Python for Astronomers

numpy

numpy

A Python package that adds support for large, multi-dimensional arrays and matrices.

numpy

- Why do we want to use an array ?
  - PERFORMANCE !!!
  - Convenient handling of numerical data

From Numeric to numpy

- First package: “Numeric”
  - 1995 by Jim Hugunin and others
- Later: “numarray”
  - developed at STScI
- Now: “**numpy**”
  - merging of 'Numeric' and 'numarray'

Array vs. list

- Add Elements of a list; doing it the pure Python way:

```
>>> l1 = [1,2,3]
>>> l2 = [4,5,6]
>>> print l1 + l2
>>> l3 = [0]*3
>>> for i in range(3):
...     l3[i] = l1[i] + l2[i]
>>> print l3
```

Array vs. list

- What we would like to do:

```
>>> l1 = [1,2,3]
>>> l2 = [4,5,6]
>>> # do some magic stuff here !!!
>>> print l1+l2 #should print out
[5,7,9]
>>>
```

- This is where arrays come into play

Creating arrays

- New Container datatype **array**

```
>>> import numpy
>>> l = [1,2,3]
>>> a = numpy.array(l)
>>> print l
>>> print a
>>> a
```

Creating arrays

- WARNING !!!

```
>>> import numpy
>>> a = numpy.array([1,2,3])
>>> a = numpy.array(1,2,3)
```

Array basics

- Now repeat the previous example

```
>>> l1 = [1,2,3]
>>> l2 = [4,5,6]
>>> a1 = numpy.array(l1)
>>> a2 = numpy.array(l2)
>>> print l1+l2
>>> print a1+a2
```

Array basics

- Operators supported by numpy arrays

```
>>> a1+a2 # addition
>>> a1-a2 # substraction
>>> a1*a2 # multiplication
>>> a1/a2 # division
>>> a2%a1 # modulo
>>> a1**2 # elementwise squared
>>> a1+4, a1*3, a1/2
```

Creating array (2)

```
>>> l1 = [1, 2, 3]
>>> l2 = [1, 2, 2.5]
>>> a1 = numpy.array(l1)
>>> a2 = numpy.array(l2)
>>> print l1
>>> print l2
>>> print a1
>>> print a2
```

Arrays

A numpy array

“is a table of elements (usually numbers),

all of the **same** type”.

Creating arrays (3)

```
>>> # implicit declaration of
>>> # the datatype
>>>
>>> ai = numpy.array([1, 2, 3])
>>> af = numpy.array([1, 2, 3.])
>>> ac = numpy.array([1j, 2, 3.])
>>> print ai
>>> print af
>>> print ac
```

Creating arrays (4)

```
>>> # explicit declaration of
>>> # the datatype with standard
>>> # Python types
>>>
>>> af = numpy.array([1,2,3], float)
>>> ac = numpy.array([1,2,3], complex)
>>> print af
>>> print ac
```

Creating arrays (5)

```
>>> # explicit declaration of
>>> # the datatype with numpy types
>>>
>>> af=numpy.array([1,2], numpy.float)
>>> print af
```

Array datatypes

```
>>> # check the datatype of an array
>>>
>>> af=numpy.array([1,2], numpy.float)
>>> print af.dtype    # dtype attribute
>>>
```

Array datatypes

- Numpy knows the difference between “float” & “double” (and “long double”)
  - numpy.float (== Python float type)
  - numpy.float32 (single precission)
  - numpy.float64 (double precission)
  - numpy.double (synonym, double prec.)
  - *numpy.longdouble (even more platform dependent !!! may be: numpy.float96 or numpy.float128)*
- Trailing number gives number of bits

Array datatypes

- Integer datatypes
  - numpy.int8 ...int16 ...int32 ...int64
- Unsigned (!) integer datatypes
  - numpy.uint8 ...uint16 ...uint32 ...int64
- Complex datatypes
  - numpy.complex64 ...complex128 ...complex256

Creating arrays

- numpy has some special functions to create arrays:

```
>>> z = numpy.zeros(3)
>>> print z
>>> o = numpy.ones(7)
>>> print o
>>> e = numpy.empty(10)
>>> print e    # uninitialised values !
```

Multidimensional arrays

```
>>> l2 = [[1,2,3],[4,5,6]]
>>> a2 = numpy.array(l2)
>>> print l2
>>> print a2
>>>
>>> l3 = [[1,2,3],[4,5]]
>>> a3 = numpy.array(l3)
```

Multidimensional arrays

```
>>> # accessing elements:
>>> # indexing and slicing
>>> print l2[0][1]
>>> print a2[0][1]
>>>
>>> print a2[0,1]  # works only with
>>> print a2[0,1:] # arrays !!!
>>> print a2[:,0]  #
```

Multidimensional arrays

```
>>> # ATTENTION !!!
>>> # different behaviour of slicing
>>> a = numpy.array([[0,1,2],[3,4,5]])
>>> print a[1:][1:]
>>> print a[1:,1:]
```

Multidimensional arrays

```
>>> # determine the dimensions
>>> # remember: l2=[[1,2,3],[4,5,6]]
>>> # how many rows, how many columns ?
>>> print len(l2), len(l2[0])
>>> print len(a2), len(a2[0])
>>>
>>> print a2.shape   # shape attribute
```

Multidimensional arrays

```
>>> # number of dimensions
>>> print len(a2.shape)
>>> print a2.ndim  # ndim attribute
>>> # = the number of axes

>>> print a2.size  # size attribute
>>> print a2.shape[0] * a2.shape[1]
```

Array Attributes

- array.ndim: number of dimensions
- array.shape: the dimensions
- array.size: total number of elements
- array.dtype: type of the data

Multidimensional arrays

```
>>> # numpy.zeros, numpy.ones, et al.
>>> # can be used to create multidim
>>> # arrays, too !
>>> o = numpy.ones( (3,10) )
>>> e = numpy.empty( (4,2,19), float)
```

The dot product / matrixmultiplication

```
>>> v = numpy.ones(3)
>>> M = numpy.array([[-1,0,0],[.5,1,0],
... [.25,0,1]], float)
>>> print 'M \dot v ='
>>> print numpy.dot(M,v)
```

The dot product / matrixmultiplication

```
>>> M2 = numpy.array([[1,2],
...       [3,4],[5,6]])
>>> print numpy.dot(M2,v)
>>>
>>> M3 = numpy.array([[1,2,3],[4,5,6]])
>>> print numpy.dot(M3,v)
```

The dot product / matrixmultiplication

```
>>> v2 = numpy.array([[1,2,3],])
>>> print numpy.dot(M,v2)
>>>
>>> v3 = numpy.array([[1],[2],[3]])
>>> print numpy.dot(M,v3)
```

Excercises

- Try to achieve the same result as in the second Excercise of the file I/O section, but now using numpy arrays:
  Determine the sum of columns 1 and 2 from /home/rschaaf/test.dat and write the output to a file.