2+1 if len(a)%2==0 else (len(a)+1)//2

# cf.
output2 = np.fft.fft(a)
print(output2)
```

```
[ 1.+0.j  0.-1.j -1.+0.j]
[ 1.+0.j  0.-1.j -1.+0.j  0.+1.j]
```

Q8. Compute the one-dimensional inverse DFT of the output in the above question.

In [12]:

```
inversed = np.fft.ifft(output)
print("inversed=", a)
```

```
inversed= [0, 1, 0, 0]
```

Q9. Return the DFT sample frequencies of `a`.

In [13]:

```
signal = np.array([-2, 8, 6, 4, 1, 0, 3, 5], dtype=np.float32)
fourier = np.fft.fft(signal)
n = signal.size
freq = np.fft.fftfreq(n, d=1)
print(freq)
[ 0.     0.125  0.25   0.375 -0.5   -0.375 -0.25  -0.125]
```

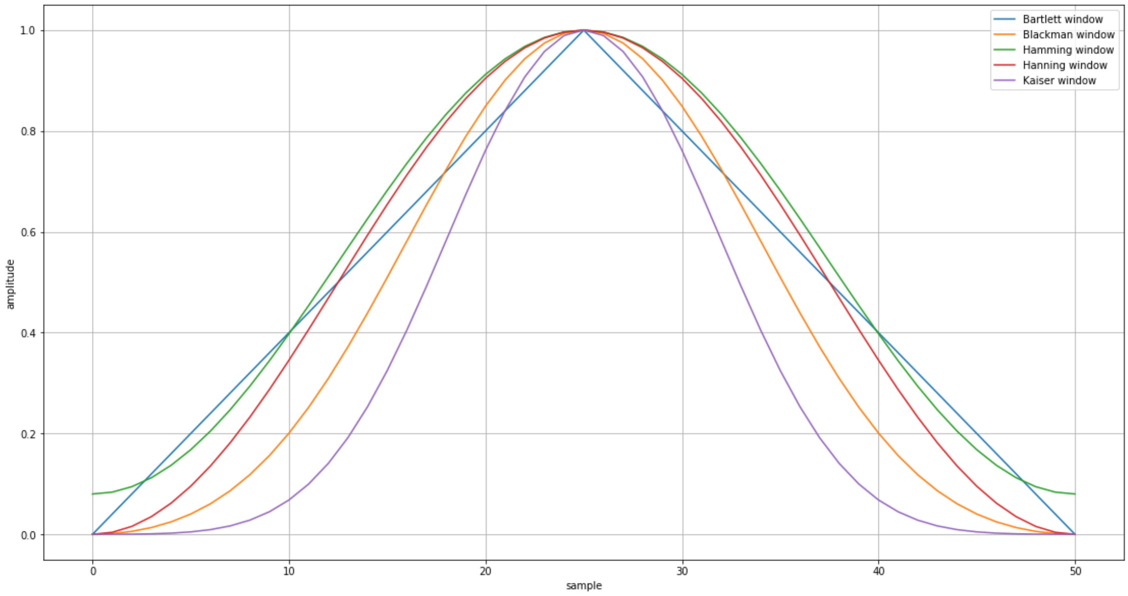
Window Functions

In [14]:

```
fig = plt.figure(figsize=(19, 10))

# Hamming window
window = np.hamming(51)
plt.plot(np.bartlett(51), label="Bartlett window")
plt.plot(np.blackman(51), label="Blackman window")
plt.plot(np.hamming(51), label="Hamming window")
plt.plot(np.hanning(51), label="Hanning window")
plt.plot(np.kaiser(51, 14), label="Kaiser window")
plt.xlabel("sample")
plt.ylabel("amplitude")
plt.legend()
plt.grid()

plt.show()
```



Logic functions

In [1]:

```
import numpy as np
```

In [2]:

```
np.__version__
```

Out[2]:

'1.11.2'

Truth value testing

Q1. Let x be an arbitrary array. Return True if none of the elements of x is zero. Remind that 0 evaluates to False in python.

In [4]:

```
x = np.array([1,2,3])
print np.all(x)

x = np.array([1,0,3])
print np.all(x)
```

```
True
False
```

Q2. Let x be an arbitrary array. Return True if any of the elements of x is non-zero.

In [5]:

```
x = np.array([1,0,0])
print np.any(x)

x = np.array([0,0,0])
print np.any(x)
```

```
True
False
```

Array contents

Q3. Predict the result of the following code.

In [8]:

```
x = np.array([1, 0, np.nan, np.inf])
#print np.isfinite(x)
```

Q4. Predict the result of the following code.

In [10]:

```
x = np.array([1, 0, np.nan, np.inf])
#print np.isinf(x)
```

Q5. Predict the result of the following code.

In [12]:

```
x = np.array([1, 0, np.nan, np.inf])
#print np.isnan(x)
```

Array type testing

Q6. Predict the result of the following code.

In [15]:

```
x = np.array([1+1j, 1+0j, 4.5, 3, 2, 2j])
#print np.iscomplex(x)
```

Q7. Predict the result of the following code.

In [18]:

```
x = np.array([1+1j, 1+0j, 4.5, 3, 2, 2j])
#print np.isreal(x)
```

Q8. Predict the result of the following code.

In [21]:

```
#print np.isscalar(3)
#print np.isscalar([3])
#print np.isscalar(True)
```

Logical operations

Q9. Predict the result of the following code.

In [31]:

```
#print np.logical_and([True, False], [False, False])
#print np.logical_or([True, False, True], [True, False, False])
#print np.logical_xor([True, False, True], [True, False, False])
#print np.logical_not([True, False, 0, 1])
```

Comparison

Q10. Predict the result of the following code.

In [42]:

```
#print np.allclose([3], [2.999999])
#print np.array_equal([3], [2.999999])
```

Q11. Write numpy comparison functions such that they return the results as you see.

In [51]:

```
x = np.array([4, 5])
y = np.array([2, 5])
print np.greater(x, y)
print np.greater_equal(x, y)
print np.less(x, y)
print np.less_equal(x, y)
```

```
[ True False]
[ True  True]
[False False]
[False  True]
```

Q12. Predict the result of the following code.

In [50]:

```
#print np.equal([1, 2], [1, 2.000001])
#print np.isclose([1, 2], [1, 2.000001])
```

Mathematical functions

In [1]:

```
import numpy as np
```

In [2]:

```
np.__version__
```

Out[2]:

```
'1.11.2'
```

In [8]:

Trigonometric functions

Q1. Calculate sine, cosine, and tangent of x, element-wise.

In [26]:

```
x = np.array([0., 1., 30, 90])
print "sine:", np.sin(x)
print "cosine:", np.cos(x)
print "tangent:", np.tan(x)
```

```
sine: [ 0.          0.84147098 -0.98803162  0.89399666]
cosine: [ 1.          0.54030231  0.15425145 -0.44807362]
tangent: [ 0.          1.55740772 -6.4053312  -1.99520041]
```

Q2. Calculate inverse sine, inverse cosine, and inverse tangent of x, element-wise.

In [31]:

```
x = np.array([-1., 0, 1.])
print "inverse sine:", np.arcsin(x2)
print "inverse cosine:", np.arccos(x2)
print "inverse tangent:", np.arctan(x2)
```

```
inverse sine: [ 1.57079633  0.          1.57079633]
inverse cosine: [ 0.          1.57079633  0.        ]
inverse tangent: [ 0.78539816  0.          0.78539816]
```

Q3. Convert angles from radians to degrees.

In [45]:

```
x = np.array([-np.pi, -np.pi/2, np.pi/2, np.pi])

out1 = np.degrees(x)
out2 = np.rad2deg(x)
assert np.array_equiv(out1, out2)
print out1
```

```
[-180.  -90.   90.  180.]
```

Q4. Convert angles from degrees to radians.

In [48]:

```
x = np.array([-180.,  -90.,   90.,  180.])

out1 = np.radians(x)
out2 = np.deg2rad(x)
assert np.array_equiv(out1, out2)
print out1
```

```
[-3.14159265 -1.57079633  1.57079633  3.14159265]
```

Hyperbolic functions

Q5. Calculate hyperbolic sine, hyperbolic cosine, and hyperbolic tangent of x, element-wise.

In [65]:

```
x = np.array([-1., 0, 1.])
print np.sinh(x)
print np.cosh(x)
print np.tanh(x)
```

```
[-1.17520119  0.          1.17520119]
```

```
[ 1.54308063  1.          1.54308063]
[-0.76159416  0.          0.76159416]
```

Rounding

Q6. Predict the results of these, paying attention to the difference among the family functions.

In [84]:

```
x = np.array([2.1, 1.5, 2.5, 2.9, -2.1, -2.5, -2.9])

out1 = np.around(x)
out2 = np.floor(x)
out3 = np.ceil(x)
out4 = np.trunc(x)
out5 = [round(elem) for elem in x]

print out1
print out2
print out3
print out4
print out5
```

```
[ 2.  2.  2.  3. -2. -2. -3.]
[ 2.  1.  2.  2. -3. -3. -3.]
[ 3.  2.  3.  3. -2. -2. -2.]
[ 2.  1.  2.  2. -2. -2. -2.]
[2.0, 2.0, 3.0, 3.0, -2.0, -3.0, -3.0]
```

Q7. Implement out5 in the above question using numpy.

In [87]:

```
print np.floor(np.abs(x) + 0.5) * np.sign(x)
# Read http://numpy-discussion.10968.n7.nabble.com/why-numpy-round-get
-a-different-result-from-python-round-function-td19098.html
```

```
[ 2.  2.  3.  3. -2. -3. -3.]
```

Sums, products, differences

Q8. Predict the results of these.

In [99]:

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

outs = [np.sum(x),
        np.sum(x, axis=0),
        np.sum(x, axis=1, keepdims=True),
        "",
        np.prod(x),
        np.prod(x, axis=0),
        np.prod(x, axis=1, keepdims=True),
        "",
        np.cumsum(x),
        np.cumsum(x, axis=0),
```

```
        np.cumsum(x, axis=1),
        "",
        np.cumprod(x),
        np.cumprod(x, axis=0),
        np.cumprod(x, axis=1),
        "",
        np.min(x),
        np.min(x, axis=0),
        np.min(x, axis=1, keepdims=True),
        "",
        np.max(x),
        np.max(x, axis=0),
        np.max(x, axis=1, keepdims=True),
        "",
        np.mean(x),
        np.mean(x, axis=0),
        np.mean(x, axis=1, keepdims=True)]

for out in outs:
    if out == "":
        print
    else:
        print("->", out)
('->', 36)
('->', array([ 6,  8, 10, 12]))
('->', array([[10],
       [26]]))

('->', 40320)
('->', array([ 5, 12, 21, 32]))
('->', array([[  24],
       [1680]]))

('->', array([ 1,  3,  6, 10, 15, 21, 28, 36]))
('->', array([[ 1,  2,  3,  4],
       [ 6,  8, 10, 12]]))
('->', array([[ 1,  3,  6, 10],
       [ 5, 11, 18, 26]]))

('->', array([    1,     2,     6,    24,   120,   720,  5040, 40320])
)
('->', array([[ 1,  2,  3,  4],
       [ 5, 12, 21, 32]]))
('->', array([[   1,    2,    6,   24],
       [   5,   30,  210, 1680]]))

('->', 1)
('->', array([1, 2, 3, 4]))
```

```
('->', array([[1],
       [5]]))

('->', 8)
('->', array([5, 6, 7, 8]))
('->', array([[4],
       [8]]))

('->', 4.5)
('->', array([ 3.,  4.,  5.,  6.]))
('->', array([[ 2.5],
       [ 6.5]]))
/usr/local/lib/python2.7/dist-packages/ipykernel/__main__.py:34: FutureWarning: elementwise comparison failed; returning scalar instead, but in the future will perform elementwise comparison
```

Q9. Calculate the difference between neighboring elements, element-wise.

In [100]:

```
x = np.array([1, 2, 4, 7, 0])
print np.diff(x)
[ 1  2  3 -7]
```

Q10. Calculate the difference between neighboring elements, element-wise, and prepend [0, 0] and append[100] to it.

In [108]:

```
x = np.array([1, 2, 4, 7, 0])

out1 = np.ediff1d(x, to_begin=[0, 0], to_end=[100])
out2 = np.insert(np.append(np.diff(x), 100), 0, [0, 0])
assert np.array_equiv(out1, out2)
print out2
[  0   0   1   2   3  -7 100]
```

Q11. Return the cross product of x and y.

In [110]:

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

print np.cross(x, y)
[-3  6 -3]
```

Exponents and logarithms

Q12. Compute , element-wise.

In [115]:

```
x = np.array([1., 2., 3.], np.float32)

out = np.exp(x)
print out
[  2.71828175   7.38905621  20.08553696]
```

Q13. Calculate exp(x) - 1 for all elements in x.

In [118]:

```
x = np.array([1., 2., 3.], np.float32)

out1 = np.expm1(x)
out2 = np.exp(x) - 1.
assert np.allclose(out1, out2)
print out1
```

```
[  1.71828175   6.38905621  19.08553696]
```

Q14. Calculate for all p in x.

In [124]:

```
x = np.array([1., 2., 3.], np.float32)

out1 = np.exp2(x)
out2 = 2 ** x
assert np.allclose(out1, out2)
print out1
```

```
[ 2.  4.  8.]
```

Q15. Compute natural, base 10, and base 2 logarithms of x element-wise.

In [128]:

```
x = np.array([1, np.e, np.e**2])

print "natural log =", np.log(x)
print "common log =", np.log10(x)
print "base 2 log =", np.log2(x)
```

```
natural log = [ 0.  1.  2.]
common log = [ 0.          0.43429448  0.86858896]
base 2 log = [ 0.          1.44269504  2.88539008]
```

Q16. Compute the natural logarithm of one plus each element in x in floating-point accuracy.

In [131]:

```
x = np.array([1e-99, 1e-100])

print np.log1p(x)
# Compare it with np.log(1 +x)
```

```
[  1.00000000e-099   1.00000000e-100]
```

Floating point routines

Q17. Return element-wise True where signbit is set.

In [135]:

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

out1 = np.signbit(x)
out2 = x < 0
assert np.array_equiv(out1, out2)
print out1
```

```
[ True  True  True False False False False]
```
Q18. Change the sign of x to that of y, element-wise.

In [140]:

```
x = np.array([-1, 0, 1])
y = -1.1

print np.copysign(x, y)
```
```
[-1. -0. -1.]
```
Arithmetic operations
Q19. Add x and y element-wise.

In [141]:

```
x = np.array([1, 2, 3])
y = np.array([-1, -2, -3])

out1 = np.add(x, y)
out2 = x + y
assert np.array_equal(out1, out2)
print out1
```
```
[0 0 0]
```
Q20. Subtract y from x element-wise.

In [142]:

```
x = np.array([3, 4, 5])
y = np.array(3)

out1 = np.subtract(x, y)
out2 = x - y
assert np.array_equal(out1, out2)
print out1
```
```
[0 1 2]
```
Q21. Multiply x by y element-wise.

In [144]:

```
x = np.array([3, 4, 5])
y = np.array([1, 0, -1])

out1 = np.multiply(x, y)
out2 = x * y
assert np.array_equal(out1, out2)
print out1
```
```
[ 3  0 -5]
```
Q22. Divide x by y element-wise in two different ways.

In [161]:

```
x = np.array([3., 4., 5.])
y = np.array([1., 2., 3.])

out1 = np.true_divide(x, y)
```

```
out2 = x / y
assert np.array_equal(out1, out2)
print out1

out3 = np.floor_divide(x, y)
out4 = x // y
assert np.array_equal(out3, out4)
print out3

# Note that in Python 2 and 3, the handling of `divide` differs.
# See https://docs.scipy.org/doc/numpy/reference/generated/numpy.divid
e.html#numpy.divide
```

```
[ 3.          2.          1.66666667]
[ 3.  2.  1.]
```

Q23. Compute numerical negative value of x, element-wise.

In [146]:

```
x = np.array([1, -1])

out1 = np.negative(x)
out2 = -x
assert np.array_equal(out1, out2)
print out1
```

```
[-1  1]
```

Q24. Compute the reciprocal of x, element-wise.

In [155]:

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

out1 = np.reciprocal(x)
out2 = 1/x
assert np.array_equal(out1, out2)
print out1
```

```
[ 1.   0.5  5. ]
```

Q25. Compute , element-wise.

In [163]:

```
x = np.array([[1, 2], [3, 4]])
y = np.array([[1, 2], [1, 2]])

out = np.power(x, y)
print out
```

```
[[ 1  4]
 [ 3 16]]
```

Q26. Compute the remainder of x / y element-wise in two different ways.

In [168]:

```
x = np.array([-3, -2, -1, 1, 2, 3])
y = 2
```

```
out1 = np.mod(x, y)
out2 = x % y
assert np.array_equal(out1, out2)
print out1

out3 = np.fmod(x, y)
print out3
```

```
[1 0 1 1 0 1]
[-1  0 -1  1  0  1]
```

Miscellaneous

Q27. If an element of x is smaller than 3, replace it with 3. And if an element of x is bigger than 7, replace it with 7.

In [174]:

```
x = np.arange(10)

out1 = np.clip(x, 3, 7)
out2 = np.copy(x)
out2[out2 < 3] = 3
out2[out2 > 7] = 7

assert np.array_equiv(out1, out2)
print out1
```

```
[3 3 3 3 4 5 6 7 7 7]
```

Q28. Compute the square of x, element-wise.

In [176]:

```
x = np.array([1, 2, -1])

out1 = np.square(x)
out2 = x * x
assert np.array_equal(out1, out2)
print out1
```

```
[1 4 1]
```

Q29. Compute square root of x element-wise.

In [177]:

```
x = np.array([1., 4., 9.])

out = np.sqrt(x)
print out
```

```
[ 1.  2.  3.]
```

Q30. Compute the absolute value of x.

In [178]:

```
x = np.array([[1, -1], [3, -3]])

out = np.abs(x)
print out
```

```
[[1 1]
 [3 3]]
```

Q31. Compute an element-wise indication of the sign of x, element-wise.

In [181]:

```
x = np.array([1, 3, 0, -1, -3])

out1 = np.sign(x)
out2 = np.copy(x)
out2[out2 > 0] = 1
out2[out2 < 0] = -1
assert np.array_equal(out1, out2)
print out1
```

```
[ 1  1  0 -1 -1]
```

Random Sampling

In [2]:

```
import numpy as np
```

In [3]:

```
np.__version__
```

Out[3]:

```
'1.11.2'
```

Simple random data

Q1. Create an array of shape (3, 2) and populate it with random samples from a uniform distribution over [0, 1).

In [49]:

```
np.random.rand(3, 2)
# Or np.random.random((3,2))
```

Out[49]:

```
array([[ 0.13879034,  0.71300174],
       [ 0.08121322,  0.00393554],
       [ 0.02349471,  0.56677474]])
```

Q2. Create an array of shape (1000, 1000) and populate it with random samples from a standard normal distribution. And verify that the mean and standard deviation is close enough to 0 and 1 repectively.

In [42]:

```
out1 = np.random.randn(1000, 1000)
out2 = np.random.standard_normal((1000, 1000))
out3 = np.random.normal(loc=0.0, scale=1.0, size=(1000, 1000))
assert np.allclose(np.mean(out1), np.mean(out2), atol=0.1)
assert np.allclose(np.mean(out1), np.mean(out3), atol=0.1)
assert np.allclose(np.std(out1), np.std(out2), atol=0.1)
assert np.allclose(np.std(out1), np.std(out3), atol=0.1)
```

```
print np.mean(out3)
print np.std(out1)
-0.00110028519551
0.999683483393
```

Q3. Create an array of shape (3, 2) and populate it with random integers ranging from 0 to 3 (inclusive) from a discrete uniform distribution.

In [44]:

```
np.random.randint(0, 4, (3, 2))
```

Out[44]:

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

Q4. Extract 1 elements from x randomly such that each of them would be associated with probabilities .3, .5, .2. Then print the result 10 times.

In [58]:

```
x = [b'3 out of 10', b'5 out of 10', b'2 out of 10']
```

In [60]:

```
for _ in range(10):
    print np.random.choice(x, p=[.3, .5, .2])
2 out of 10
5 out of 10
3 out of 10
5 out of 10
5 out of 10
5 out of 10
2 out of 10
2 out of 10
5 out of 10
5 out of 10
```

Q5. Extract 3 different integers from 0 to 9 randomly with the same probabilities.

In [66]:

```
np.random.choice(10, 3, replace=False)
```

Out[66]:

```
array([5, 4, 0])
```

Permutations

Q6. Shuffle numbers between 0 and 9 (inclusive).

In [86]:

```
x = np.arange(10)
np.random.shuffle(x)
print x
[2 3 8 4 5 1 0 6 9 7]
```

In [88]:

```
# Or
print np.random.permutation(10)
```

```
[5 2 7 4 1 0 6 8 9 3]
```

Random generator

Q7. Assign number 10 to the seed of the random generator so that you can get the same value next time.

In [91]:

```
np.random.seed(10)
```

Set routines

In [4]:

```
import numpy as np
```

In [5]:

```
np.__version__
```

Out[5]:

```
'1.11.2'
```

Making proper sets

Q1. Get unique elements and reconstruction indices from x. And reconstruct x.

In [15]:

```
x = np.array([1, 2, 6, 4, 2, 3, 2])
out, indices = np.unique(x, return_inverse=True)
print "unique elements =", out
print "reconstruction indices =", indices
print "reconstructed =", out[indices]
unique elements = [1 2 3 4 6]
reconstruction indices = [0 1 4 3 1 2 1]
reconstructed = [1 2 6 4 2 3 2]
```

Boolean operations

Q2. Create a boolean array of the same shape as x. If each element of x is present in y, the result will be True, otherwise False.

In [19]:

```
x = np.array([0, 1, 2, 5, 0])
y = np.array([0, 1])
print np.in1d(x, y)
[ True  True False False  True]
```

Q3. Find the unique intersection of x and y.

In [20]:

```
x = np.array([0, 1, 2, 5, 0])
y = np.array([0, 1, 4])
print np.intersect1d(x, y)
[0 1]
```

Q4. Find the unique elements of x that are not present in y.

In [21]:

```
x = np.array([0, 1, 2, 5, 0])
```

```
y = np.array([0, 1, 4])
print np.setdiff1d(x, y)
```

[2 5]

Q5. Find the xor elements of x and y.

In [40]:

```
x = np.array([0, 1, 2, 5, 0])
y = np.array([0, 1, 4])
out1 = np.setxor1d(x, y)
out2 = np.sort(np.concatenate((np.setdiff1d(x, y), np.setdiff1d(y, x))
))
assert np.allclose(out1, out2)
print out1
```

[2 4 5]

Q6. Find the union of x and y.

In [42]:

```
x = np.array([0, 1, 2, 5, 0])
y = np.array([0, 1, 4])
out1 = np.union1d(x, y)
out2 = np.sort(np.unique(np.concatenate((x, y))))
assert np.allclose(out1, out2)
print np.union1d(x, y)
```

[0 1 2 4 5]

Soring, searching, and counting

In [3]:

```
import numpy as np
```

In [2]:

```
np.__version__
```

Out[2]:

'1.11.2'

Sorting

Q1. Sort x along the second axis.

In [11]:

```
x = np.array([[1,4],[3,1]])
out = np.sort(x, axis=1)
x.sort(axis=1)
assert np.array_equal(out, x)
print out
```

```
[[1 4]
 [1 3]]
```

Q2. Sort pairs of surnames and first names and return their indices. (first by surname, then by name).

In [13]:

```
surnames =    ('Hertz',    'Galilei', 'Hertz')
first_names = ('Heinrich', 'Galileo', 'Gustav')
print np.lexsort((first_names, surnames))
[1 2 0]
```

Q3. Get the indices that would sort x along the second axis.

In [17]:

```
x = np.array([[1,4],[3,1]])
out = np.argsort(x, axis=1)
print out
[[0 1]
 [1 0]]
```

Q4. Create an array such that its fifth element would be the same as the element of sorted x, and it divide other elements by their value.

In [48]:

```
x = np.random.permutation(10)
print "x =", x
print "\nCheck the fifth element of this new array is 5, the first fou
r elements are all smaller than 5, and 6th through the end are bigger
than 5\n",
out = np.partition(x, 5)
x.partition(5) # in-place equivalent
assert np.array_equal(x, out)
print out
x = [5 1 6 3 9 8 2 7 4 0]

Check the fifth element of this new array is 5, the first four element
s are all smaller than 5, and 6th through the end are bigger than 5
[2 0 4 3 1 5 8 7 6 9]
```

Q5. Create the indices of an array such that its third element would be the same as the element of sorted x, and it divide other elements by their value.

In [56]:

```
x = np.random.permutation(10)
print "x =", x
partitioned = np.partition(x, 3)
indices = np.argpartition(x, 3)
print "partitioned =", partitioned
print "indices =", partitioned
assert np.array_equiv(x[indices], partitioned)
x = [2 8 3 7 5 6 4 0 9 1]
partitioned = [0 1 2 3 4 5 8 6 9 7]
indices = [0 1 2 3 4 5 8 6 9 7]
```

Searching

Q6. Get the maximum and minimum values and their indices of x along the second axis.

In [78]:

```
x = np.random.permutation(10).reshape(2, 5)
print "x =", x
```

```
print "maximum values =", np.max(x, 1)
print "max indices =", np.argmax(x, 1)
print "minimum values =", np.min(x, 1)
print "min indices =", np.argmin(x, 1)
x = [[0 5 9 8 2]
 [3 7 4 1 6]]
maximum values = [9 7]
max indices = [2 1]
minimum values = [0 1]
min indices = [0 3]
```

Q7. Get the maximum and minimum values and their indices of x along the second axis, ignoring NaNs.

In [79]:

```
x = np.array([[np.nan, 4], [3, 2]])
print "maximum values ignoring NaNs =", np.nanmax(x, 1)
print "max indices =", np.nanargmax(x, 1)
print "minimum values ignoring NaNs =", np.nanmin(x, 1)
print "min indices =", np.nanargmin(x, 1)
maximum values ignoring NaNs = [ 4.  3.]
max indices = [1 0]
minimum values ignoring NaNs = [ 4.  2.]
min indices = [1 1]
```

Q8. Get the values and indices of the elements that are bigger than 2 in x.

In [113]:

```
x = np.array([[1, 2, 3], [1, 3, 5]])
print "Values bigger than 2 =", x[x>2]
print "Their indices are ", np.nonzero(x > 2)
assert np.array_equiv(x[x>2], x[np.nonzero(x > 2)])
assert np.array_equiv(x[x>2], np.extract(x > 2, x))
Values bigger than 2 = [3 3 5]
Their indices are  (array([0, 1, 1], dtype=int64), array([2, 1, 2], dt
ype=int64))
```

Q9. Get the indices of the elements that are bigger than 2 in the flattend x.

In [4]:

```
x = np.array([[1, 2, 3], [1, 3, 5]])
print np.flatnonzero(x>2)
assert np.array_equiv(np.flatnonzero(x), x.ravel().nonzero())
[2 4 5]
```

Q10. Check the elements of x and return 0 if it is less than 0, otherwise the element itself.

In [105]:

```
x = np.arange(-5, 4).reshape(3, 3)
print np.where(x <0, 0, x)
[[0 0 0]
 [0 0 0]
 [1 2 3]]
```

Q11. Get the indices where elements of y should be inserted to x to maintain order.

In [109]:

```
x = [1, 3, 5, 7, 9]
y = [0, 4, 2, 6]
np.searchsorted(x, y)
```

Out[109]:

```
array([0, 2, 1, 3], dtype=int64)
```

Counting

Q12. Get the number of nonzero elements in x.

In [120]:

```
x = [[0,1,7,0,0],[3,0,0,2,19]]
print np.count_nonzero(x)
assert np.count_nonzero(x) == len(x[x!=0])
```

Statistics

In [2]:

```
import numpy as np
```

In [3]:

```
np.__version__
```

Out[3]:

```
'1.11.3'
```

Order statistics

Q1. Return the minimum value of x along the second axis.

In [10]:

```
x = np.arange(4).reshape((2, 2))
print("x=\n", x)
print("ans=\n", np.amin(x, 1))
```

```
x=
 [[0 1]
 [2 3]]
ans=
 [0 2]
```

Q2. Return the maximum value of x along the second axis. Reduce the second axis to the dimension with size one.

In [12]:

```
x = np.arange(4).reshape((2, 2))
print("x=\n", x)
print("ans=\n", np.amax(x, 1, keepdims=True))
```

```
x=
 [[0 1]
 [2 3]]
ans=
 [[1]
```

```
 [3]]
```

Q3. Calcuate the difference between the maximum and the minimum of x along the second axis.

In [19]:

```
x = np.arange(10).reshape((2, 5))
print("x=\n", x)

out1 = np.ptp(x, 1)
out2 = np.amax(x, 1) - np.amin(x, 1)
assert np.allclose(out1, out2)
print("ans=\n", out1)
```

```
x=
 [[0 1 2 3 4]
 [5 6 7 8 9]]
ans=
 [4 4]
```

Q4. Compute the 75th percentile of x along the second axis.

In [30]:

```
x = np.arange(1, 11).reshape((2, 5))
print("x=\n", x)

print("ans=\n", np.percentile(x, 75, 1))
```

```
x=
 [[ 1  2  3  4  5]
 [ 6  7  8  9 10]]
ans=
 [ 4.  9.]
```

Averages and variances

Q5. Compute the median of flattened x.

In [33]:

```
x = np.arange(1, 10).reshape((3, 3))
print("x=\n", x)

print("ans=\n", np.median(x))
```

```
x=
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
ans=
 5.0
```

Q6. Compute the weighted average of x.

In [62]:

```
x = np.arange(5)
weights = np.arange(1, 6)

out1 = np.average(x, weights=weights)
```

```
out2 = (x*(weights/weights.sum())).sum()
assert np.allclose(out1, out2)
print(out1)
```

```
2.66666666667
```

Q7. Compute the mean, standard deviation, and variance of x along the second axis.

In [72]:

```
x = np.arange(5)
print("x=\n",x)

out1 = np.mean(x)
out2 = np.average(x)
assert np.allclose(out1, out2)
print("mean=\n", out1)

out3 = np.std(x)
out4 = np.sqrt(np.mean((x - np.mean(x)) ** 2 ))
assert np.allclose(out3, out4)
print("std=\n", out3)

out5 = np.var(x)
out6 = np.mean((x - np.mean(x)) ** 2 )
assert np.allclose(out5, out6)
print("variance=\n", out5)
```

```
x=
 [0 1 2 3 4]
mean=
 2.0
std=
 1.41421356237
variance=
 2.0
```

Correlating

Q8. Compute the covariance matrix of x and y.

In [82]:

```
x = np.array([0, 1, 2])
y = np.array([2, 1, 0])

print("ans=\n", np.cov(x, y))
```

```
ans=
 [[ 1. -1.]
 [-1.  1.]]
```

Q9. In the above covariance matrix, what does the -1 mean?

It means `x` and `y` correlate perfectly in opposite directions.

Q10. Compute Pearson product-moment correlation coefficients of x and y.

In [87]:

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

```
y = np.array([2, 4, 5])

print("ans=\n", np.corrcoef(x, y))
```

```
ans=
 [[ 1.          0.92857143]
 [ 0.92857143  1.        ]]
```

Q11. Compute cross-correlation of x and y.

In [90]:

```
x = np.array([0, 1, 3])
y = np.array([2, 4, 5])

print("ans=\n", np.correlate(x, y))
```

```
ans=
 [19]
```

Histograms

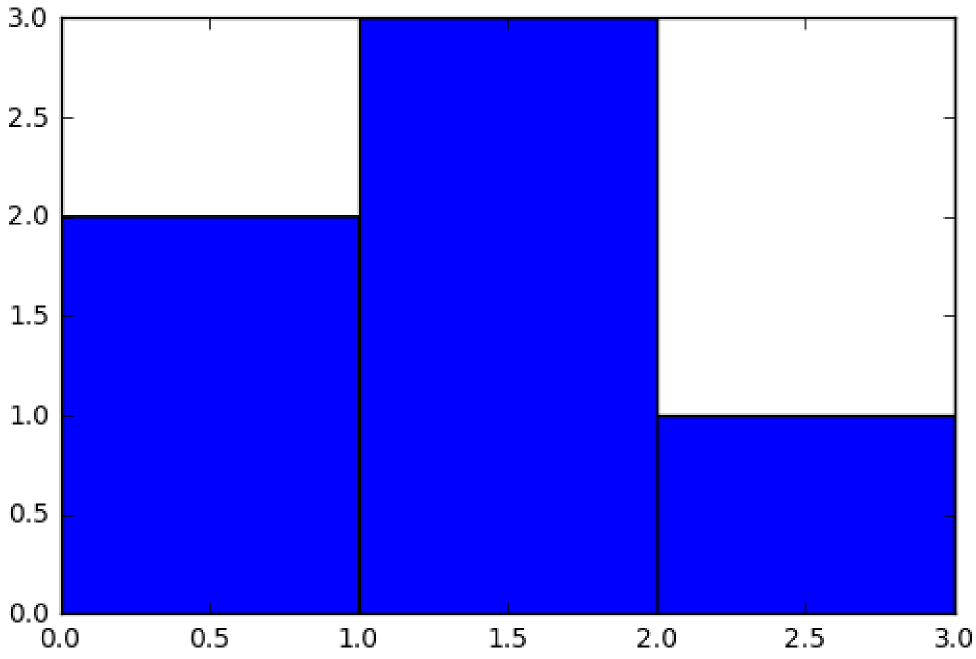
Q12. Compute the histogram of x against the bins.

In [105]:

```
x = np.array([0.5, 0.7, 1.0, 1.2, 1.3, 2.1])
bins = np.array([0, 1, 2, 3])
print("ans=\n", np.histogram(x, bins))

import matplotlib.pyplot as plt
%matplotlib inline
plt.hist(x, bins=bins)
plt.show()
```

```
ans=
 (array([2, 3, 1], dtype=int64), array([0, 1, 2, 3]))
```



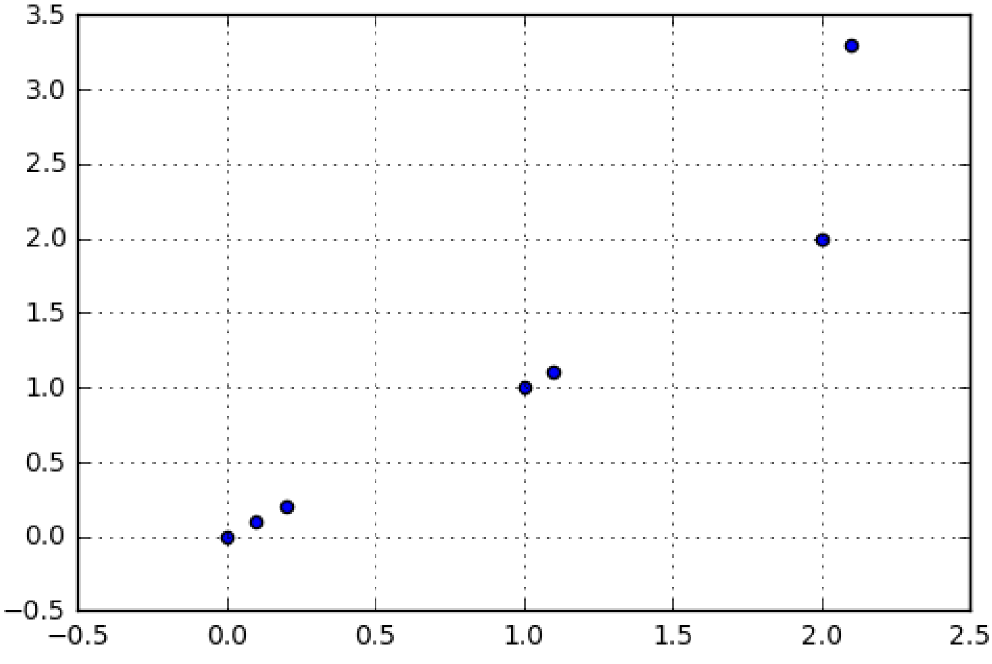
Q13. Compute the 2d histogram of x and y.

In [127]:

```
xedges = [0, 1, 2, 3]
yedges = [0, 1, 2, 3, 4]
x = np.array([0, 0.1, 0.2, 1., 1.1, 2., 2.1])
y = np.array([0, 0.1, 0.2, 1., 1.1, 2., 3.3])
H, xedges, yedges = np.histogram2d(x, y, bins=(xedges, yedges))
print("ans=\n", H)

plt.scatter(x, y)
plt.grid()
```

```
ans=
 [[ 3.  0.  0.  0.]
 [ 0.  2.  0.  0.]
 [ 0.  0.  1.  1.]]
```



Q14. Count number of occurrences of 0 through 7 in x.

In [129]:

```
x = np.array([0, 1, 1, 3, 2, 1, 7])
print("ans=\n", np.bincount(x))
```

```
ans=
 [1 3 1 1 0 0 0 1]
```

Q15. Return the indices of the bins to which each value in x belongs.

In [130]:

```
x = np.array([0.2, 6.4, 3.0, 1.6])
bins = np.array([0.0, 1.0, 2.5, 4.0, 10.0])

print("ans=\n", np.digitize(x, bins))
```

```
ans=
 [1 4 3 2]
```