NumPy

Based on the work of Travis E. Oliphant oliphant@enthought.com

What is NumPy?

- Python is a fabulous language
  - Easy to extend
  - Great syntax which encourages easy to write and maintain code
  - Incredibly large standard-library and third-party tools

- No built-in multi-dimensional array (but it supports the needed syntax for extracting elements from one)

- NumPy provides a **fast** built-in object (***ndarray***) which is a multi-dimensional array of a homogeneous data-type.

Overview of NumPy

N-D ARRAY (NDARRAY)

- N-dimensional array of rectangular data

- Element of the array can be C-structure or simple data-type or object data-type.

- Fast algorithms on machine data-types (int, float, etc.)

Introducing NumPy Arrays

SIMPLE ARRAY CREATION

```
>>> a = array([0,1,2,3])
>>> a
array([0, 1, 2, 3])
```

CHECKING THE TYPE

```
>>> type(a)
<type 'array'>
```

NUMERIC 'TYPE' OF ELEMENTS

```
>>> a.dtype
dtype('int32')
```

BYTES PER ELEMENT

```
>>> a.itemsize # per element
4
```

ARRAY SHAPE

```
# shape returns a tuple
# listing the length of the
# array along each dimension.
>>> a.shape
(4,)
>>> shape(a)
(4,)
```

ARRAY SIZE

```
# size reports the entire
# number of elements in an
# array.
>>> a.size
4
>>> size(a)
4
```

Introducing NumPy Arrays

BYTES OF MEMORY USED

```
# returns the number of bytes
# used by the data portion of
# the array.
 >>> a.nbytes
12
```

NUMBER OF DIMENSIONS

```
>>> a.ndim
1
```

ARRAY COPY

```
# create a copy of the array
>>> b = a.copy()
>>> b
array([0, 1, 2, 3])
```

CONVERSION TO LIST

```
# convert a numpy array to a
# python list.
>>> a.tolist()
[0, 1, 2, 3]

# For 1D arrays, list also
# works equivalently, but
# is slower.
>>> list(a)
[0, 1, 2, 3]
```

Setting Array Elements

ARRAY INDEXING

```
>>> a[0]
0
>>> a[0] = 10
>>> a
[10, 1, 2, 3]
```

FILL

```
# set all values in an array.
>>> a.fill(0)
>>> a
[0, 0, 0, 0]

# This also works, but may
# be slower.
>>> a[:] = 1
>>> a
[1, 1, 1, 1]
```

BEWARE OF TYPE COERSION

```
>>> a.dtype
dtype('int32')

# assigning a float to into
# an int32 array will
# truncate decimal part.
>>> a[0] = 10.6
>>> a
[10, 1, 2, 3]

# fill has the same behavior
>>> a.fill(-4.8)
>>> a
[-4, -4, -4, -4]
```

Multi-Dimensional Arrays

MULTI-DIMENSIONAL ARRAYS

```
>>> a = array([[ 0, 1, 2, 3],
               [10,11,12,13]])
>>> a
array([[ 0, 1, 2, 3],
       [10,11,12,13]])
```

(ROWS,COLUMNS)

```
>>> a.shape
(2, 4)
>>> shape(a)
(2, 4)
```

ELEMENT COUNT

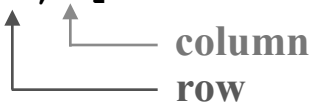
```
>>> a.size
8
>>> size(a)
8
```

NUMBER OF DIMENSIONS

```
>>> a.ndims
2
```

GET/SET ELEMENTS

```
>>> a[1,3]
13
```




```
>>> a[1,3] = -1
>>> a
array([[ 0, 1, 2, 3],
       [10,11,12,-1]])
```

ADDRESS FIRST ROW USING SINGLE INDEX

```
>>> a[1]
array([10, 11, 12, -1])
```

Exercise: creating array “a”

CREATING ARRAY “a” AS FOLLOWS:

0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

Setting up Array “a”

CREATING ARRAY “a” AS FOLLOWS:

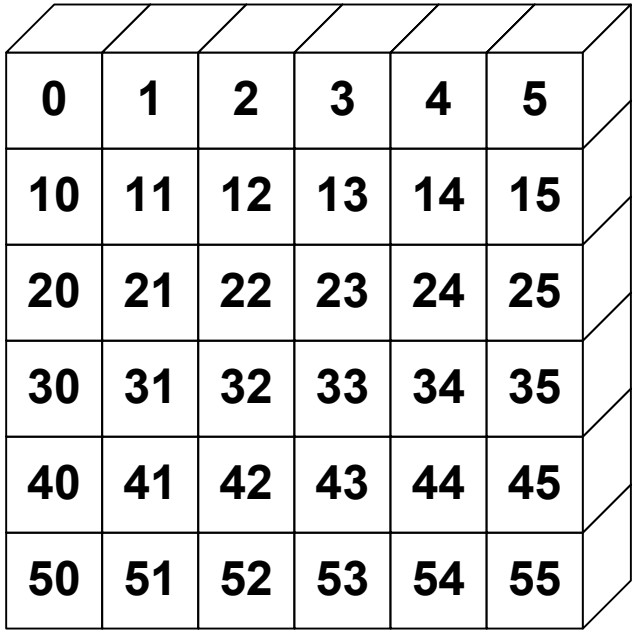
```
>>> a = np.arange(0,6)

>>> a = a.repeat(6)

>>> a = a.reshape(6,6).T

>>> b = np.arange(0,6)*10

>>> a = (b+a.T).T
```



- OR -

CREATING ARRAY “a” AS FOLLOWS:

0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

```
a = np.ones((6,6),dtype='int32')

for row in range(6):
        for col in range(6):
                a[row,col] = row*10+col
```

Array Slicing

SLICING WORKS MUCH LIKE STANDARD PYTHON SLICING

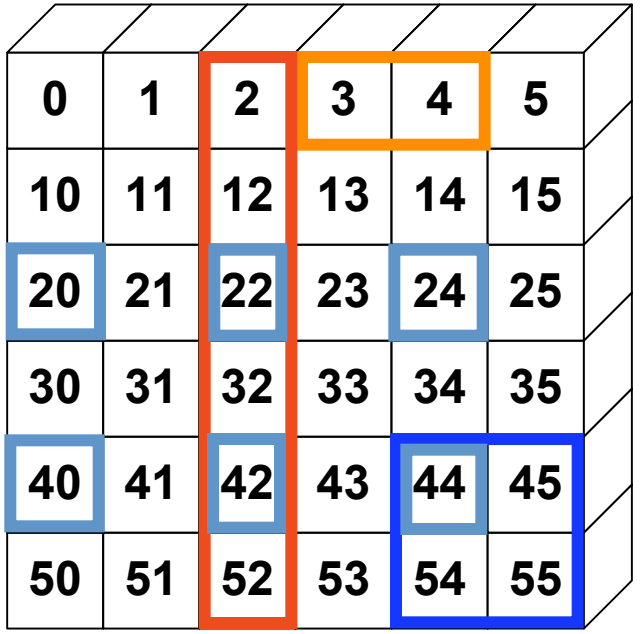
```
>>> a[0,3:5]
array([3, 4])

>>> a[4:,4:]
array([[44, 45],
       [54, 55]])

>>> a[:,2]
array([2,12,22,32,42,52])
```

STRIDES ARE ALSO POSSIBLE

```
>>> a[2::2,::2]
array([[20, 22, 24],
       [40, 42, 44]])
```



Slices Are References

Slices are references to memory in original array. Changing values in a slice also changes the original array. (*including strides*)

```
>>> a = array((0,1,2,3,4))

# create a slice containing only the
# last element of a
>>> b = a[2:4]
>>> b[0] = 10

# changing b changed a!
>>> a
array([ 1,  2, 10, 3, 4])
```

Quiz #1

What will be printed on the screen using the following code?

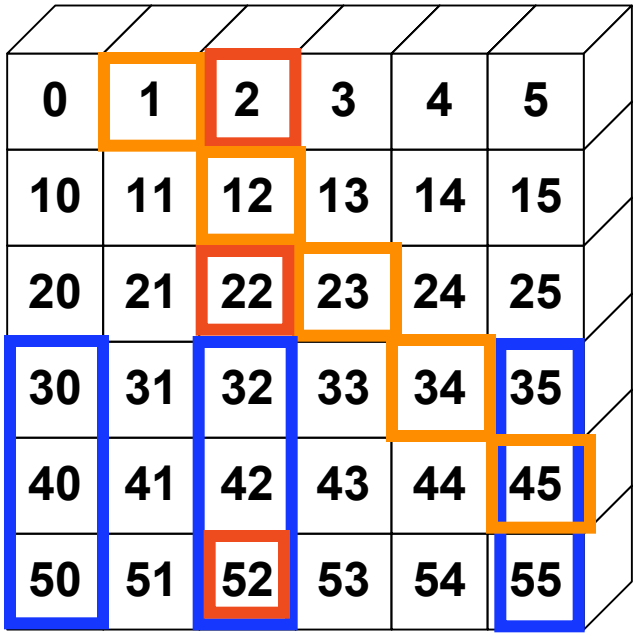
```
a = np.array((0,1,2,3))
b = a
b[2] = -20
b = b.reshape(2,2)
b[1,1] = -5
print a
print a.sum()
print '--------'
print b
print b.sum()
```

Fancy Indexing in 2D

```
>>> a[(0,1,2,3,4),(1,2,3,4,5)]
array([ 1, 12, 23, 34, 45])

>>> a[3:,[0, 2, 5]]
array([[30, 32, 35],
       [40, 42, 45]])
       [50, 52, 55]])

>>> mask = array([1,0,1,0,0,1],
                 dtype=bool)
>>> a[mask,2]
array([2,22,52])
```

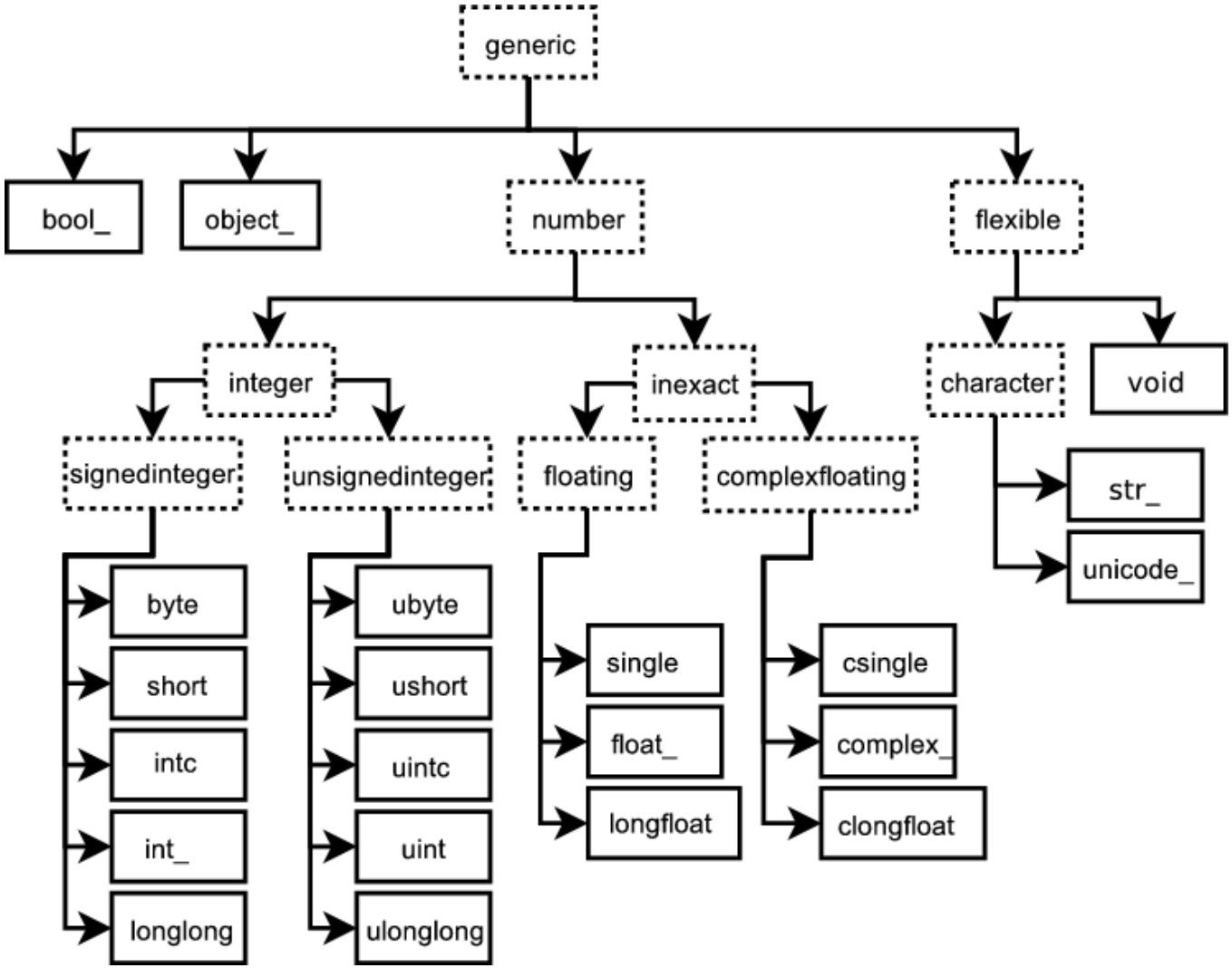


Unlike slicing, fancy indexing creates copies instead of views into original arrays.

NumPy dtypes

Basic Type	Available NumPy types	Comments
Boolean	bool	Elements are 1 byte in size
Integer	int8, int16, int32, int64, int128, int	int defaults to the size of int in C for the platform
Unsigned Integer	uint8, uint16, uint32, uint64, uint128, uint	uint defaults to the size of unsigned int in C for the platform
Float	float32, float64, float, longfloat,	Float is always a double precision floating point value (64 bits). longfloat represents large precision floats. Its size is platform dependent.
Complex	complex64, complex128, complex	The real and complex elements of a complex64 are each represented by a single precision (32 bit) value for a total size of 64 bits.
Strings	str, unicode	Unicode is always UTF32 (UCS4)
Object	object	Represent items in array as Python objects.
Records	void	Used for arbitrary data structures in record arrays.

Built-in “scalar” types



Defining Data-Types

- An item can include fields of different data-types.
- A field is described by a data-type object and a byte offset --- this definition allows nested records.
- The array construction command interprets tuple elements as field entries.

```
>>> my_data_type = np.dtype("int32,float32,bool")

>>> a = np.array([(1,2.35,True), (2,3.42,False)],
dtype=my_data_type)

>>> print a['f0']
[Hello World]
```

Array Calculation Methods

SUM FUNCTION

```
>>> a = array([[1,2,3],
               [4,5,6]], float)

# Sum defaults to summing all
# *all* array values.
>>> sum(a)
21.

# supply the keyword axis to
# sum along the 0th axis.
>>> sum(a, axis=0)
array([5., 7., 9.])

# supply the keyword axis to
# sum along the last axis.
>>> sum(a, axis=-1)
array([6., 15.])
```

SUM ARRAY METHOD

```
# The a.sum() defaults to
# summing *all* array values
>>> a.sum()
21.

# Supply an axis argument to
# sum along a specific axis.
>>> a.sum(axis=0)
array([5., 7., 9.])
```

PRODUCT

```
# product along columns.
>>> a.prod(axis=0)
array([ 4., 10., 18.])

# functional form.
>>> prod(a, axis=0)
array([ 4., 10., 18.])
```

Min/Max

MIN

```
>>> a = array([2.,3.,0.,1.])
>>> a.min(axis=0)
0.
# use Numpy's amin() instead
# of Python's builtin min()
# for speed operations on
# multi-dimensional arrays.
>>> amin(a, axis=0)
0.
```

MAX

```
>>> a = array([2.,1.,0.,3.])
>>> a.max(axis=0)
3.

# functional form
>>> amax(a, axis=0)
3.
```

ARGMIN

```
# Find index of minimum value.
>>> a.argmin(axis=0)
2
# functional form
>>> argmin(a, axis=0)
2
```

ARGMAX

```
# Find index of maximum value.
>>> a.argmax(axis=0)
1
# functional form
>>> argmax(a, axis=0)
1
```

Statistics Array Methods

MEAN

```
>>> a = array([[1,2,3],
              [4,5,6]], float)

# mean value of each column
>>> a.mean(axis=0)
array([ 2.5,  3.5,  4.5])
>>> mean(a, axis=0)
array([ 2.5,  3.5,  4.5])
>>> average(a, axis=0)
array([ 2.5,  3.5,  4.5])

# average can also calculate
# a weighted average
>>> average(a, weights=[1,2],
...         axis=0)
array([ 3., 4., 5.])
```

STANDARD DEV./VARIANCE

```
# Standard Deviation
>>> a.std(axis=0)
array([ 1.5,  1.5,  1.5])

# Variance
>>> a.var(axis=0)
array([2.25, 2.25, 2.25])
>>> var(a, axis=0)
array([2.25, 2.25, 2.25])
```

Other Array Methods

CLIP

```
# Limit values to a range

>>> a = array([[1,2,3],
               [4,5,6]], float)

# Set values < 3 equal to 3.
# Set values > 5 equal to 5.
>>> a.clip(3,5)
>>> a
array([[ 3.,  3.,  3.],
       [ 4.,  5.,  5.]])
```

ROUND

```
# Round values in an array.
# Numpy rounds to even, so
# 1.5 and 2.5 both round to 2.
>>> a = array([1.35, 2.5, 1.5])
>>> a.round()
array([ 1.,  2.,  2.])

# Round to first decimal place.
>>> a.round(decimals=1)
array([ 1.4,  2.5,  1.5])
```

POINT TO POINT

```
# Calculate max - min for
# array along columns
>>> a.ptp(axis=0)
array([ 3.0,  3.0,  3.0])
# max - min for entire array.
>>> a.ptp(axis=None)
5.0
```

Summary of (most) array attributes/methods

BASIC ATTRIBUTES

```
a.dtype - Numerical type of array elements. float32, uint8, etc.
a.shape - Shape of the array. (m,n,o,...)
a.size - Number of elements in entire array.
a.itemsize - Number of bytes used by a single element in the array.
a.nbytes - Number of bytes used by entire array (data only).
a.ndim - Number of dimensions in the array.
```

SHAPE OPERATIONS

```
a.flat - An iterator to step through array as if it is 1D.
a.flatten() - Returns a 1D copy of a multi-dimensional array.
a.resize(new_size) - Change the size/shape of an array in-place.
a.swapaxes(axis1, axis2) - Swap the order of two axes in an array.
a.transpose(*axes) - Swap the order of any number of array axes.
a.T - Shorthand for a.transpose()
```

Summary of (most) array attributes/methods

FILL AND COPY

```
a.copy() – Return a copy of the array.
a.fill(value) – Fill array with a scalar value.
```

CONVERSION / COERSION

```
a.tolist() – Convert array into nested lists of values.
a.tostring() – raw copy of array memory into a python string.
a.astype(dtype) – Return array coerced to given dtype.
```

COMPLEX NUMBERS

```
a.real – Return the real part of the array.
a.imag – Return the imaginary part of the array.
a.conjugate() – Return the complex conjugate of the array.
a.conj()– Return the complex conjugate of an array.(same as conjugate)
```

Summary of (most) array attributes/methods

SAVING

```
a.dump(file) - Store a binary array data out to the given file.
a.dumps() - returns the binary pickle of the array as a string.
a.tofile(fid, sep="", format="%s") Formatted ascii output to file.
```

SEARCH / SORT

```
a.nonzero() - Return indices for all non-zero elements in a.
a.sort(axis=-1) - Inplace sort of array elements along axis.
a.argsort(axis=-1) - Return indices for element sort order along axis.
a.searchsorted(b) - Return index where elements from b would go in a.
```

ELEMENT MATH OPERATIONS

```
a.clip(low, high) - Limit values in array to the specified range.
a.round(decimals=0) - Round to the specified number of digits.
a.cumsum(axis=None) - Cumulative sum of elements along axis.
a.cumprod(axis=None) - Cumulative product of elements along axis.
```

Summary of (most) array attributes/methods

REDUCTION METHODS

All the following methods "reduce" the size of the array by 1 dimension by carrying out an operation along the specified axis. If axis is None, the operation is carried out across the entire array.

a.sum(axis=None) – Sum up values along axis.
a.prod(axis=None) – Find the product of all values along axis.
a.min(axis=None)– Find the minimum value along axis.
a.max(axis=None) – Find the maximum value along axis.
a.argmin(axis=None) – Find the index of the minimum value along axis.
a.argmax(axis=None) – Find the index of the maximum value along axis.
a.ptp(axis=None) – Calculate a.max(axis) – a.min(axis)
a.mean(axis=None) – Find the mean (average) value along axis.
a.std(axis=None) – Find the standard deviation along axis.
a.var(axis=None) – Find the variance along axis.

a.any(axis=None) – True if any value along axis is non-zero. (or)
a.all(axis=None) – True if all values along axis are non-zero. (and)

Array Operations

SIMPLE ARRAY MATH

```
>>> a = array([1,2,3,4])
>>> b = array([2,3,4,5])
>>> a + b
array([3, 5, 7, 9])
```

Numpy defines the following constants:

pi = 3.14159265359

e = 2.71828182846

MATH FUNCTIONS

```
# Create array from 0 to 10
>>> x = arange(11.)

# multiply entire array by
# scalar value
>>> a = (2*pi)/10.
>>> a
0.6283185307179862
>>> a*x
array([ 0.,0.628,…,6.283])

# inplace operations
>>> x *= a
>>> x
array([ 0.,0.628,…,6.283])

# apply functions to array.
>>> y = sin(x)
```

Mathematic Binary Operators

```
a + b →  add(a,b)
a - b →  subtract(a,b)
a % b →  remainder(a,b)
```

```
a * b  →  multiply(a,b)
a / b  →  divide(a,b)
a ** b →  power(a,b)
```

MULTIPLY BY A SCALAR

```
>>> a = array((1,2))
>>> a*3.
array([3., 6.])
```

ELEMENT BY ELEMENT ADDITION

```
>>> a = array([1,2])
>>> b = array([3,4])
>>> a + b
array([4, 6])
```

ADDITION USING AN OPERATOR FUNCTION

```
>>> add(a,b)
array([4, 6])
```

IN PLACE OPERATION

```
# Overwrite contents of a.
# Saves array creation
# overhead
>>> add(a,b,a) # a += b
array([4, 6])
>>> a
array([4, 6])
```

Comparison and Logical Operators

equal (==)	not_equal (!=)	greater (>)
greater_equal (>=)	less (<)	less_equal (<=)
logical_and	logical_or	logical_xor
logical_not		

2D EXAMPLE

```
>>> a = array(((1,2,3,4),(2,3,4,5)))
>>> b = array(((1,2,5,4),(1,3,4,5)))
>>> a == b
array([[True, True, False, True],
       [False, True, True, True]])
# functional equivalent
>>> equal(a,b)
array([[True, True, False, True],
       [False, True, True, True]])
```

Bitwise Operators

bitwise_and (&)	invert (~)	right_shift(a,shifts)
bitwise_or (|)	bitwise_xor	left_shift (a,shifts)

BITWISE EXAMPLES

```
>>> a = array((1,2,4,8))
>>> b = array((16,32,64,128))
>>> bitwise_or(a,b)
array([ 17,  34,  68, 136])

# bit inversion
>>> a = array((1,2,3,4), uint8)
>>> invert(a)
array([254, 253, 252, 251], dtype=uint8)

# left shift operation
>>> left_shift(a,3)
array([ 8, 16, 24, 32], dtype=uint8)
```

Trig and Other Functions

TRIGONOMETRIC

```
sin(x)        sinh(x)
cos(x)        cosh(x)
arccos(x)     arccosh(x)

arctan(x)     arctanh(x)
arcsin(x)     arcsinh(x)
arctan2(x,y)
```

OTHERS

```
exp(x)        log(x)
log10(x)      sqrt(x)
absolute(x)   conjugate(x)
negative(x)   ceil(x)
floor(x)      fabs(x)
hypot(x,y)    fmod(x,y)
maximum(x,y)  minimum(x,y)
```

`hypot(x,y)`

Element by element distance calculation using $\sqrt{x^2 + y^2}$

Broadcasting

When there are multiple inputs, then they all must be "broadcastable" to the same shape.

- All arrays are promoted to the same number of dimensions (by pre-prending 1's to the shape)
- All dimensions of length 1 are expanded as determined by other inputs with non-unit lengths in that dimension.

```
a = np.array([0,1,2])
b = np.array([[0],[10],[20],[30]])
print a+b
```

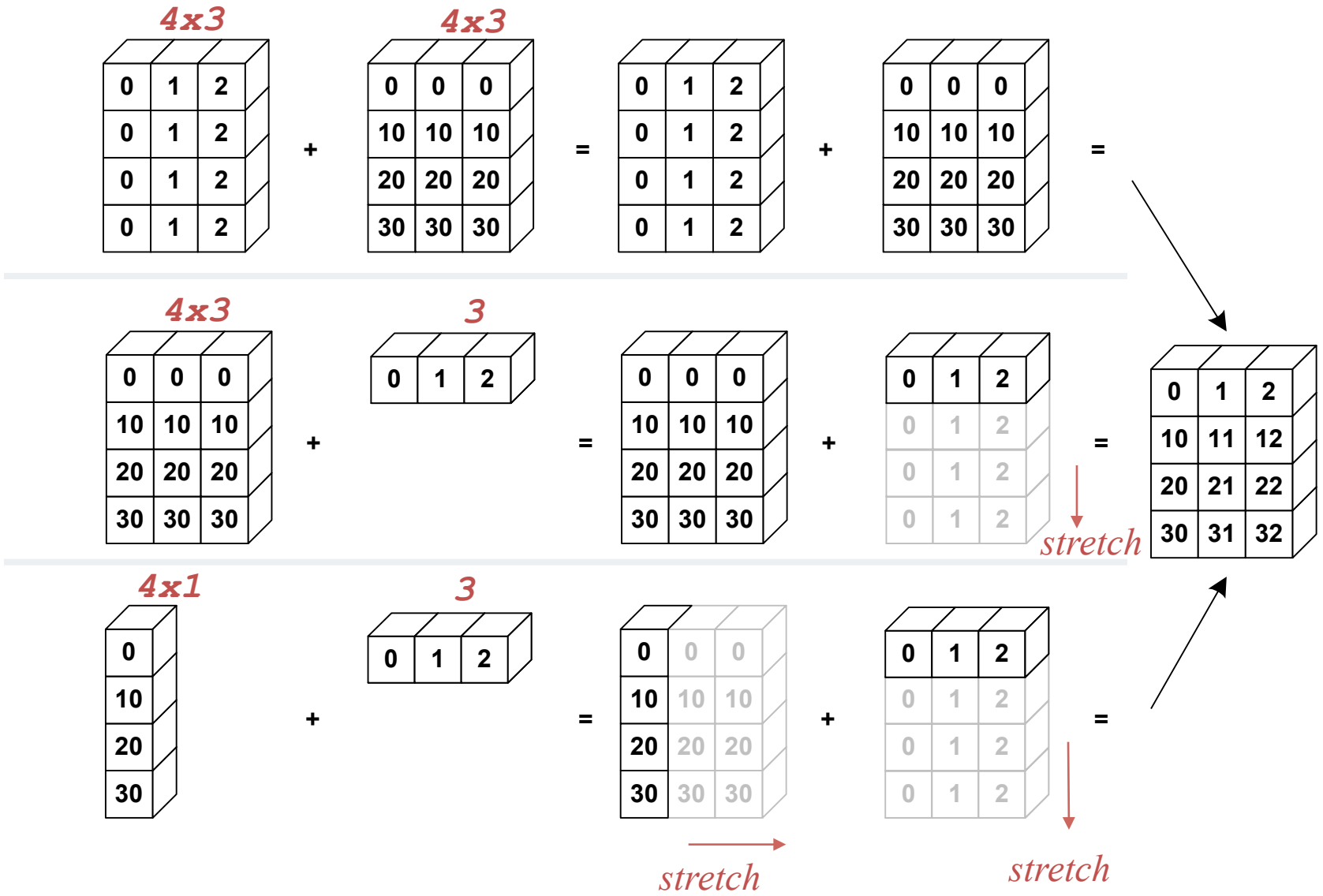
```
[[ 0  1  2]
 [10 11 12]
 [20 21 22]
 [30 31 32]]
```

a has shape (3,) the ufunc sees it as having shape (1,3)

b has shape (4,1)

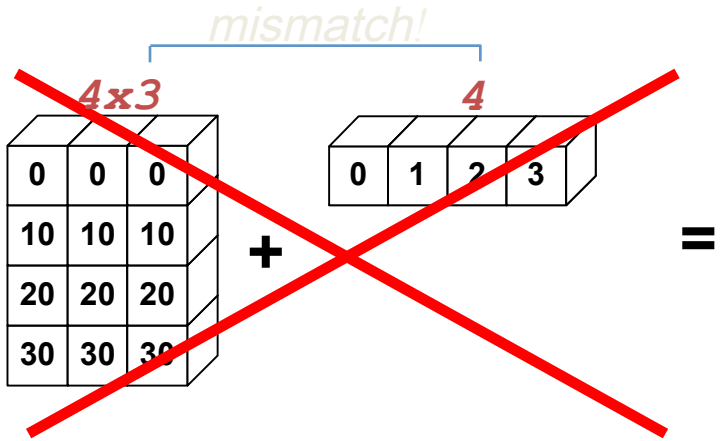
The ufunc result has shape (4,3)

Array Broadcasting



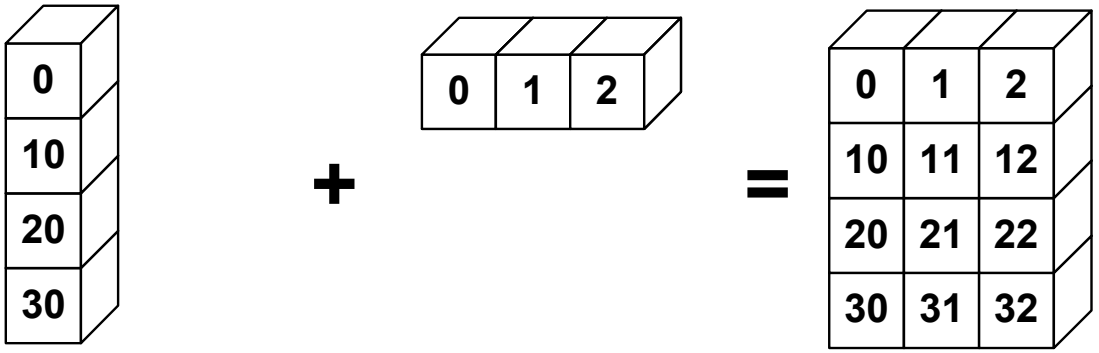
Broadcasting Rules

The *trailing* axes of both arrays must either be 1 or have the same size for broadcasting to occur. Otherwise, a "`ValueError: frames are not aligned`" exception is thrown.



Broadcasting in Action

```
>>> a = array((0,10,20,30))
>>> b = array((0,1,2))
>>> y = a[:, None] + b
```



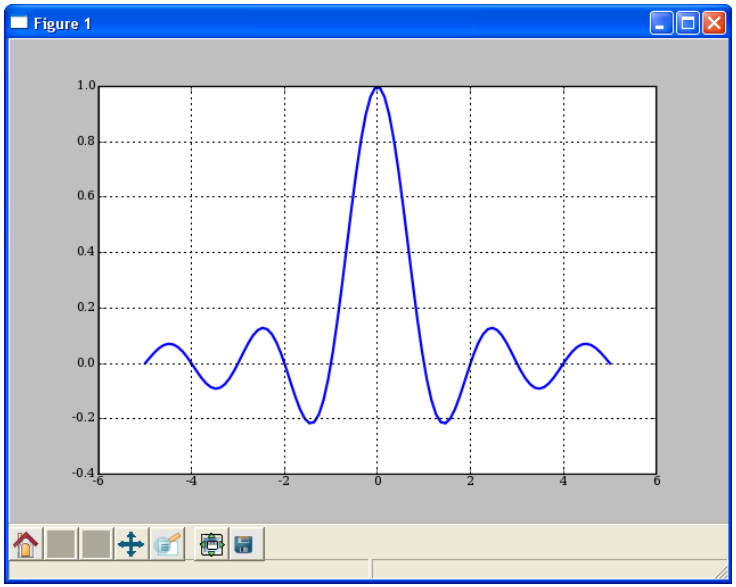
Vectorizing Functions

VECTORIZING FUNCTIONS

Example

```
def sinc(x):
    if x == 0.0:
        return 1.0
    else:
        w = pi*x
        return sin(w) / w
```

```
# attempt
>>> sinc([1.3,1.5])
TypeError: can't multiply
sequence to non-int
>>> x = r_[-5:5:100j]
>>> y = vsinc(x)
>>> plot(x, y)
```

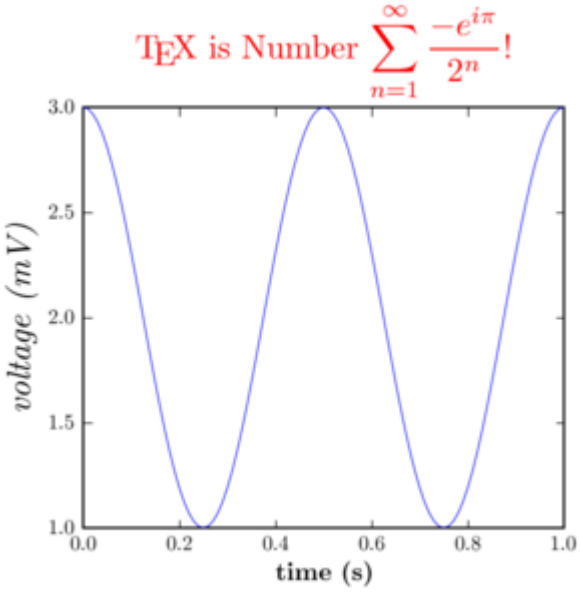


SOLUTION

```
>>> from numpy import vectorize
>>> vsinc = vectorize(sinc)
>>> vsinc([1.3,1.5])
array([-0.1981, -0.2122])
```

Matplotlib

- Requires NumPy extension. Provides powerful plotting commands.
- http://matplotlib.sourceforge.net



Recommendations

- **Matplotlib** for day-to-day data exploration.

  Matplotlib has a large community, tons of plot types, and is well integrated into ipython. It is the de-facto standard for 'command line' plotting from ipython.

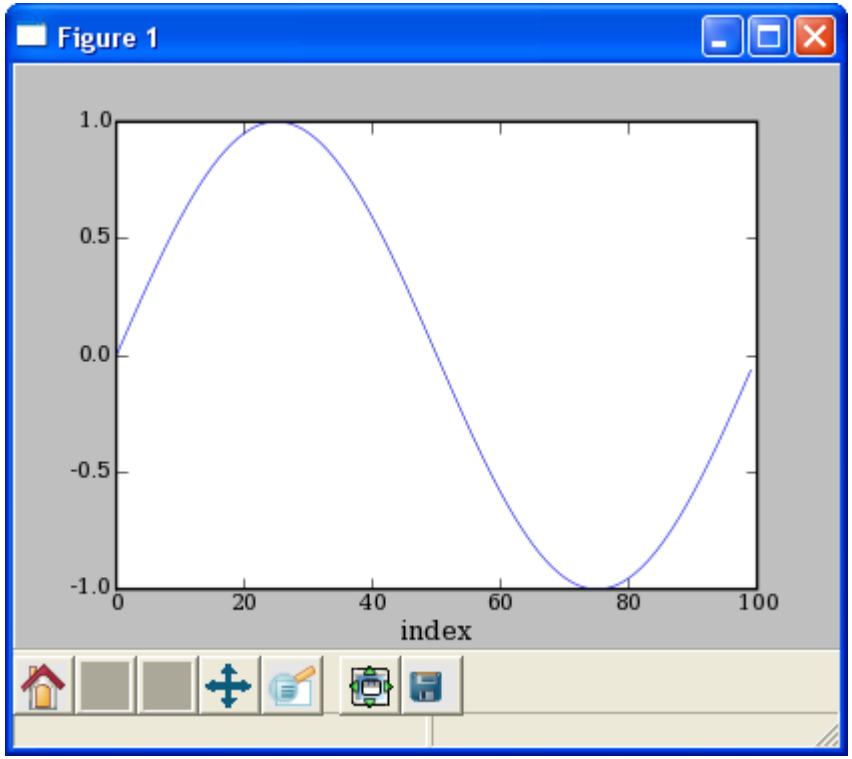
- **Chaco** for building interactive plotting applications

  Chaco is architected for building highly interactive and configurable plots in python. It is more useful as plotting toolkit than for making one-off plots.

Line Plots

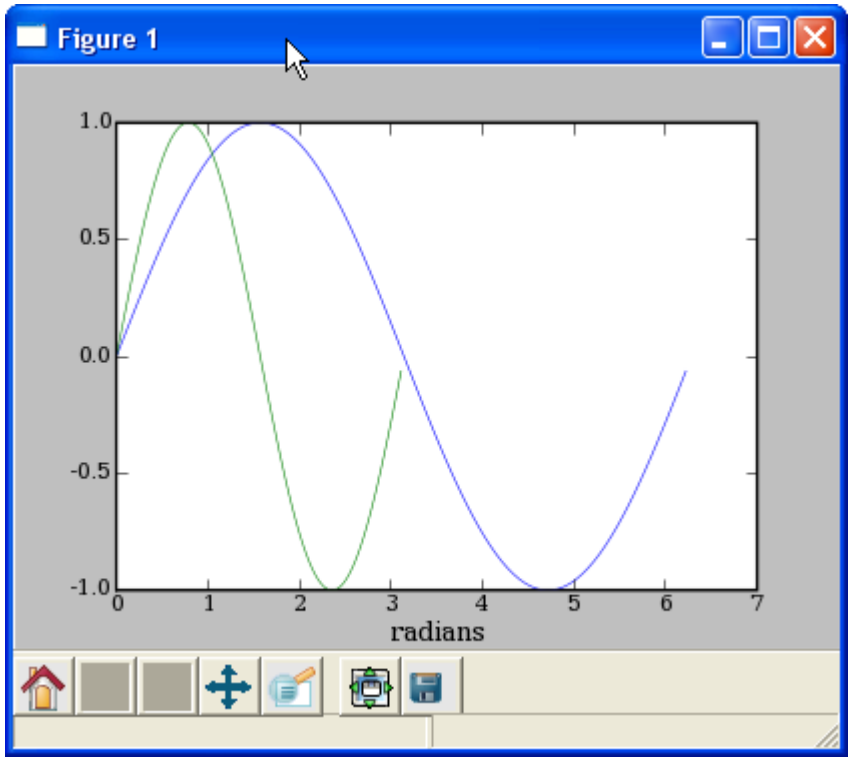
PLOT AGAINST INDICES

```
>>> x = arange(50)*2*pi/50.
>>> y = sin(x)
>>> plot(y)
>>> xlabel('index')
```



MULTIPLE DATA SETS

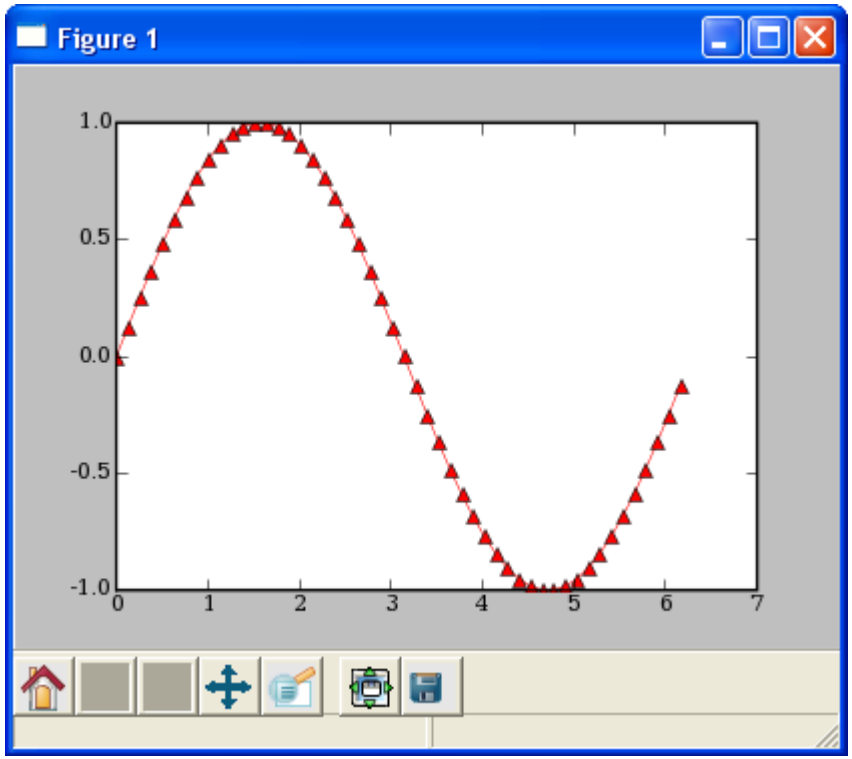
```
>>> plot(x,y,x2,y2)
>>> xlabel('radians')
```



Line Plots

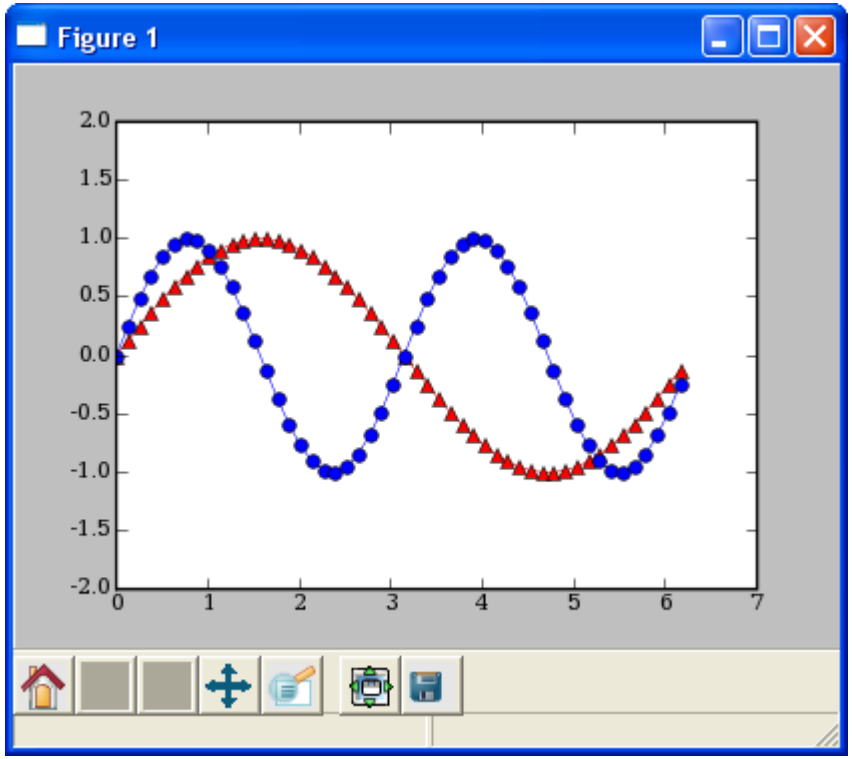
LINE FORMATTING

```
# red, dot-dash, triangles
>>> plot(x,sin(x),'r-^')
```



MULTIPLE PLOT GROUPS

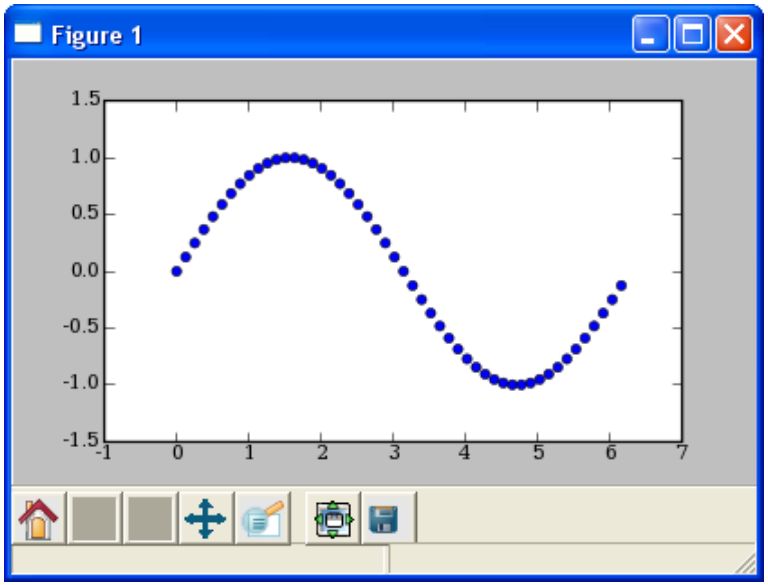
```
>>> plot(x,y1,'b-o', x,y2), r-^')
>>> axis([0,7,-2,2])
```



Scatter Plots

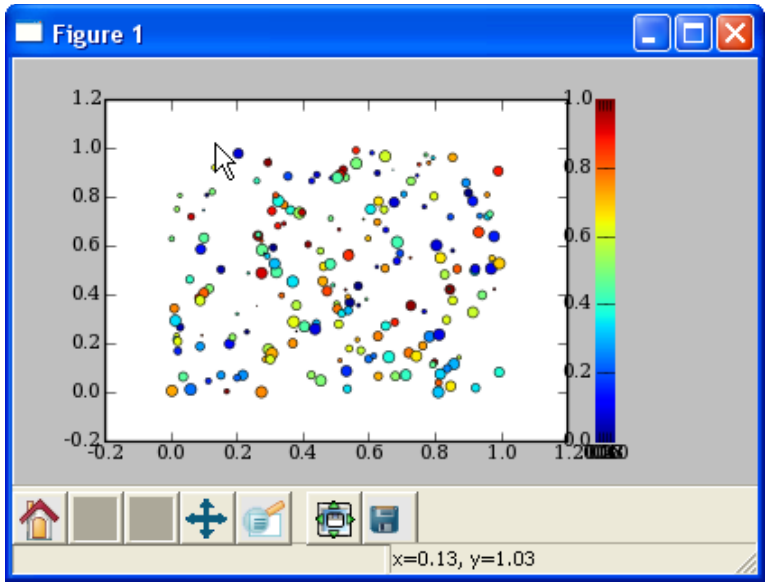
SIMPLE SCATTER PLOT

```
>>> x = arange(50)*2*pi/50.
>>> y = sin(x)
>>> scatter(x,y)
```



COLORMAPPED SCATTER

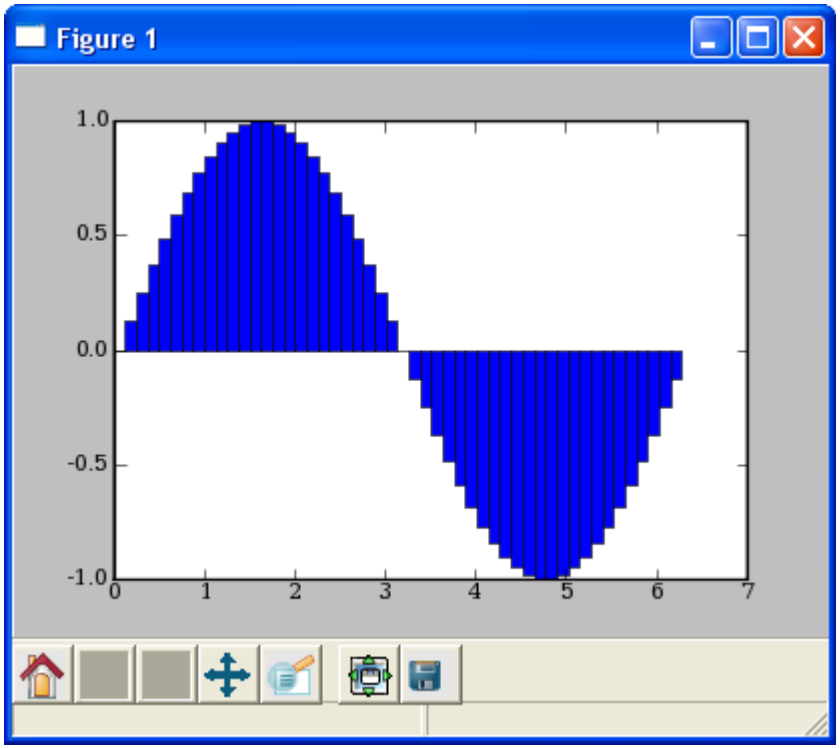
```
# marker size/color set with data
>>> x = rand(200)
>>> y = rand(200)
>>> size = rand(200)*30
>>> color = rand(200)
>>> scatter(x, y, size, color)
>>> colorbar()
```



Bar Plots

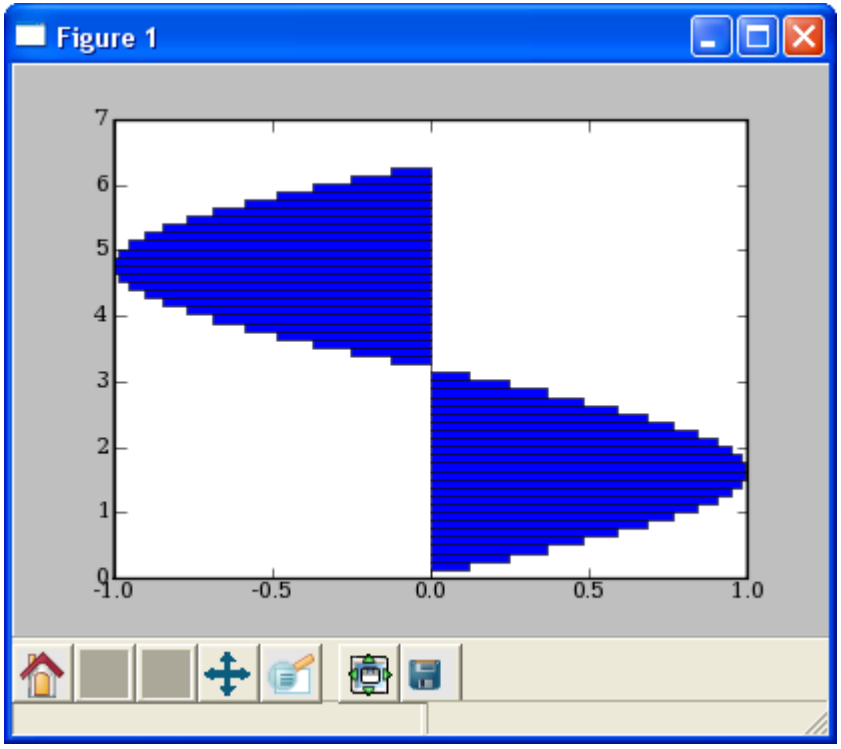
BAR PLOT

```
>>> bar(x,sin(x),
...      width=x[1]-x[0])
```



HORIZONTAL BAR PLOT

```
>>> hbar(x,sin(x),
...       height=x[1]-x[0],
...      orientation='horizontal')
```



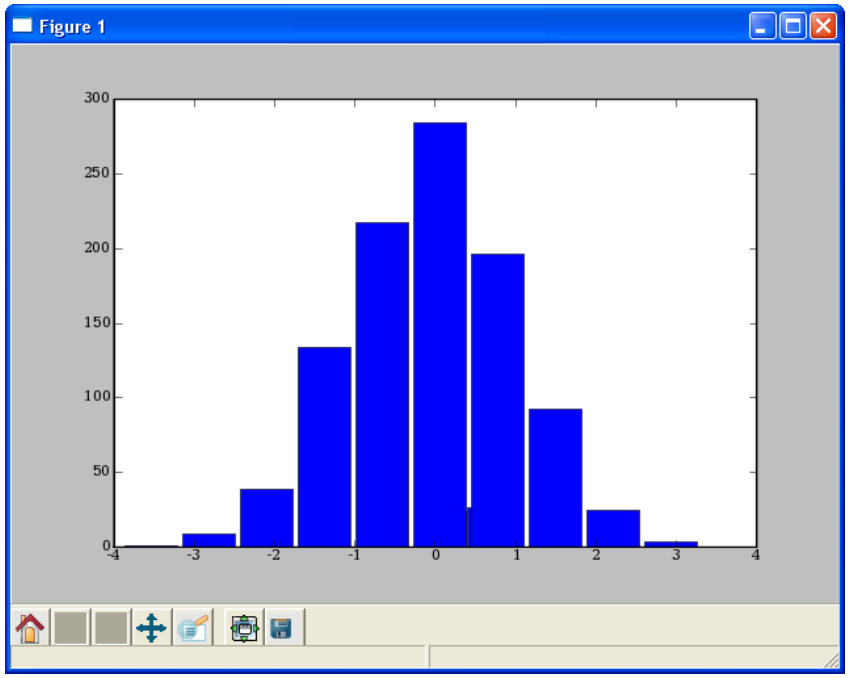
Bar Plots

DEMO/MATPLOTLIB_PLOTTING/BARCHART_DEMO.PY

HISTOGRAMS

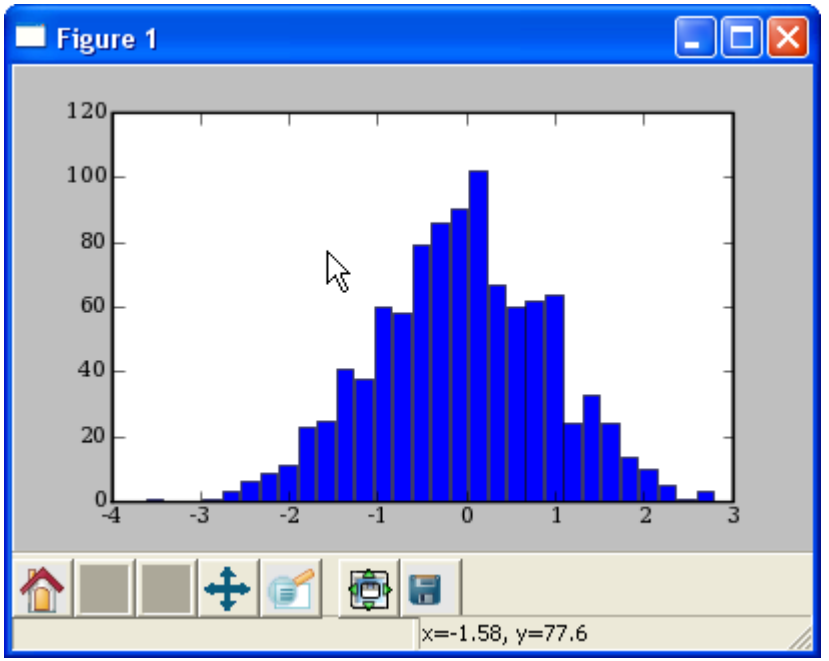
HISTOGRAM

```
# plot histogram
# default to 10 bins
>>> hist(randn(1000))
```



HISTOGRAM 2

```
# change the number of bins
>>> hist(randn(1000), 30)
```



Questions? :-)