Introduction to NumPy

BYU
Electrical Engineering
Computer Engineering

Travis E. Oliphant
Electrical Engineering
Brigham Young University
Provo, Utah

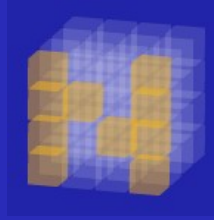
http://numeric.scipy.org

NumPy

BYU
Electrical Engineering
Computer Engineering

- NumPy
  - An N-dimensional “homogeneous” array object
  - Universal element-by-element function objects (ufuncs)
  - Basic linear algebra
  - Random number generation
  - Fast Fourier transforms
  - Masked arrays
  - Fortran (and simple C) wrapping tool (f2py)
- It builds on the original Numeric code base but adds the features developed by Numarray (plus additional features)

Num????????



BYU
Electrical Engineering
Computer Engineering

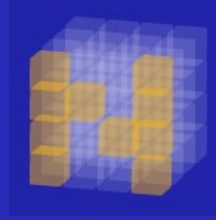
- Numeric
  - started in 1995 by Jim Hugunin
  - Large user-base (33972 downloads of 24.2)
  - Data-types limited and difficult to add new data-types
- Numarray (module numarray)
  - started in 2001 by Todd Miller, Rick White, and Perry Greenfield (STSCI) --- (39115 downloads of 1.5.1)
  - Added record arrays and character arrays
  - Allowed use of index arrays to select elements of an array
  - Support for memory-mapped files
  - Slow for small arrays (Python implementation)
- NumPy (module numpy)
  - Built on the Numeric code-base
  - Enhanced data-types (14674 downloads of 0.9.8)
  - Hybrid of Numarray and Numeric

NumPy is the future

BYU
Electrical Engineering
Computer Engineering

- Numeric is no longer maintained.
- Numarray developers at STSCI have stated that they will only support it for a transition period less than one year.
- NumPy includes compatibility layers for both Numeric and Numarray.
- The community behind NumPy is vibrant and growing.
- ***Guide to NumPy*** a book which covers the system rather completely is available for purchase now but will be completely free in at most 4 years. http://www.trelgol.com

What is NumPy?



BYU
Electrical Engineering
Computer Engineering

- 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.

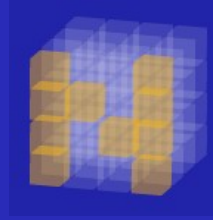
NumPy Array

BYU
Electrical Engineering
Computer Engineering

- A NumPy array is a homogeneous collection of "items" of the same "data-type" (dtype)

```
>>> import numpy as N
>>> a = N.array([[1,2,3],[4,5,6]],float)
>>> print a
[[1. 2. 3.]
 [4. 5. 6.]]
>>> print a.shape, "\n", a.itemsize
(2, 3)
8
>>> print a.dtype, a.dtype.type
'<f8' <type 'float64scalar'>
>>> type(a[0,0])
<type 'float64scalar'>
>>> type(a[0,0]) is type(a[1,2])
True
```

Memory Model



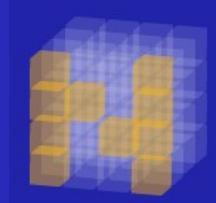
NumPy

BYU
Electrical Engineering
Computer Engineering

```
>>> print a.strides
(24, 8)
>>> print a.flags.fortran, a.flags.contiguous
False True
>>> print a.T.strides
(8, 24)
>>> print a.T.flags.fortran, a.T.flags.contiguous
True False
```

- Every dimension of a ndarray is accessed by stepping (striding) a fixed number of bytes through memory.
- If memory is contiguous, then the strides are “pre-computed” indexing-formulas for either Fortran-order (first-dimension varies the fastest), or C-order (last-dimension varies the fastest) arrays.

Array slicing (Views)



NumPy

BYU
Electrical Engineering
Computer Engineering

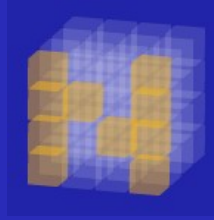
- Memory model allows “simple indexing” (integers and slices) into the array to be a **view** of the same data.

```
>>> b = a[:,::2]
>>> b[0,1] = 100
>>> print a
[[   1.    2.  100.]]
 [   4.    5.    6.]]
>>> c = a[:,::2].copy()
>>> c[1,0] = 500
>>> print a
[[   1.    2.  100.]]
 [   4.    5.    6.]]
```

Other uses of view

```
>>> b = a.view('i8')
>>> [hex(val.item()) for val in
b.flat]
['0x3FF0000000000000L',
 '0x4000000000000000L',
 '0x4059000000000000L',
 '0x4010000000000000L',
 '0x4014000000000000L',
 '0x4018000000000000L']
```

Data-types



NumPy

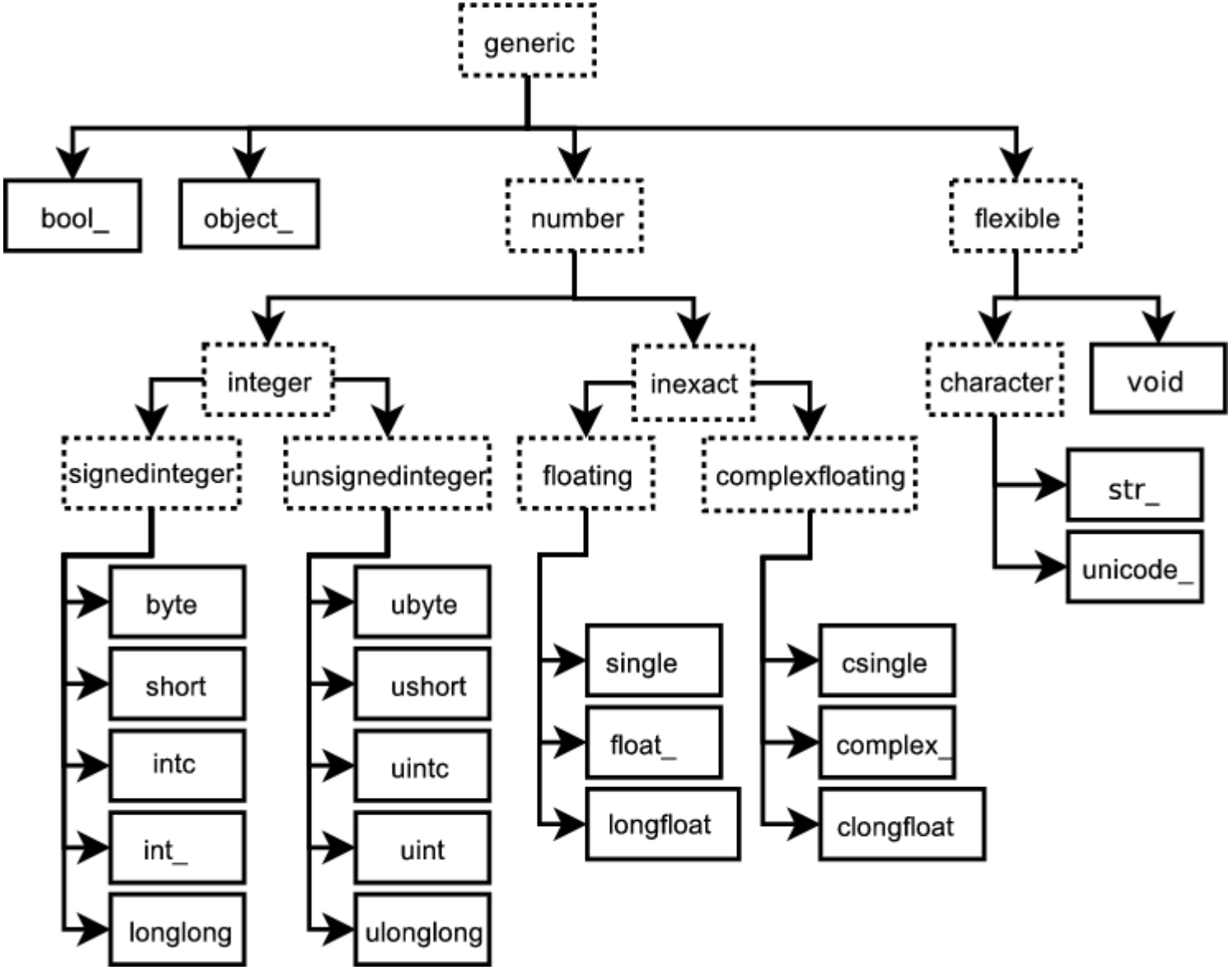
BYU
Electrical Engineering
Computer Engineering

- There are two related concepts of “type”
  - The data-type object (dtype)
  - The Python “type” of the object created from a single array item (hierarchy of scalar types)
- The **dtype** object provides the details of how to interpret the memory for an item. It's an instance of a single dtype class.
- The “type” of the extracted elements are true Python classes that exist in a hierarchy of Python classes (similar to Numarray).
- Every dtype object has a type attribute which provides the Python object returned when an element is selected from the array

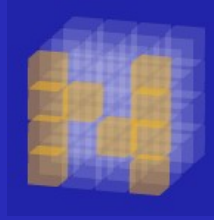
Built-in “scalar” types

NumPy

BYU
Electrical Engineering
Computer Engineering



Data-type object (dtype)

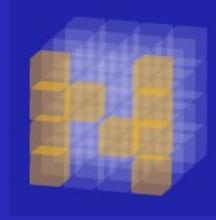


NumPy

BYU
Electrical Engineering
Computer Engineering

- There are 21 “built-in” (static) data-type objects
- New (dynamic) data-type objects are created to handle
  - Alteration of the byteorder
  - Change in the element size (for string, unicode, and void built-ins)
  - Addition of fields
  - Change of the type object (only allowed for sub-classes of the void-scalar)
- Creation of data-types is quite flexible.
- New user-defined “built-in” data-types can also be added (but must be done in C and involves filling a function-pointer table)

Data-type fields



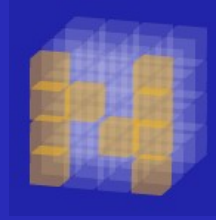
NumPy

BYU
Electrical Engineering
Computer Engineering

- 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.

```
>>> dt = N.dtype(“i4,f8,a5”)
>>> print dt.fields
{'f1': (dtype('<i4'), 0), 'f2': (dtype('<f8'), 4), 'f3':
(dtype('|S5'), 12)}
>>> a = N.array([(1,2.0,”Hello”), (2,3.0,”World”)],
dtype=dt)
>>> print a['f3']
[Hello World]
```

Array attributes



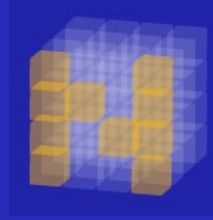
NumPy

BYU
Electrical Engineering
Computer Engineering

Attribute	Description
data	Buffer object representing memory
dtype	Data-type object
flags	Flags object (*e.g.* contiguous, aligned, writeable)
flat	1D iterator object
imag	Imaginary part or read-only zeros
real	Real part
T	Transpose view
base	Memory-exposing object
ctypes	Object for ctypes interfacing
itemsize	Bytes in each item
size	Number of items
nbytes	Number of bytes
ndim	Number of dimensions
shape	Tuple showing shape
strides	Tuple showing strides

bold : can be set

Array methods

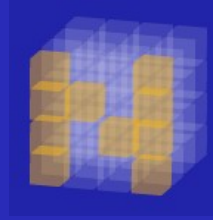


BYU
Electrical Engineering
Computer Engineering

Array Conversion

Method	Arguments	Description
astype	`(dtype <None>)`	Cast to another data type
byteswap	`(inplace <False>)`	Byteswap array elements
copy	`()`	Copy array
dump	`(file)`	Pickle to stream or file
dumps	`()`	Get pickled string
fill	`(scalar)`	Fill an array with scalar value
getfield	`(dtype=, offset=0)`	Return a field of the array
setflags	`(write=None, align=None, uic=None)`	Set array flags
tofile	`(file=, sep='', format='')`	Raw write to file
tolist	`()`	Array as a nested list
item	`()`	Python scalar from first element
tostring	`(order='C')`	String of raw memory
view	`(obj)`	View as another data type or class

Array methods



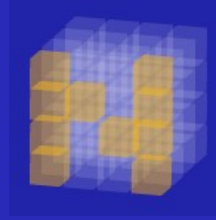
NumPy

BYU
Electrical Engineering
Computer Engineering

Item selection and shape manipulation

Method	Arguments	Description
argsort	`(axis=None, kind='quick')`	Indices showing how to sort array.
choose	`(c0, c1 , ..., cn, out=None, clip='raise')`	Choose from different arrays based on value of s
compress	`(condition=, axis=None, out=None)`	Elements of self where condition is true.
diagonal	`(offset=0, axis1=0, axis2=1)`	Return a diagonal from self.
flatten	`(order='C')`	A 1-d copy of self.
nonzero	`()`	True where self is not zero.
put	`(indices=, values=, mode='raise')`	Place values at 1-d index locations of self.
putmask	`(mask=, values=)`	Place values in 1-d index locations where mask
ravel	`(order='C')`	1-d version of self (no data copy if self is C-style
repeat	`(repeats=, axis=None)`	Repeat elements of self.
reshape	`(d1,d2,...,dn, order='C')`	Return reshaped version of self.
resize	`(d1,d2,...,dn, refcheck=1, order='Any')`	Resize self in-place.
searchsorted	`(values)`	Show where values would be placed in self (ass
sort	`(axis=None, kind='quick')`	Copy of self sorted along axis.
squeeze	`()`	Squeeze out all length-1 dimensions.
swapaxes	`(axis1, axis2)`	Swap two dimensions of self.
take	`(indices=, axis=None, out=None, mode='raise')`	Select elements of self along axis according to i

Array methods



NumPy

BYU
Electrical Engineering
Computer Engineering

Array Calculation

Method	Arguments	Description
all	(axis=None)	true if all entries are true.
any	(axis=None)	true if any entries are true.
argmax	(axis=None)	index of largest value.
argmin	(axis=None)	index of smallest value.
clip	(min=, max=)	self[self>max]=max; self[self<min]=min
conj	()	complex conjugate
cumprod	(axis=None, dtype=None)	cumulative product
cumsum	(axis=None, dtype=None)	cumulative sum
max	(axis=None)	maximum of self
mean	(axis=None, dtype=None)	mean of self
min	(axis=None)	minimum of self
prod	(axis=None, dtype=None)	multiply elements of self together
ptp	(axis=None)	self.max(axis)-self.min(axis)
var	(axis=None, dtype=None)	variance of self
std	(axis=None, dtype=None)	standard deviation of self
sum	(axis=None, dtype=None)	add elements of self together
trace	(offset, axis1=0, axis2=0, dtype=None)	Sum along a diagonal

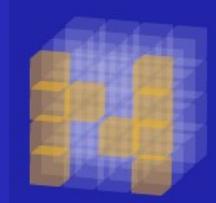
Universal Functions

BYU
Electrical Engineering
Computer Engineering

- ufuncs are objects that rapidly evaluate a function element-by-element over an array.
- Core piece is a 1-d loop written in C that performs the operation over the largest dimension of the array
- For 1-d arrays it is equivalent to but much faster than list comprehension

```
>>> type(N.exp)
<type 'numpy.ufunc'>
>>> x = array([1,2,3,4,5])
>>> print N.exp(x)
[   2.71828183    7.3890561    20.08553692   54.59815003
148.4131591 ]
>>> print [math.exp(val) for val in x]
[2.7182818284590451,
7.3890560989306504,20.085536923187668,
 54.598150033144236,148.4131591025766]
```

Broadcasting



NumPy

BYU
Electrical Engineering
Computer Engineering

- 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.

```
>>> x = [1,2,3,4];
>>> y = [[10],[20],[30]]
>>> print N.add(x,y)
[[11 12 13 14]
 [21 22 23 24]
 [31 32 33 34]]
>>> x = array(x)
>>> y = array(y)
>>> print x+y
[[11 12 13 14]
 [21 22 23 24]
 [31 32 33 34]]
```

x has shape (4,) the ufunc sees it as having shape (1,4)

y has shape (3,1)

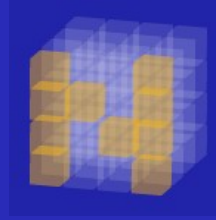
The ufunc result has shape (3,4)

Available ufuncs

BYU
Electrical Engineering
Computer Engineering

absolute	equal	less_equal	remainder
add	exp	log	right_shift
arccos	**expm1**	log10	**rint**
arccosh	fabs	**log1p**	**sign**
arcsin	floor	logical_and	**signbit**
arcsinh	floor_divide	logical_not	sin
arctan	fmod	logical_or	sinh
arctan2	**frexp**	logical_xor	sqrt
arctanh	greater	maximum	**square**
bitwise_and	greater_equal	minimum	subtract
bitwise_or	hypot	mod	tan
bitwise_xor	invert	**modf**	tanh
ceil	**isfinite**	multiply	true_divide
conj	**isinf**	negative	
conjugate	**isnan**	not_equal	
cos	**ldexp**	**ones_like**	
cosh	left_shift	power	
divide	less	**reciprocal**	

Array Interface

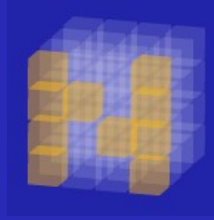


BYU
Electrical Engineering
Computer Engineering

- How do different Python modules share array information?
  - Put NumPy in the Python standard library?
  - Require installation of NumPy?
  - **Use the array interface**

http://numeric.scipy.org/array_interface.html

Demo



NumPy

BYU

Electrical Engineering
Computer Engineering

- Array creation
- Array math
- FFT
- Eigen-decomposition
- Random-number generation
- f2py

Other Tools

BYU
Electrical Engineering
Computer Engineering

- http://www.scipy.org