

Python for Data Analysis and Scientific Computing

X433.3 (2 semester units in COMPSCI)

Instructor Alexander I. Iliev, Ph.D.

NumPy arrays

- NumPy arrays
 - array attributes

T	Same as self.transpose(), except that self is returned if self.ndim < 2.
data	Python buffer object pointing to the start of the array's data.
dtype	Data-type of the array elements.
flags	Information about the memory layout of the array.
flatten	A 1-D iterator over the array.
imag	The imaginary part of the array.
real	The real part of the array.
size	Number of elements in the array.
itemsize	Length of one array element in bytes.
nbytes	Total bytes consumed by the elements of the array.
ndim	Number of array dimensions.
shape	Tuple of array dimensions.
strides	Tuple of bytes to step in each dimension when traversing an array.
ctypes	An object to simplify the interaction of the array with the ctypes module.
base	Base object if memory is from some other object.

*source – NumPy reference

NumPy arrays

- NumPy arrays

Examples:

... try it in class

```
Python
In [43]: f = np.ndarray(shape=(2,3,2), dtype=complex)

In [44]: f
Out[44]:
array([[[ 0.00000000e+000 -2.00000013e+000j,
          2.12215769e-314 +9.88131292e-324j],
        [ 0.00000000e+000 +0.00000000e+000j,
          0.00000000e+000 -9.84629069e+109j],
        [ 0.00000000e+000 +0.00000000e+000j,
          2.25697366e-314 +0.00000000e+000j]],
       [[ 0.00000000e+000 +2.25697468e-314j,
          0.00000000e+000 +0.00000000e+000j],
        [ -2.58861351e-056 +0.00000000e+000j,
          0.00000000e+000 -2.05241193e-191j],
        [ 2.12381808e-314 +2.25685768e-314j,
          -4.57473710e+035 +2.24500133e-314j]])

In [45]: f.real
Out[45]:
array([[[ 0.00000000e+000,  2.12215769e-314],
        [ 0.00000000e+000,  0.00000000e+000],
        [ 0.00000000e+000,  2.25697366e-314]],
       [[ 0.00000000e+000,  0.00000000e+000],
        [ -2.58861351e-056,  0.00000000e+000],
        [ 2.12381808e-314, -4.57473710e+035]])

In [46]: f.real.T
Out[46]:
array([[[ 0.00000000e+000,  0.00000000e+000],
        [ 0.00000000e+000, -2.58861351e-056],
        [ 0.00000000e+000,  2.12381808e-314]],
       [[ 2.12215769e-314,  0.00000000e+000],
        [ 0.00000000e+000,  0.00000000e+000],
        [ 2.25697366e-314, -4.57473710e+035]])
```

NumPy arrays

- NumPy arrays

Examples:

Note - it can be seen that the attributes of `ndarray` can be used in a nested fashion

... try it in class

```
Python
In [47]: f.imag.flags
Out[47]:
C_CONTIGUOUS : False
F_CONTIGUOUS : False
OWNDATA : False
WRITEABLE : True
ALIGNED : True
UPDATEIFCOPY : False

In [48]: f.imag.data
Out[48]: <memory at 0x110298ce0>

In [49]: f.real.dtype
Out[49]: dtype('float64')

In [50]: f.dtype
Out[50]: dtype('complex128')

In [51]: f.shape
Out[51]: (2, 3, 2)

In [52]: f.T.shape
Out[52]: (2, 3, 2)

In [53]: f.size
Out[53]: 12

In [54]: f.itemsize
Out[54]: 16

In [55]: f.nbytes
Out[55]: 192

In [56]: f.ndim
Out[56]: 3
```

NumPy arrays

- NumPy arrays
 - **flags** – gives information about the memory layout of the array

C_CONTIGUOUS (C)	The data is in a single, C-style contiguous segment .
F_CONTIGUOUS (F)	The data is in a single, Fortran-style contiguous segment .
OWNDATA (O)	The array owns the memory it uses or borrows it from another object.
WRITEABLE (W)	The data area can be written to. Setting this to False locks the data, making it read-only. A view (slice, etc.) inherits WRITEABLE from its base array at creation time, but a view of a writeable array may be subsequently locked while the base array remains writeable. (The opposite is not true, in that a view of a locked array may not be made writeable. However, currently, locking a base object does not lock any views that already reference it , so under that circumstance it is possible to alter the contents of a locked array via a previously created writeable view onto it.) Attempting to change a non-writeable array raises a RuntimeError exception.
ALIGNED (A)	The data and all elements are aligned appropriately for the hardware .
UPDATEIFCOPY (U)	This array is a copy of some other array . When this array is de-allocated, the base array will be updated with the contents of this array.
FNC	F_CONTIGUOUS and not C_CONTIGUOUS.
FORC	F_CONTIGUOUS or C_CONTIGUOUS (one-segment test).
BEHAVED (B)	ALIGNED and WRITEABLE.
CARRAY (CA)	BEHAVED and C_CONTIGUOUS.
FARRAY (FA)	BEHAVED and F_CONTIGUOUS and not C_CONTIGUOUS.

*source – NumPy reference

NumPy arrays

- NumPy arrays
 - `flatten` – returns a copy of the same **flattened** array **in one dimension**

```
Python
In [57]: g = np.arange(12, 24).reshape(3, 4)

In [58]: g
Out[58]:
array([[12, 13, 14, 15],
       [16, 17, 18, 19],
       [20, 21, 22, 23]])

In [59]: g[:, :]
Out[59]:
array([[12, 13, 14, 15],
       [16, 17, 18, 19],
       [20, 21, 22, 23]])

In [60]: g.flat[6]
Out[60]: 18

In [61]: g.flat[9]
Out[61]: 21

In [62]: g.flat[:]
Out[62]: array([12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23])

In [63]: g.flatten()
Out[63]: array([12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23])

In [64]: g.T.flat[6]
Out[64]: 14

In [33]: g.flatten(order='C')
Out[33]: array([12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23])

In [34]: g.flatten(order='F')
Out[34]: array([12, 16, 20, 13, 17, 21, 14, 18, 22, 15, 19, 23])

In [18]: g.T
Out[18]:
array([[12, 16, 20],
       [13, 17, 21],
       [14, 18, 22],
       [15, 19, 23]])
```

... try it in class

NumPy arrays

- NumPy arrays
 - **shape** – besides checking or specifying the shape of an array, by using the **shape** command we can also **re-shape** an array so long that we do not change the number of elements in it

Example:

```
Python
In [65]: h = np.array([[12,34,41],[54,67,89],[102,13,45],[78,456,218]])

In [66]: h
Out[66]:
array([[ 12,  34,  41],
       [ 54,  67,  89],
       [102,  13,  45],
       [ 78, 456, 218]])

In [67]: h.shape
Out[67]: (4, 3)

In [68]: h.shape = (2,6)

In [69]: h
Out[69]:
array([[ 12,  34,  41,  54,  67,  89],
       [102,  13,  45,  78, 456, 218]])

In [70]: h.shape = (3,6)

-----
ValueError                                Traceback (most recent call last)
<ipython-input-70-76a181f81944> in <module>()
----> 1 h.shape = (3,6)

ValueError: total size of new array must be unchanged
```

NumPy arrays

- NumPy arrays
 - **strides** – represents the number of bytes (8-bit each) **needed to travel in each direction** (in memory) in a **multidimensional array** in order to get to certain element in that array along a given axis

Example:

Note – the given array *i* is stored in a **continuous block of memory** of:

480 bytes, or:

```
In [154]: (3*4*5)*8
Out[154]: 480
```

```
Python
In [71]: i = np.reshape(np.arange(3*4*5), (5,3,4))

In [72]: i
Out[72]:
array([[[ 0,  1,  2,  3],
        [ 4,  5,  6,  7],
        [ 8,  9, 10, 11]],

       [[12, 13, 14, 15],
        [16, 17, 18, 19],
        [20, 21, 22, 23]],

       [[24, 25, 26, 27],
        [28, 29, 30, 31],
        [32, 33, 34, 35]],

       [[36, 37, 38, 39],
        [40, 41, 42, 43],
        [44, 45, 46, 47]],

       [[48, 49, 50, 51],
        [52, 53, 54, 55],
        [56, 57, 58, 59]])

In [73]: np.shape(i)
Out[73]: (5, 3, 4)

In [111]: i.shape
Out[111]: (5, 3, 4)

In [139]: i.nbytes
Out[139]: 480

In [140]: i[0]
Out[140]:
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])

In [141]: i[0].nbytes
Out[141]: 96

In [142]: i[0][0]
Out[142]: array([0, 1, 2, 3])

In [143]: i[0][0].nbytes
Out[143]: 32

In [144]: i[0][0][0]
Out[144]: 0

In [145]: i[0][0][0].nbytes
Out[145]: 8
```

same as:

NumPy arrays

- NumPy arrays

- strides

Example:

Note – you can easily refer to an element from the array, knowing its position as shown in lines `Out [74],[75]`

... try it in class

```
Python
In [74]: i[4][2][1]
Out[74]: 57

In [75]: i[4,2,1]
Out[75]: 57

In [76]: np.dtype(i[4,2,1])
Out[76]: dtype('int64')

In [77]: i
Out[77]:
array([[[ 0,  1,  2,  3],
        [ 4,  5,  6,  7],
        [ 8,  9, 10, 11]],

       [[12, 13, 14, 15],
        [16, 17, 18, 19],
        [20, 21, 22, 23]],

       [[24, 25, 26, 27],
        [28, 29, 30, 31],
        [32, 33, 34, 35]],

       [[36, 37, 38, 39],
        [40, 41, 42, 43],
        [44, 45, 46, 47]],

       [[48, 49, 50, 51],
        [52, 53, 54, 55],
        [56, 57, 58, 59]]])

In [78]: i.strides
Out[78]: (96, 32, 8)
```

NumPy arrays

- NumPy arrays

- strides

Example:

Note – ... you can calculate it in an iterative way shown in line

Out[83]

... try it in class

```
Python
In [79]: np.array([4,2,1])
Out[79]: array([4, 2, 1])

In [80]: np.array([4,2,1]) * i.strides
Out[80]: array([384, 64, 8])

In [81]: sum(np.array([4,2,1]) * i.strides)
Out[81]: 456

In [82]: i.itemsize
Out[82]: 8

In [83]: sum(np.array([4,2,1]) * i.strides)/i.itemsize
Out[83]: 57.0

In [84]: i
Out[84]: array([[ 0,  1,  2,  3],
                [ 4,  5,  6,  7],
                [ 8,  9, 10, 11]],

               [[12, 13, 14, 15],
                [16, 17, 18, 19],
                [20, 21, 22, 23]],

               [[24, 25, 26, 27],
                [28, 29, 30, 31],
                [32, 33, 34, 35]],

               [[36, 37, 38, 39],
                [40, 41, 42, 43],
                [44, 45, 46, 47]],

               [[48, 49, 50, 51],
                [52, 53, 54, 55],
                [56, 57, 58, 59]])

In [166]: np.array([4,2,1])*i.strides/i.itemsize
Out[166]: array([48.,  8.,  1.])
```

NumPy arrays

- NumPy arrays
 - **transpose** – transpose can easily be performed by using a specific attribute (command)

Example:

note:

.T and .transpose()

do the same job!

... try it in class

```
Python
In [85]: i.transpose
Out[85]: <function ndarray.transpose>

In [86]: i.transpose()
Out[86]:
array([[ 0, 12, 24, 36, 48],
       [ 4, 16, 28, 40, 52],
       [ 8, 20, 32, 44, 56]],

       [[ 1, 13, 25, 37, 49],
       [ 5, 17, 29, 41, 53],
       [ 9, 21, 33, 45, 57]],

       [[ 2, 14, 26, 38, 50],
       [ 6, 18, 30, 42, 54],
       [10, 22, 34, 46, 58]],

       [[ 3, 15, 27, 39, 51],
       [ 7, 19, 31, 43, 55],
       [11, 23, 35, 47, 59]])

In [87]: np.shape(i.transpose())
Out[87]: (4, 3, 5)
```

NumPy with other languages

- NumPy arrays
 - ctypes:
 - this module is part of the **standard Python** distribution package
 - it is used for **shared C-libraries**, in case you have some useful code written in C and would like to put a **Python wrapper around** it to incorporate a specific routine written in C in your code
 - this possibility **opens up** a great number of already well written and tested **C routines**
 - the **problem** when using this module however is that it can lead to **nasty crashes** because of **poor type checking**

Example:

a problem can occur when you **pass an array as a pointer to a raw memory location** and you forget to check if the subroutine may **access memory outside of the array boundaries**

NumPy with other languages

- NumPy arrays
 - ctypes:
 - when using *ctypes* remember that this approach **uses a raw memory location** to a compiled code and it **may not be error prone** to user mistakes
 - **good knowledge of the shared library** and this module is a must
 - this approach most times **requires extra Python code to handle errors** of different kind to:
 - check for the **data types**
 - **array boundaries** of the passes objects
 - this however **will slow down the interface** because of all additional checking and type conversion (C to Python) that is necessary
 - this tool is **for people with strong Python skills**, but weak C knowledge

NumPy with other languages

- NumPy arrays
 - *ctypes*:
 - to use *ctypes* **you must have** the following:
 - have **a library** to be shared
 - **load the library** to be shared
 - **convert the Python objects to *ctypes*** arguments that can be interpreted correctly
 - **call the function from the library** containing the *ctypes* arguments
 - when using *ctypes* some of the basic attributes that can be used are:
 - **data**, **shape** and **strides** (... for more attributes please refer to the NumPy guide)
 - one should be careful when **using temporary arrays** or arrays constructed on the fly, because they return a pointer to **an invalid memory location** since it has been **de-allocated** as soon as the next Python statement is reached

Examples:

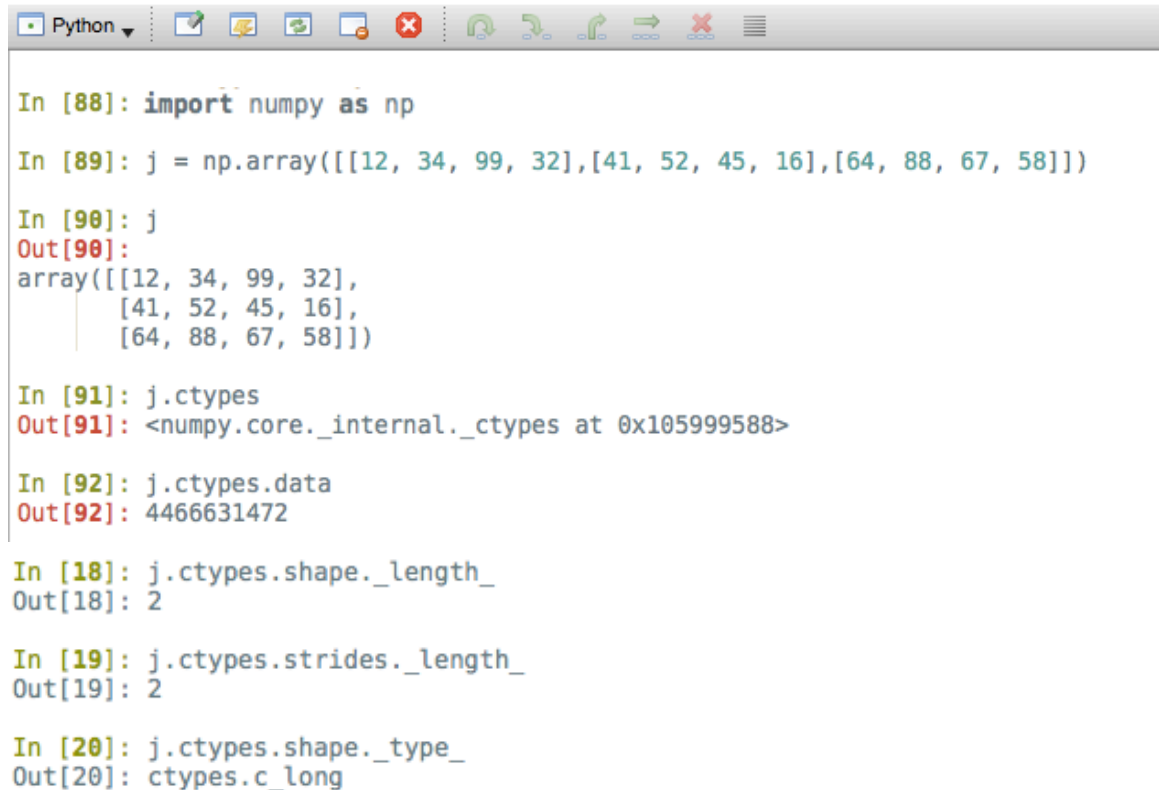
- a) `(a+b).ctypes` – wrong , because the array created as **(a+b) is de-allocated** before the next statement
- b) `c = (a+b).ctypes` – correct, because **c will have a reference** to the array

NumPy with other languages

- NumPy arrays

- ctypes:

Examples:



```
In [88]: import numpy as np

In [89]: j = np.array([[12, 34, 99, 32],[41, 52, 45, 16],[64, 88, 67, 58]])

In [90]: j
Out[90]:
array([[12, 34, 99, 32],
       [41, 52, 45, 16],
       [64, 88, 67, 58]])

In [91]: j.ctypes
Out[91]: <numpy.core._internal._ctypes at 0x105999588>

In [92]: j.ctypes.data
Out[92]: 4466631472

In [18]: j.ctypes.shape._length_
Out[18]: 2

In [19]: j.ctypes.strides._length_
Out[19]: 2

In [20]: j.ctypes.shape._type_
Out[20]: ctypes.c_long
```

... try it in class

NumPy with other languages

- NumPy arrays

- **ctypes**: Example:

1. begin with **writing your C library** and save the file 'ctypes_lib.c':

```
1  #include <stdio.h>
2
3  void myprint(void);
4  void myprint()
5  {
6      printf("This is ctypes example in Python\n");
7  }
```

2. **install your gcc** if you don't have one (skip this step if you do):

- PC: find a compiler and install using the .exe file. Try using **Cygwin** - a Unix-like environment on Win
- Mac OS X in the terminal type: **xcode-select --install**

3. you need to **compile the file as shared library** using this notation:

- PC: `$ gcc -shared -Wl,-soname, ctypes_lib -o ctypes_lib.so -fPIC ctypes_lib.c`
- Mac OS X: `$ gcc -shared -Wl,-install_name, ctypes_lib.so -o ctypes_lib.so -fPIC ctypes_lib.c`

```
Macintosh:lecture4 alex$ gcc -shared -Wl,-install_name,ctypes_lib.so -o ctypes_lib.so -fPIC ctypes_lib.c
Macintosh:lecture4 alex$
```


NumPy with other languages

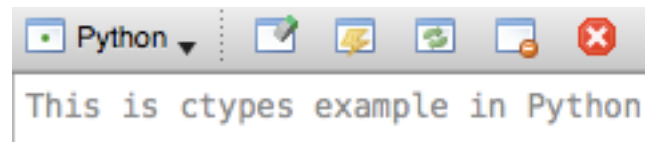
- NumPy arrays

- `ctypes`: Example:

4. Create your `ctypes` Python wrapper module `'ctypes_lib_tester.py'` and execute it:

```
1  ## Ctypes example of using a C file code:
2
3  import ctypes
4
5  c_test_lib = ctypes.CDLL('ctypes_lib.so')
6  c_test_lib.myprint()
```

5. The result should be:



A screenshot of a Python terminal window. The title bar shows a 'Python' dropdown and several icons. The terminal output displays the text 'This is ctypes example in Python'.

6. If you run:

```
In [1]: c_test_lib.myprint()
Out[1]: 33
```

this only **prints the number of characters** in the 'c' library, so for the text:

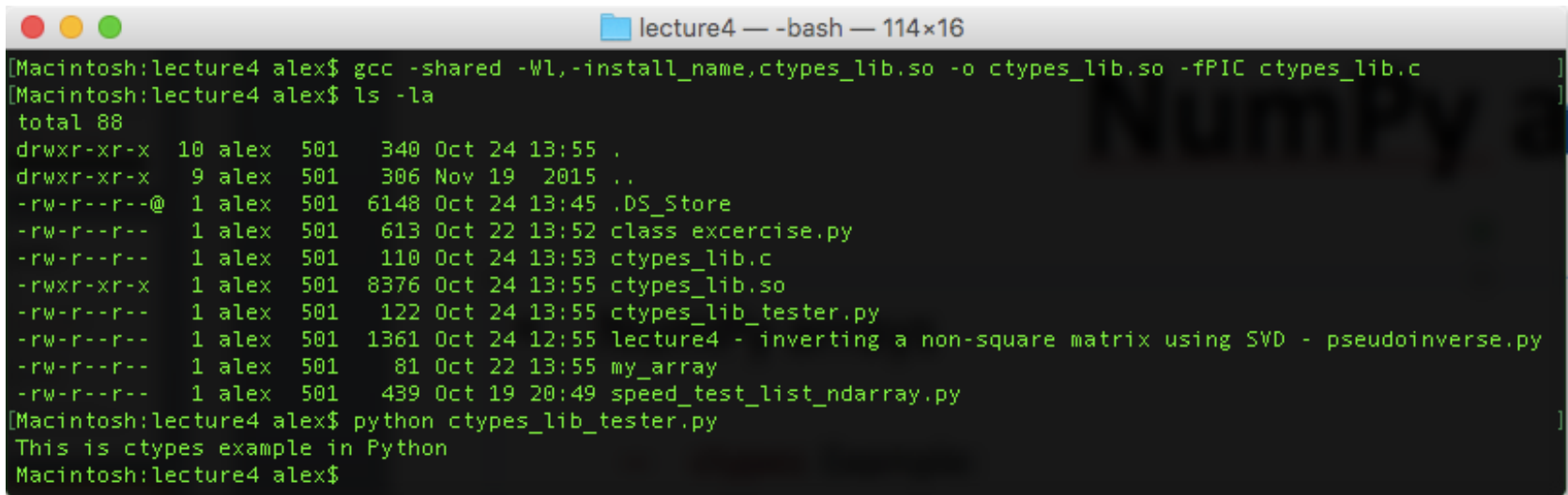
"This is ctypes example in Python\n" there are 33, including the end of line character '\n'

NumPy with other languages

- NumPy arrays

- **ctypes**: Example:

7. If you **compile** and **execute** the library in the terminal here is the result:



```
lecture4 — -bash — 114x16
[Macintosh:lecture4 alex$ gcc -shared -Wl,-install_name,ctypes_lib.so -o ctypes_lib.so -fPIC ctypes_lib.c ]
[Macintosh:lecture4 alex$ ls -la ]
total 88
drwxr-xr-x  10 alex  501   340 Oct 24 13:55 .
drwxr-xr-x   9 alex  501   306 Nov 19  2015 ..
-rw-r--r--@  1 alex  501  6148 Oct 24 13:45 .DS_Store
-rw-r--r--   1 alex  501   613 Oct 22 13:52 class_excercise.py
-rw-r--r--   1 alex  501   110 Oct 24 13:53 ctypes_lib.c
-rwxr-xr-x   1 alex  501  8376 Oct 24 13:55 ctypes_lib.so
-rw-r--r--   1 alex  501   122 Oct 24 13:55 ctypes_lib_tester.py
-rw-r--r--   1 alex  501  1361 Oct 24 12:55 lecture4 - inverting a non-square matrix using SVD - pseudoinverse.py
-rw-r--r--   1 alex  501    81 Oct 22 13:55 my_array
-rw-r--r--   1 alex  501   439 Oct 19 20:49 speed_test_list_ndarray.py
[Macintosh:lecture4 alex$ python ctypes_lib_tester.py ]
This is ctypes example in Python
Macintosh:lecture4 alex$
```

NumPy with other languages






C/C++

There are various tools which make it easier to bridge the gap between Python and C/C++:

- » [Pyrex](#) - write your extension module on Python 💡
- » [Cython](#) -- Cython -- an improved version of Pyrex
- » [CXX](#) - PyCXX - helper lib for writing Python extensions in C++
- » [ctypes](#) is a Python module allowing to create and manipulate C data types in Python. These can then be passed to C-functions loaded from dynamic link libraries.
- » [elmer](#) - compile and run python code from C, as if it was written in C
- » [PicklingTools](#) is a collection of libraries for exchanging Python Dictionaries between C++ and Python.
- » [weave](#) - include C code lines in Python program (deprecated in favor of Cython)
- » [ackward](#) exposes parts of Python's standard library as idiomatic C++
- » [CFFI](#) - interact with almost any C code from Python, based on C-like declarations that you can often copy-paste from header files or documentation.

NumPy with other languages

Java

- » [Jython](#) - Python implemented in Java
- » [JPytype](#) - Allows Python to run java commands
- » [Jepp](#) - Java embedded Python
- »  [JCC](#) - a C++ code generator for calling Java from C++/Python
- »  [Javabridge](#) - a package for running and interacting with the JVM from CPython
- »  [py4j](#) - Allows Python to run java commands.
- »  [voc](#) - Part of [BeeWare](#) suite. Converts python code to Java bytecode.
- »  [p2j](#) - Converts Python code to Java. No longer developed.

NumPy with other languages


Perl

See  http://www.faqs.com/knowledge_base/view.phtml/aid/17202/fid/1102

» [PyPerl](#)  <http://search.cpan.org/dist/pyperl/>


»  [Inline::Python](#)

» [PyPerlish](#) - Perl idioms in Python

For converting/porting Perl code to Python the tool 'Bridgekeeper'  <http://www.crazy-compilers.com/bridgekeeper/> may be handy.

PHP

» [PiP \(Python in PHP\)](#)  <http://www.csh.rit.edu/~jon/projects/pip/>

» [PHP "Serialize" in Python](#)  <http://hurring.com/scott/code/python/serialize/> (broken link; see the  [Web Archive Wayback Machine](#) for the latest working version)

R

» [RPy](#)  <http://rpy.sourceforge.net>

» [RSPython](#)  <http://www.omegahat.net/RSPython>

NumPy arrays

- NumPy arrays

- base:

- is an **attribute** used when we need to keep **track of the memory reference of the original object owner** in case two objects are **referring to the same memory location**
 - it is a way of NumPy to keep track of the data source in memory for any given array

Example:

```
Python
In [95]: k = np.array([[12, 34, 99],[41, 52, 45],[64, 88, 67]])

In [96]: k
Out[96]:
array([[12, 34, 99],
       [41, 52, 45],
       [64, 88, 67]])

In [97]: k.base is None
Out[97]: True

In [98]: l = k[0:2]

In [99]: l
Out[99]:
array([[12, 34, 99],
       [41, 52, 45]])

In [100]: l.base is k
Out[100]: True
```

... try it

Indexing and slicing of arrays

- Indexing and slicing of arrays (recap)
 - indexing:
 - so far we saw indexing of arrays in many occasions
 - indexing in NumPy arrays **begins from '0'** and is runs in an scalar/integer progression
 - slicing:
 - a slice is an object **referring to a portion (slice) of an array**
 - the newly created slice is an array generated by **slicing the original array** and it always **represents a view of the latter**

Indexing and slicing of arrays

- Indexing and slicing of arrays

Examples:

```
Python
In [101]: m = np.array([[12, 34, 99],[41, 52, 45],[64, 88, 67]])

In [102]: m
Out[102]:
array([[12, 34, 99],
       [41, 52, 45],
       [64, 88, 67]])

In [103]: m.shape
Out[103]: (3, 3)

In [104]: m[0:]      # slicing
Out[104]:
array([[12, 34, 99],
       [41, 52, 45],
       [64, 88, 67]])

In [105]: m[1]       # indexing/slicing
Out[105]: array([41, 52, 45])

In [106]: m[1:2]     # slicing
Out[106]: array([[41, 52, 45]])

In [107]: m[1][0]    # indexing
Out[107]: 41

In [108]: m[:3]      # slicing
Out[108]:
array([[12, 34, 99],
       [41, 52, 45],
       [64, 88, 67]])

In [109]: m[2,1]     # indexing
Out[109]: 88

In [110]: m[-1,-3]   # slicing - representing element (-1+3=2, -3+3=0)
Out[110]: 64
```

... try it in class

Numpy - matrix, array and ndarray

- What are **matrix**, **array** and **ndarray** in NumPy

```
1  ## Recap: What are matrix, array and ndarray in NumPy:
2  from numpy import matrix, array, ndarray, int16, dot
3
4  # Arrays should be constructed using `array`, `zeros` or `empty`:
5  A = array([[2,3,4],[4,5,6]])
6
7  # Construct and assign a matrix:
8  B = matrix([[2,3,4],[4,5,6]])
9
10 # Construct an empty array:
11 C = ndarray([2,3], dtype=int16)
12
13 # Assign:
14 C[0,:] = A[0,:]
15 C[1,:] = B[1,:]
16
17 A*A
18 B*B.T
19 C*C
20
21 # To get matrix multiplication of an ndarray:
22 dot(A,A.T)
23 dot(C,C.T)
```

Numpy - matrix, array and ndarray

- What are **matrix**, **array** and **ndarray** in NumPy

```
In [1]: from numpy import matrix, array, ndarray, int16, dot
```

```
In [2]: A = array([[2,3,4],[4,5,6]])
```

```
In [3]: B = matrix([[2,3,4],[4,5,6]])
```

```
In [4]: C = ndarray([2,3], dtype=int16)
```

```
In [5]: A
```

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

```
In [6]: B
```

```
Out[6]:  
matrix([[2, 3, 4],  
        [4, 5, 6]])
```

```
In [7]: C
```

```
Out[7]:  
array([[ 0,  0,  0],  
       [ 0, 15653, 4166]], dtype=int16)
```

```
In [8]: C[0,:] = A[0,:]
```

```
In [9]: C[1,:] = B[1,:]
```

```
In [10]: C
```

```
Out[10]:  
array([[2, 3, 4],  
       [4, 5, 6]], dtype=int16)
```

```
In [11]: type(A)
```

```
Out[11]: numpy.ndarray
```

```
In [12]: type(C)
```

```
Out[12]: numpy.ndarray
```

```
In [13]: A*A
```

```
Out[13]:  
array([[ 4,  9, 16],  
       [16, 25, 36]])
```

```
In [14]: B*B.T
```

```
Out[14]:  
matrix([[29, 47],  
        [47, 77]])
```

```
In [15]: C*C
```

```
Out[15]:  
array([[ 4,  9, 16],  
       [16, 25, 36]], dtype=int16)
```

```
In [16]: dot(A,A.T)
```

```
Out[16]:  
array([[29, 47],  
       [47, 77]])
```

```
In [17]: dot(C,C.T)
```

```
Out[17]:  
array([[29, 47],  
       [47, 77]], dtype=int16)
```

Discussion

- Discussion

matrix inversion:

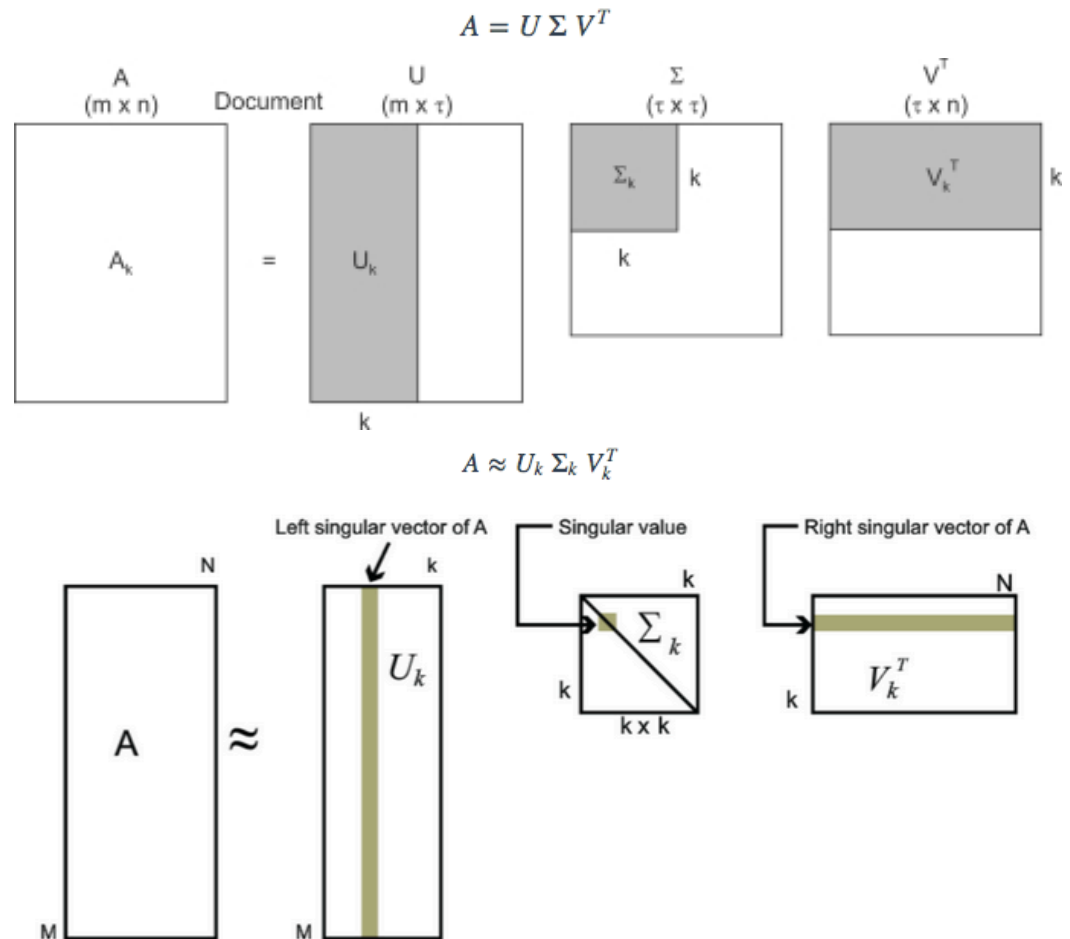
- by definition a matrix is **commutative** with its inverse on multiplication:

$$A_{[m \times n]} * A_{[n \times m]}^{-1} = A_{[n \times m]}^{-1} * A_{[m \times n]} = I$$

so, it must be that $m=n$!

- A non-square matrix inverse is possible using SVD:

There exists a **left inverse** U and a **right inverse** V that is defined for **all** matrices including non-square matrices



Discussion

- Discussion

matrix inversion:

1. by definition a matrix is

commutative with its

inverse on multiplication:

$$A_{[m \times n]} * A_{[n \times m]}^{-1} = A_{[n \times m]}^{-1} * A_{[m \times n]} = I$$

so, it must be that $m=n$!

2. A non-square matrix

inverse is possible using

SVD:

There exists a **left inverse** U and a **right inverse** V that is defined for **all** matrices including non-square matrices

```
1 # Using Singular Value Decomposition (SVD) for manually performing a pseudoinverse on a non-square matrix:
2 # It is not the actual inverse matrix, but the "best approximation" of such in the sense of least squares
3
4 from numpy import random, matrix, linalg, diag, allclose, dot
5
6 # Create Matrix A with size (3,5) containing random numbers:
7 A = random.random(15)
8 A = A.reshape(3,5)
9 A = matrix(A)
10
11 # 1-3. Using the SVD function will return:
12 # U - a matrix with columns = the eigenvectors (L) of the A*A.T
13 # holds Left-singular vectors
14 # s - a diagonal matrix with diagonal = the singular values of matrix A:
15 # the singular (diagonal) values in s are square roots of eigenvalues from U and V
16 # Σ+ is the pseudoinverse of Σ, which is formed by replacing every non-zero
17 # diagonal entry by its reciprocal and transposing the resulting matrix
18 # V - a matrix with columns = the eigenvectors (R) of the A.T*A
19 # holds Right-singular vectors
20 # U and V - must preserve the properties of the original matrix A, so they are orthogonal
21 U,s,V = linalg.svd(A, full_matrices=False)
22
23 # Construct a diagonal matrix 'S', from the diagonal 's':
24 S = diag(s)
25
26 # 2-3. Invert the square diagonal matrix by inverting each diagonal element:
27 S[0,0], S[1,1], S[2,2] = 1/diag(S[0:3,0:3])
28
29 # 3-3. Now we use the SVD elements to obtain the pseudo-inverse of matrix A:
30 X = dot(U, dot(S, V))
31 X = X.T # Final step: we must transpose
32
33 # Check each matrix:
34 A.shape, U.shape, S.shape, V.shape
35
36 # Comparison test 1:
37 A.I-X
38
39 # Comparison test 2:
40 allclose(A.I, X)
```

Random options in NumPy

- Random options in NumPy

```
In [55]: np.random.rand(3,2)
Out[55]:
array([[0.83252202, 0.38362844],
       [0.63309619, 0.29230593],
       [0.95616478, 0.07331045]])
```

Random values in a given shape.

Create an array of the given shape and populate it with random samples from a uniform distribution over `[0, 1)`.

```
In [56]: np.random.randn(3,2)
Out[56]:
array([[ -1.326648  , -0.43371855],
       [ 0.76498162,  0.21129565],
       [ 0.06951502,  0.24104293]])
```

`randn(d0, d1, ..., dn)`

Return a sample (or samples) from the "standard normal" distribution.

```
In [57]: np.random.random(3)
Out[57]: array([0.91596981, 0.60502728, 0.11632779])
```

Return random floats in the half-open interval `[0.0, 1.0)`.

```
In [58]: np.random.randint(30,45)
Out[58]: 38
```

`randint(low, high=None, size=None, dtype='l')`

Return random integers from `low` (inclusive) to `high` (exclusive).

Exercise

- Class exercise

Recall:

```
46 file = open('files/lecture3/test.txt', 'r') # opens file for reading
47 sentences = file.readlines()
48 print(sentences)
49 print(len(sentences))
50 file.close()
51
52 file = open('files/lecture3/test.txt', 'w') # opens file for writing
53 file.write('We will overwrite the previous text \n and go to a new line as well')
54 file.close()
55
56 file = open('files/lecture3/test.txt', 'r') # opens file for reading
57 sentences = file.readlines()
58 print(sentences)
59 print(len(sentences))
60 file.close()
```

Task:

- create a private class named **'file_operations'** with two methods/functions in it called:
'write_my_file' (will take an array) and **'read_my_file'** (will return a string)
- create an array with 1 row and 6 columns containing 6 numbers bound between 5 and 45
- store it in a text file (use **'ndarray.tofile'** method) *hint: use [help\(\)](#) on **'ndarray.tofile'** to see usage*
- open the file for reading and cast the string as a numpy array using `dtype=int16`
- create a matrix with 3 rows and 1 column
- multiply the array and the matrix
- write the result back to the same file
- open the file for reading and display your result on the screen using **print**

Result:

```
[[ 45 126  81 111  99  48]
 [ 30  84  54  74  66  32]
 [ 75 210 135 185 165  80]]
```

In [1]: |

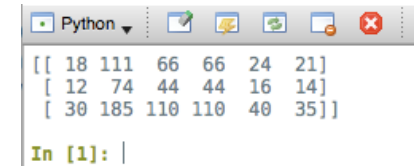
Lecture 4 exercise discussion

- Class exercise

Solution:

```
""" Execute in: Python 3.4.2 or higher """
1 from numpy import array as ar
2 from numpy import matrix as mx
3 from numpy import int16, random
4
5 class _file_operations():
6     def write_my_file(A):
7         file = open('my_array', 'w')
8         file.writelines('%s' %str(A))
9         file.close()
10
11     def read_my_file():
12         file = open('my_array', 'r')
13         B = file.read()
14         file.close()
15         return B
16
17 B = random.randint(5,45,6)
18 B.tofile('my_array', sep=',', format='%s')
19 B = _file_operations.read_my_file()
20 B = ar(B.split(','), dtype=int16)
21 C = mx([[3],[2],[5]])
22 D = C*B
23 _file_operations.write_my_file(D)
24 print(_file_operations.read_my_file())
```

Result:



```
Python
[[ 18 111 66 66 24 21]
 [ 12 74 44 44 16 14]
 [ 30 185 110 110 40 35]]
In [1]: |
```

Lecture 4 exercise discussion

- Class exercise

Solution:

```
In [1]: B = random.randint(5,45,6)

In [2]: B
Out[2]: array([21, 43, 16, 26, 43, 18])

In [3]: B.tofile('my_array', sep=',', format='%s')

In [4]: B = _file_operations.read_my_file()

In [5]: B
Out[5]: '21,43,16,26,43,18'

In [6]: type(B)
Out[6]: str

In [7]: B = ar(B.split(','), dtype=int16)

In [8]: B
Out[8]: array([21, 43, 16, 26, 43, 18], dtype=int16)

In [9]: C = mx([[3],[2],[5]])

In [10]: C
Out[10]:
matrix([[3],
        [2],
        [5]])

In [11]: D = C*B

In [12]: D
Out[12]:
matrix([[ 63, 129,  48,  78, 129,  54],
        [ 42,  86,  32,  52,  86,  36],
        [105, 215,  80, 130, 215,  90]])
```


HW assignment 2

1. Include a section with your name
2. Create matrix A with size (3,5) containing random numbers
3. Find the size and length of matrix A
4. Resize (crop/slice) matrix A to size (3,4)
5. Find the transpose of matrix A and assign it to B
6. Find the minimum value in column 1 of matrix B
7. Find the minimum and maximum values for the entire matrix A
8. Create vector X (an array) with 4 random numbers
9. Create a function and pass vector X and matrix A in it
10. In the new function multiply vector X with matrix A and assign the result to D
(note: you may get an error! ... think why and fix it. Recall matrix manipulation in class!)
11. Create a complex number Z with absolute and real parts != 0
12. Show its real and imaginary parts as well as its absolute value
13. Multiply result D with the absolute value of Z and record it to C
14. Convert matrix B from a matrix to a string and overwrite B
15. Display a text on the screen: 'Name is done with HW2', but pass your 'Name' as a string variable
16. Organize your code: use each line from this assignment as a comment line before each step
17. Save all steps as a script in a .py file
18. Submit your running code in the HW2 folder in Canvas by Sunday 11:59pm