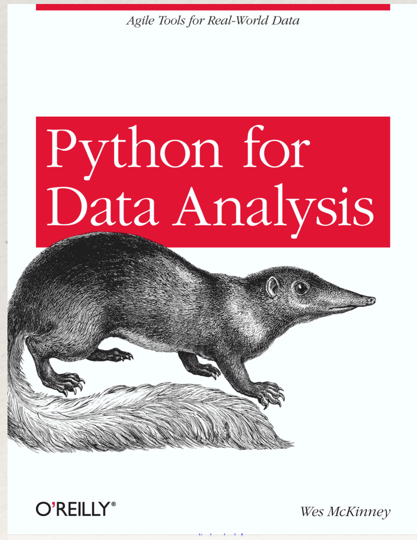
UC Berkeley
SCHOOL OF INFORMATION

Week 10

Presentations.

Programming for Data Science

Your object-oriented projects and a few notes about NumPy Vector



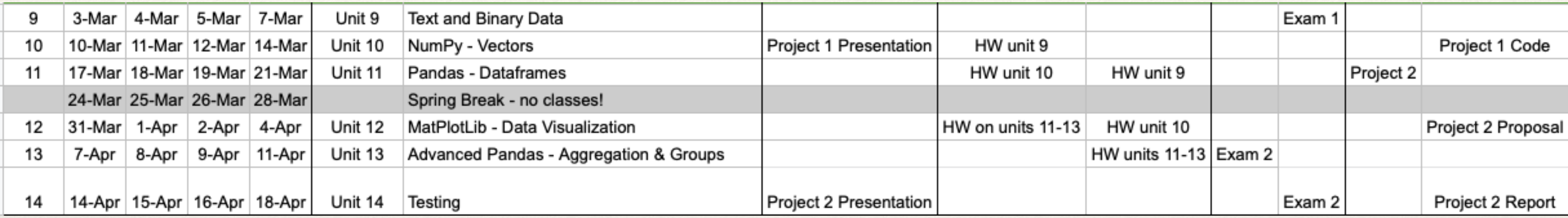
a copy of this text is in the resources folder.

Agenda

UC Berkeley
SCHOOL OF INFORMATION

- A few notes
- Links about Vectors
  - NumPy “cheat sheet” in the resources folder
  - A Quickstart tutorial (SciPy.org) https://docs.scipy.org/doc/numpy/user/quickstart.html
  - A review for a course in visual recognition (per student question) http://cs231n.github.io/python-numpy-tutorial/
  - For linear algebra (http://www2.lawrence.edu/fast/GREGGJ/Python/numpy/numpyLA.html)
  - For matrix arithmetic, etc. https://www.python-course.eu/matrix_arithmetic.php [useful for a lot of vector differences, such as text retrieval)
- Activities: there are two activities to practice (if time permits after presentations). Otherwise please check ‘em out and the solution at your leisure time (yeah, right! (grin)).

Schedule



9	3-Mar	4-Mar	5-Mar	7-Mar	Unit 9	Text and Binary Data					Exam 1		
10	10-Mar	11-Mar	12-Mar	14-Mar	Unit 10	NumPy - Vectors	Project 1 Presentation	HW unit 9					Project 1 Code
11	17-Mar	18-Mar	19-Mar	21-Mar	Unit 11	Pandas - Dataframes		HW unit 10	HW unit 9			Project 2	
	24-Mar	25-Mar	26-Mar	28-Mar		Spring Break - no classes!							
12	31-Mar	1-Apr	2-Apr	4-Apr	Unit 12	MatPlotLib - Data Visualization		HW on units 11-13	HW unit 10				Project 2 Proposal
13	7-Apr	8-Apr	9-Apr	11-Apr	Unit 13	Advanced Pandas - Aggregation & Groups			HW units 11-13	Exam 2			
14	14-Apr	15-Apr	16-Apr	18-Apr	Unit 14	Testing	Project 2 Presentation				Exam 2		Project 2 Report

Calendar

A student question about errors.

In any program, write errors to a log.

Let the end-user know of the most common errors (e.g., FnF)

Good to track date/time, IP, specific line (traceback), var, etc.

Of course, python has a library for that … (grin)

Remember:

Errors can be thrown as generic Exception, even the ones we write

Not a rule: I like to pass the errors to a single class/function to store for transaction log ana

Tools?

import sys, io, logging, traceback …

Logging and Traceback can be used for debugging and for controlling five levels of error warnings to the user and/or programmer.

```
import sys, logging, io, traceback

def logging_method():
   logging.debug("This is a debug message.")
   logging.info('This is an info message.')
   logging.warning('This is a warning message.')
   logging.error('This is an error message.')
   logging.critical('Run for the Hills! This is a critical message.')

   logging.basicConfig(filename='app.log', filemode='w', format='%(asctime)s -
       %(process)d - %(name)s - %(levelname)s - %(message)s', datefmt='%d-%b-%y %H:%M:%S', level=logging.ERROR)
```

Configure the logging object...

What data to be captured for log?

```
""" run python -0 abc.py versus python abc.py """

if __debug__:
   print("Debugging on.")
else:
   print("Debugging off.")
```

__debug__ ?!

```
def startMsg():
   print("-"*60)
   print("This is the generic error capture routine.  "\
         "All errors are sent here for processing. "\
         "Some specifics may be stored in a transaction log, "\
         "and some errors are offered gently to the end-user.")
   print("-"*60)
```

be nice to your end-users (grin)

```
def errorCapture(e):
   logging.basicConfig(filename='app.log', filemode='w', format='%(name)s - %(asctime)s - %(levelname)s -
      %(message)s', level=logging.INFO)
   logging.info('Admin logged in')
   logging.basicConfig(level=logging.DEBUG)
   logging.debug('This will get logged')
   print("Script threw this error:", e)
```

notice the config is called only once - the logging level determines which to be captured

```
  if isinstance(e, KeyboardInterrupt):
       print("\tSomeone pressed a keyboard interrupt.  Bye.")
       sys.exit()
       logger.exception("Normal stuff ... ")
   elif isinstance(e, ValueError):
       print("\tNope - a value error: ",e)
       logging.error("Demoing exception", exc_info = True)
   elif isinstance(e, FileNotFoundError):
       print("\tFile was not found, sorry.")
       logging_method()
       #log_traceback(e)
   else:
       logging.info("A message without exception")

def catchEverything():
   try:
       a = 'sequel'
       b = 0.8
       print(a + b)
   except Exception as e:
       errorCapture(e)


def tryingAFile():
   filename = 'guilhem.txt'
   try:
       with open(filename, 'r') as f:
           print(f)
   except IOError as e:
       errorCapture(e)
   finally:
       print("Thanks for playing.")

""" deliberate errors """
startMsg()
tryingAFile()
catchEverything()
```

Events! There are system and other kinds of “events” to “listen to.” Keyboard, mouse movements, system events - capture them or not. But vital for UX, InfoVis, any kind of interactivity with the end-user and GUI programming.

Always try/except any file i/o - files, databases, streams.

Here, too.

Everything set - do it.

A Unicode gotcha

- OxFF = decimal 255 … but when decoding it may end-up as 0000 0000 0000 0000 FFFF FFFF - meaning there's an extra 0000 at the beginning of stream and so an error : (

- https://docs.python.org/3/howto/unicode.html

Python For Data Science *Cheat Sheet*

NumPy Basics

Learn Python for Data Science **Interactively** at www.DataCamp.com

NumPy

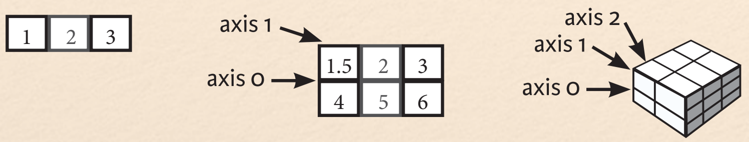
The **NumPy** library is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.

Use the following import convention:

```
>>> import numpy as np
```

NumPy Arrays

1D array | 2D array | 3D array



Creating Arrays

```
>>> a = np.array([1,2,3])
>>> b = np.array([(1.5,2,3), (4,5,6)], dtype = float)
>>> c = np.array([[(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]],
                 dtype = float)
```

Initial Placeholders

`>>> np.zeros((3,4))`	Create an array of zeros
`>>> np.ones((2,3,4),dtype=np.int16)`	Create an array of ones
`>>> d = np.arange(10,25,5)`	Create an array of evenly spaced values (step value)
`>>> np.linspace(0,2,9)`	Create an array of evenly spaced values (number of samples)
`>>> e = np.full((2,2),7)`	Create a constant array
`>>> f = np.eye(2)`	Create a 2X2 identity matrix
`>>> np.random.random((2,2))`	Create an array with random values
`>>> np.empty((3,2))`	Create an empty array

I/O

Saving & Loading On Disk

```
>>> np.save('my_array', a)
>>> np.savez('array.npz', a, b)
>>> np.load('my_array.npy')
```

Saving & Loading Text Files

```
>>> np.loadtxt("myfile.txt")
>>> np.genfromtxt("my_file.csv", delimiter=',')
>>> np.savetxt("myarray.txt", a, delimiter=" ")
```

Data Types

`>>> np.int64`	Signed 64-bit integer types
`>>> np.float32`	Standard double-precision floating point
`>>> np.complex`	Complex numbers represented by 128 floats
`>>> np.bool`	Boolean type storing `TRUE` and `FALSE` values
`>>> np.object`	Python object type
`>>> np.string_`	Fixed-length string type
`>>> np.unicode_`	Fixed-length unicode type

Inspecting Your Array

`>>> a.shape`	Array dimensions
`>>> len(a)`	Length of array
`>>> b.ndim`	Number of array dimensions
`>>> e.size`	Number of array elements
`>>> b.dtype`	Data type of array elements
`>>> b.dtype.name`	Name of data type
`>>> b.astype(int)`	Convert an array to a different type

Asking For Help

```
>>> np.info(np.ndarray.dtype)
```

Array Mathematics

Arithmetic Operations

`>>> g = a - b`
`array([[-0.5,  0. ,  0. ],`
`       [-3. , -3. , -3. ]])`
	Subtraction

`>>> np.subtract(a,b)`	Subtraction

`>>> b + a`
`array([[ 2.5,  4. ,  6. ],`
`       [ 5. ,  7. ,  9. ]])`
	Addition

`>>> np.add(b,a)`	Addition

`>>> a / b`
`array([[ 0.66666667, 1.        , 1.        ],`
`       [ 0.25      , 0.4       , 0.5       ]])`
	Division

`>>> np.divide(a,b)`	Division

`>>> a * b`
`array([[  1.5,   4. ,   9. ],`
`       [  4. ,  10. ,  18. ]])`
	Multiplication

`>>> np.multiply(a,b)`	Multiplication
`>>> np.exp(b)`	Exponentiation
`>>> np.sqrt(b)`	Square root
`>>> np.sin(a)`	Print sines of an array
`>>> np.cos(b)`	Element-wise cosine
`>>> np.log(a)`	Element-wise natural logarithm

`>>> e.dot(f)`
`array([[ 7.,  7.],`
`       [ 7.,  7.]])`
	Dot product

Comparison

`>>> a == b`
`array([[False,  True,  True],`
`       [False, False, False]], dtype=bool)`
	Element-wise comparison

`>>> a < 2`
`array([True, False, False], dtype=bool)`
	Element-wise comparison

`>>> np.array_equal(a, b)`	Array-wise comparison

Aggregate Functions

`>>> a.sum()`	Array-wise sum
`>>> a.min()`	Array-wise minimum value
`>>> b.max(axis=0)`	Maximum value of an array row
`>>> b.cumsum(axis=1)`	Cumulative sum of the elements
`>>> a.mean()`	Mean
`>>> b.median()`	Median
`>>> a.corrcoef()`	Correlation coefficient
`>>> np.std(b)`	Standard deviation

Copying Arrays

`>>> h = a.view()`	Create a view of the array with the same data
`>>> np.copy(a)`	Create a copy of the array
`>>> h = a.copy()`	Create a deep copy of the array

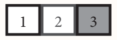
Sorting Arrays

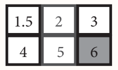
`>>> a.sort()`	Sort an array
`>>> c.sort(axis=0)`	Sort the elements of an array's axis

Subsetting, Slicing, Indexing

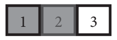
Also see Lists

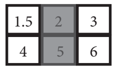
Subsetting

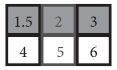
`>>> a[2]`
`3`
		Select the element at the 2nd index

`>>> b[1,2]`
`6.0`
		Select the element at row 1 column 2 (equivalent to `b[1][2]`)

Slicing

`>>> a[0:2]`
`array([1, 2])`
		Select items at index 0 and 1

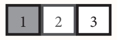
`>>> b[0:2,1]`
`array([ 2.,  5.])`
		Select items at rows 0 and 1 in column 1

`>>> b[:1]`
`array([[1.5, 2., 3.]])`
		Select all items at row 0 (equivalent to `b[0:1, :]`)

`>>> c[1,...]`
`array([[[ 3.,  2.,  1.],`
`        [ 4.,  5.,  6.]]])`
		Same as `[1,:,:]`

`>>> a[ : :-1]`
`array([3, 2, 1])`
		Reversed array `a`

Boolean Indexing

`>>> a[a<2]`
`array([1])`
		Select elements from `a` less than 2

Fancy Indexing

`>>> b[[1, 0, 1, 0],[0, 1, 2, 0]]`
`array([ 4. , 2. , 6. , 1.5])`
	Select elements (1,0),(0,1),(1,2) and (0,0)

`>>> b[[1, 0, 1, 0]][:,[0,1,2,0]]`
`array([[ 4. ,5. , 6. , 4. ],`
`       [ 1.5, 2. , 3. , 1.5],`
`       [ 4. , 5. , 6. , 4. ],`
`       [ 1.5, 2. , 3. , 1.5]])`
	Select a subset of the matrix's rows and columns

Array Manipulation

Transposing Array

`>>> i = np.transpose(b)`	Permute array dimensions
`>>> i.T`	Permute array dimensions

Changing Array Shape

`>>> b.ravel()`	Flatten the array
`>>> g.reshape(3,-2)`	Reshape, but don't change data

Adding/Removing Elements

`>>> h.resize((2,6))`	Return a new array with shape (2,6)
`>>> np.append(h,g)`	Append items to an array
`>>> np.insert(a, 1, 5)`	Insert items in an array
`>>> np.delete(a,[1])`	Delete items from an array

Combining Arrays

`>>> np.concatenate((a,d),axis=0)`
`array([ 1,  2,  3, 10, 15, 20])`
	Concatenate arrays

`>>> np.vstack((a,b))`
`array([[ 1. ,  2. ,  3. ],`
`       [ 1.5,  2. ,  3. ],`
`       [ 4. ,  5. ,  6. ]])`
	Stack arrays vertically (row-wise)

`>>> np.r_[e,f]`	Stack arrays vertically (row-wise)

`>>> np.hstack((e,f))`
`array([[ 7.,  7.,  1.,  0.],`
`       [ 7.,  7.,  0.,  1.]])`
	Stack arrays horizontally (column-wise)

`>>> np.column_stack((a,d))`
`array([[ 1, 10],`
`       [ 2, 15],`
`       [ 3, 20]])`
	Create stacked column-wise arrays

`>>> np.c_[a,d]`	Create stacked column-wise arrays

Splitting Arrays

`>>> np.hsplit(a,3)`
`[array([1]),array([2]),array([3])]`
	Split the array horizontally at the 3rd index

`>>> np.vsplit(c,2)`
`[array([[[ 1.5,  2. ,  1. ],`
`        [ 4. ,  5. ,  6. ]]]),`
` array([[[ 3.,  2.,  3.],`
`        [ 4.,  5.,  6.]]])]`
	Split the array vertically at the 2nd index

DataCamp

Learn Python for Data Science **Interactively**

Projects - take it away!