Download this page as a jupyter notebook at Lab 9

Laboratory 9: Matrices a Blue Pill Approach

Medrano, Giovanni

R11521018

ENGR 1330 Laboratory 9 - In-Lab

In [2]:
```
# Preamble script block to identify host, user, and kernel
import sys
! hostname
! whoami
print(sys.executable)
print(sys.version)
print(sys.version_info)
```

```
DESKTOP-6HAS1BN
desktop-6has1bn\medra
C:\Users\medra\anaconda3\python.exe
3.8.5 (default, Sep  3 2020, 21:29:08) [MSC v.1916 64 bit (AMD64)]
sys.version_info(major=3, minor=8, micro=5, releaselevel='final', serial=0)
```

NumPy

Numpy

Numpy is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays. The library's name is short for "Numeric Python" or "Numerical Python". If you are curious about NumPy, this cheat sheet is recommended:
https://s3.amazonaws.com/assets.datacamp.com/blog_assets/Numpy_Python_Cheat_Sheet.pdf

Arrays

A numpy array is a grid of values, all of the same type, and is indexed by a tuple of nonnegative integers. The number of dimensions is the rank of the array; the shape of an array is a tuple of integers giving the size of the array along each dimension. In other words, an array contains information about the raw data, how to locate an element and how to interpret an element.To make a numpy array, you can just use the np.array() function. All you need to do is pass a list to it. Don’t forget that, in order to work with the np.array() function, you need to make sure that the numpy library is present in your environment. If you want to read more about the differences between a Python list and NumPy array, this link is recommended: https://webcourses.ucf.edu/courses/1249560/pages/python-lists-vs-numpy-arrays-what-is-the-difference

---

Example- 1D Arrays

Let's create a 1D array from the 2000s (2000-2009):

In [17]:
```
import numpy as np              #First, we need to impoty "numpy"
mylist = [2000,2001,2002,2003,2004,2005,2006,2007,2008,2009]        #Create a list of
print(mylist)                  #Check how it looks
npArray = np.array(mylist)              #Define it as a numpy array
print(type(mylist))
print(npArray)
print(type(npArray))
```

```
[2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009]
<class 'list'>
[2000 2001 2002 2003 2004 2005 2006 2007 2008 2009]
<class 'numpy.ndarray'>
```

Example- n-Dimensional Arrays

Let's create a 5x2 array from the 2000s (2000-2009):

In [7]:
```
myotherlist = [[2000,2001],[2002,2003],[2004,2005],[2006,2007],[2008,2009]]     #Since
print(myotherlist)              #See how it looks as a list
np.array(myotherlist)           #See how it looks as a numpy array
```

```
[[2000, 2001], [2002, 2003], [2004, 2005], [2006, 2007], [2008, 2009]]
```

Out[7]:
```
array([[2000, 2001],
       [2002, 2003],
       [2004, 2005],
       [2006, 2007],
       [2008, 2009]])
```

Arrays Arithmetic

Once you have created the arrays, you can do basic Numpy operations. Numpy offers a variety of operations applicable on arrays. From basic operations such as summation, subtraction, multiplication and division to more advanced and essential operations such as matrix multiplication and other elementwise operations. In the examples below, we will go over some of these:

Example- 1D Array Arithmetic

- Define a 1D array with [0,12,24,36,48,60,72,84,96]
- Multiple all elements by 2
- Take all elements to the power of 2
- Find the maximum value of the array and its position
- Find the minimum value of the array and its position
- Define another 1D array with [-12,0,12,24,36,48,60,72,84]
- Find the summation and subtraction of these two arrays
- Find the multiplication of these two arrays

In [8]:

```
import numpy as np          #import numpy
Array1 = np.array([0,12,24,36,48,60,72,84,96])     #Step1: Define Array1
print(Array1)
print(Array1*2)     #Step2: Multiple all elements by 2
print(Array1**2)     #Step3: Take all elements to the power of 2
print(np.power(Array1,2)) #Another way to do the same thing, by using a function in num
print(np.max(Array1))     #Step4: Find the maximum value of the array
print(np.argmax(Array1))     ##Step4: Find the postition of the maximum value
print(np.min(Array1))     #Step5: Find the minimum value of the array
print(np.argmin(Array1))     ##Step5: Find the postition of the minimum value
Array2 = np.array([-12,0,12,24,36,48,60,72,84])     #Step6: Define Array2
print(Array2)
print(Array1+Array2)         #Step7: Find the summation of these two arrays
print(Array1-Array2)         #Step7: Find the subtraction of these two arrays
print(Array1*Array2)         #Step8: Find the multiplication of these two arrays
```

```
[ 0 12 24 36 48 60 72 84 96]
[  0  24  48  72  96 120 144 168 192]
[   0  144  576 1296 2304 3600 5184 7056 9216]
[   0  144  576 1296 2304 3600 5184 7056 9216]
96
8
0
0
[-12   0  12  24  36  48  60  72  84]
[-12  12  36  60  84 108 132 156 180]
[12 12 12 12 12 12 12 12 12]
[   0    0  288  864 1728 2880 4320 6048 8064]
```

You can see it is considerably easier than manipulating as a list - be aware that the multiply above is element-by-element and not the inner product of two matrices (as was shown in Lesson 08)

Example- n-Dimensional Array Arithmetic

- Define a 2x2 array with [5,10,15,20]
- Define another 2x2 array with [3,6,9,12]
- Find the summation and subtraction of these two arrays
- Find the minimum number in the multiplication of these two arrays
- Find the position of the maximum in the multiplication of these two arrays
- Find the mean of the multiplication of these two arrays
- Find the mean of the first row of the multiplication of these two arrays

In [10]:

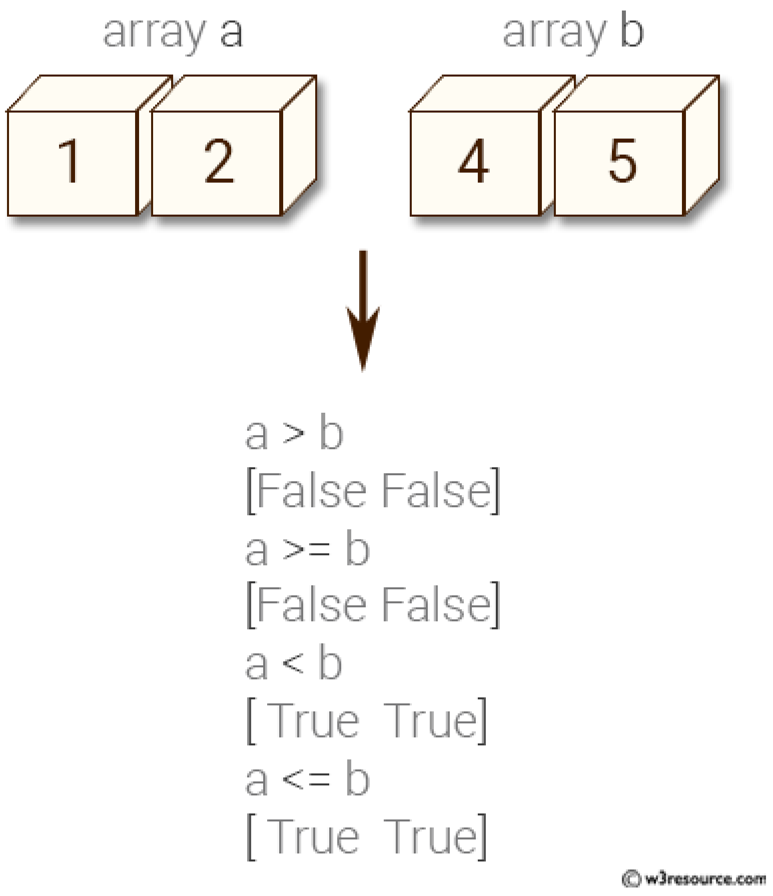
```
import numpy as np          #import numpy
Array1 = np.array([[5,10],[15,20]])     #Step1: Define Array1
```

```
print(Array1)
Array2 = np.array([[3,6],[9,12]])     #Step2: Define Array2
print(Array2)
print(Array1+Array2)     #Step3: Find the summation
print(Array1-Array2)     #Step3: Find the subtraction
MultArray = Array1@Array2         #Step4: To perform a typical matrix multiplication (o
MultArray1 = Array1.dot(Array2)         #Step4: Another way To perform a  matrix multip
print(MultArray)
print(MultArray1)
print(np.min(MultArray))     #Step4: Find the minimum value of the multiplication
print(np.argmax(MultArray))     ##Step5: Find the postition of the maximum value
print(np.mean(MultArray))     ##Step6: Find the mean of the multiplication of these two
print(np.mean(MultArray[0,:]))     ##Step7: Find the mean of the first row of the multi
```

```
[[ 5 10]
 [15 20]]
[[ 3  6]
 [ 9 12]]
[[ 8 16]
 [24 32]]
[[2 4]
 [6 8]]
[[105 150]
 [225 330]]
[[105 150]
 [225 330]]
105
3
202.5
127.5
```

Arrays Comparison

Comparing two NumPy arrays determines whether they are equivalent by checking if every element at each corresponding index are the same.



Example- 1D Array Comparison

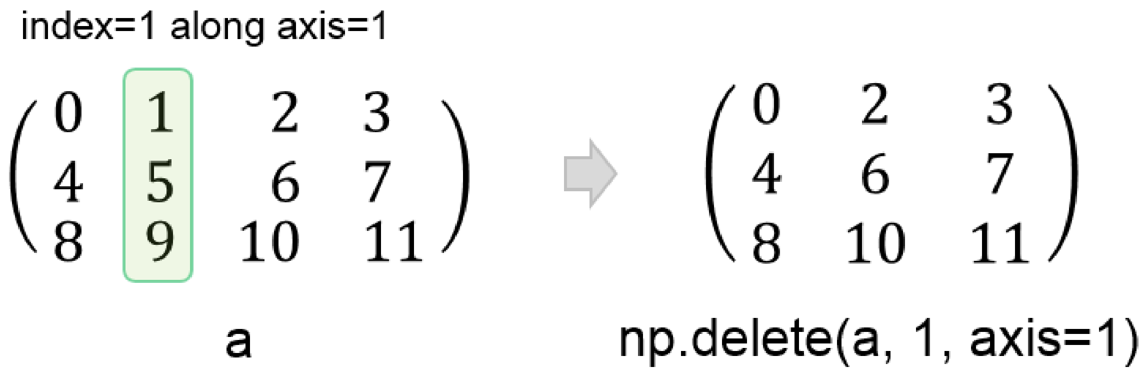
- Define a 1D array with [1.0,2.5,3.4,7,7]
- Define another 1D array with [5.0/5.0,5.0/2,6.8/2,21/3,14/2]
- Compare and see if the two arrays are equal
- Define another 1D array with [6,1.4,2.2,7.5,7]
- Compare and see if the first array is greater than or equal to the third array

In [11]:
```
import numpy as np          #import numpy
Array1 = np.array([1.0,2.5,3.4,7,7])     #Step1: Define Array1
print(Array1)
Array2 = np.array([5.0/5.0,5.0/2,6.8/2,21/3,14/2])     #Step2: Define Array1
print(Array2)
print(np.equal(Array1, Array2))            #Step3: Compare and see if the two arrays a
Array3 = np.array([6,1.4,2.2,7.5,7])     #Step4: Define Array3
print(Array3)
print(np.greater_equal(Array1, Array3))           #Step3: Compare and see if the two
```

```
[1.  2.5 3.4 7.  7. ]
[1.  2.5 3.4 7.  7. ]
[ True  True  True  True  True]
[6.  1.4 2.2 7.5 7. ]
[False  True  True False  True]
```

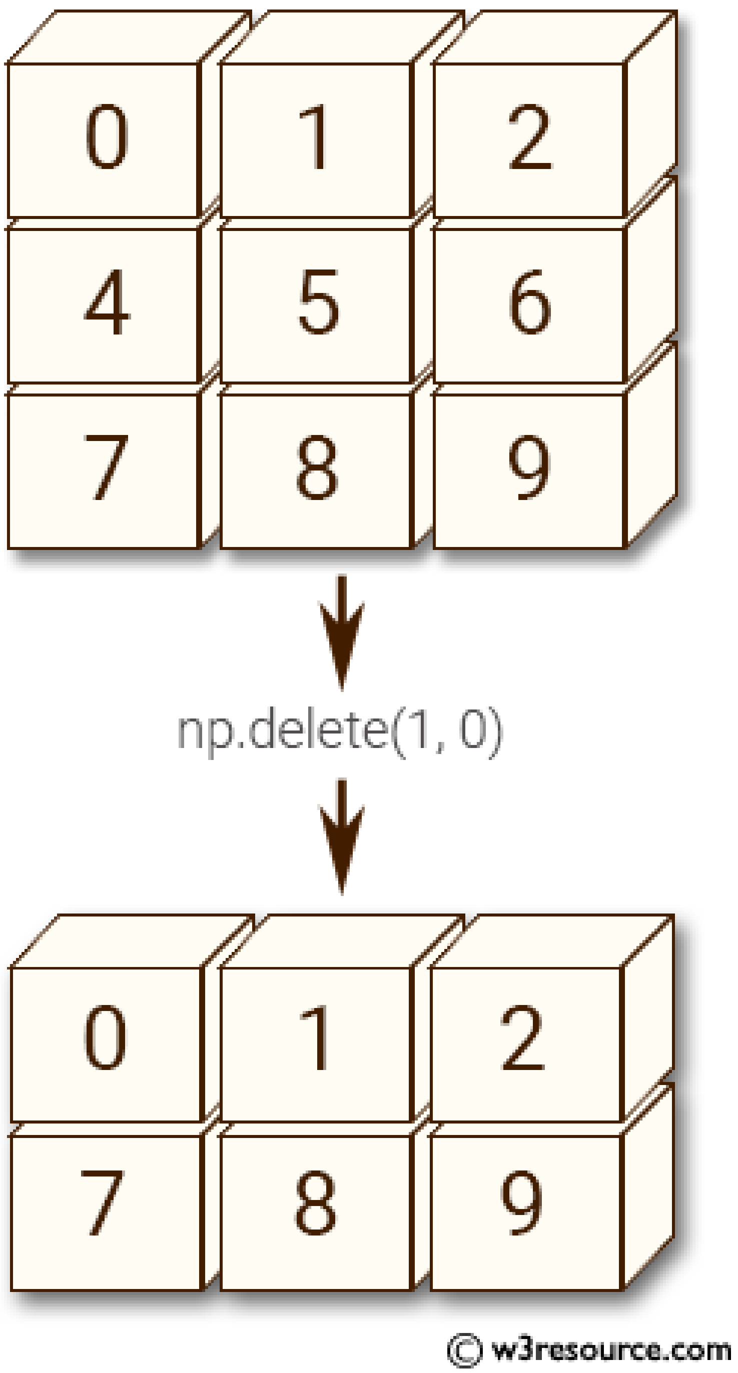
Arrays Manipulation

numpy.copy() allows us to create a copy of an array. This is particularly useful when we need to manipulate an array while keeping an original copy in memory. The numpy.delete() function returns a new array with sub-arrays along an axis deleted. Here is a poctoral representation:



$$\begin{pmatrix} 0 & \boxed{1} & 2 & 3 \\ 4 & \boxed{5} & 6 & 7 \\ 8 & \boxed{9} & 10 & 11 \end{pmatrix} \Rightarrow \begin{pmatrix} 0 & 2 & 3 \\ 4 & 6 & 7 \\ 8 & 10 & 11 \end{pmatrix}$$

Here is another pictorial representation:



Let's have a look at the examples.

Example- Copying and Deleting Arrays and Elements

- Define a 1D array, named "x" with [1,2,3]
- Define "y" so that "y=x"
- Define "z" as a copy of "x"
- Discuss the difference between y and z
- Delete the second element of x

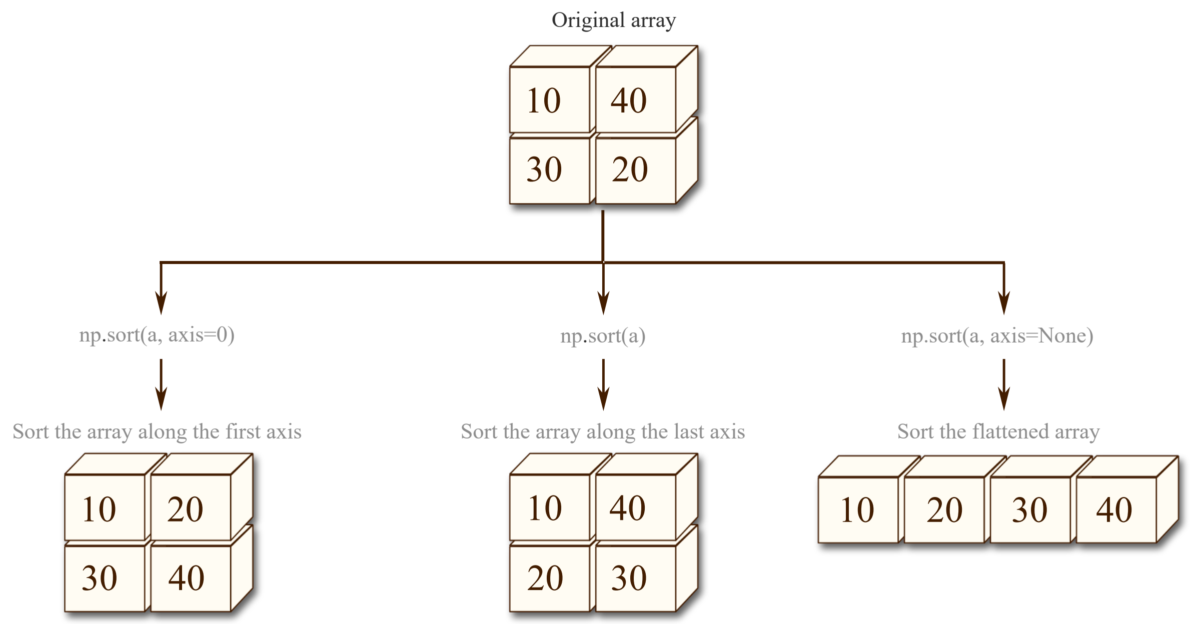
```
In [19]: import numpy as np          #import numpy
```

```
x = np.array([1,2,3])       #Step1: Define x
print("X is", x)
y = x                       #Step2: Define y as y=x
print("Y is", y)
z = np.copy(x)              #Step3: Define z as a copy of x
print("Z is", z)
# For Step4: They look similar but check this out:
x[1] = 8                    # If we change x ...
print("x is", x)
print("y is", y)
print("y is", z)
# By modifying x, y changes but z remains as a copy of the initial version of x.
x = np.delete(x, 1)        #Step5: Delete the second element of x
print(x)
```

```
X is [1 2 3]
Y is [1 2 3]
Z is [1 2 3]
x is [1 8 3]
y is [1 8 3]
y is [1 2 3]
[1 3]
```

Sorting Arrays

Sorting means putting elements in an ordered sequence. Ordered sequence is any sequence that has an order corresponding to elements, like numeric or alphabetical, ascending or descending. If you use the sort() method on a 2-D array, both arrays will be sorted.



Example- Sorting 1D Arrays

Define a 1D array as ['FIFA 2020','Red Dead Redemption','Fallout','GTA','NBA 2018','Need For Speed'] and print it out. Then, sort the array alphabetically.

In [20]:
```
import numpy as np         #import numpy
games = np.array(['FIFA 2020','Red Dead Redemption','Fallout','GTA','NBA 2018','Need Fo
print(games)
print(np.sort(games))
```

```
['FIFA 2020' 'Red Dead Redemption' 'Fallout' 'GTA' 'NBA 2018'
 'Need For Speed']
['FIFA 2020' 'Fallout' 'GTA' 'NBA 2018' 'Need For Speed'
 'Red Dead Redemption']
```

Example- Sorting n-Dimensional Arrays

Define a 3x3 array with 17,-6,2,86,-12,0,0,23,12 and print it out. Then, sort the array.

In [14]:
```
import numpy as np         #import numpy
a = np.array([[17,-6,2],[86,-12,0],[0,23,12]])
print(a)
print ("Along columns : \n", np.sort(a,axis = 0) ) #This will be sorting in each column
print ("Along rows : \n", np.sort(a,axis = 1) ) #This will be sorting in each row
print ("Sorting by default : \n", np.sort(a) )        #Same as above
print ("Along None Axis : \n", np.sort(a,axis = None) ) #This will be sorted like a 1D
```

```
[[ 17  -6   2]
 [ 86 -12   0]
 [  0  23  12]]
Along columns :
 [[  0 -12   0]
 [ 17  -6   2]
 [ 86  23  12]]
Along rows :
 [[ -6   2  17]
 [-12   0  86]
 [  0  12  23]]
Sorting by default :
 [[ -6   2  17]
 [-12   0  86]
 [  0  12  23]]
Along None Axis :
 [-12  -6   0   0   2  12  17  23  86]
```

Partitioning (Slice) Arrays

Slicing in python means taking elements from one given index to another given index.

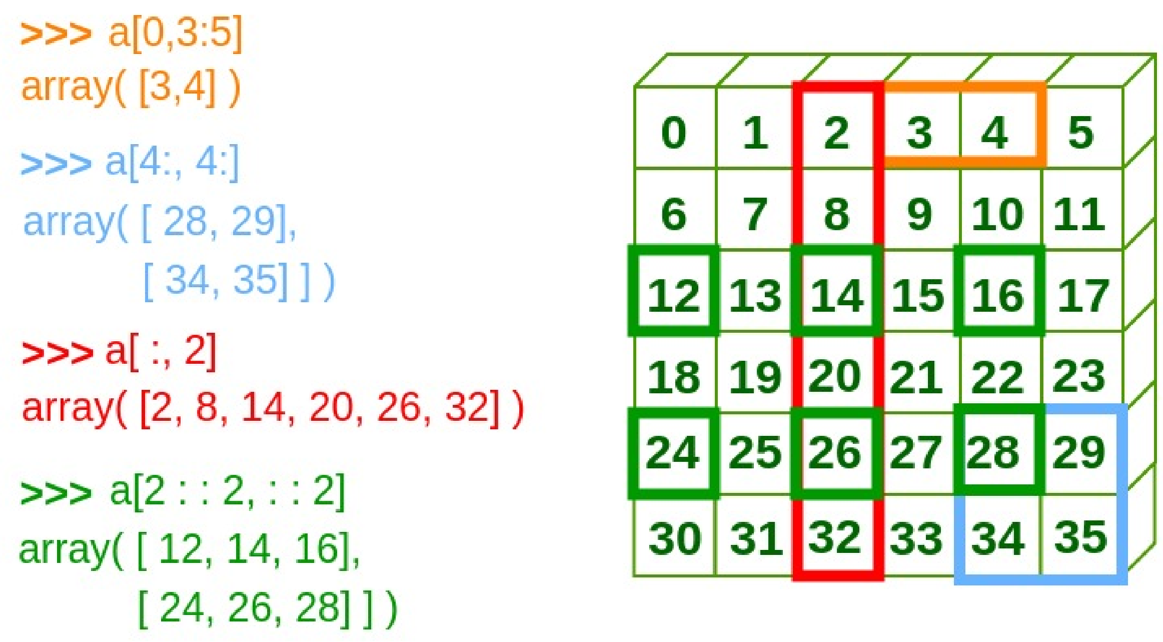
We can do slicing like this: [start:end].

We can also define the step, like this: [start:end:step].

If we don't pass start its considered 0

If we don't pass end its considered length of array in that dimension

If we don't pass step its considered 1



Example- Slicing 1D Arrays

Define a 1D array as [1,3,5,7,9], slice out the [3,5,7] and print it out.

In [21]:
```
import numpy as np          #import numpy
a = np.array([1,3,5,7,9])     #Define the array
print(a)
aslice = a[1:4]              #slice the [3,5,7]
print(aslice)                #print it out
```

```
[1 3 5 7 9]
[3 5 7]
```

Example- Slicing n-Dimensional Arrays

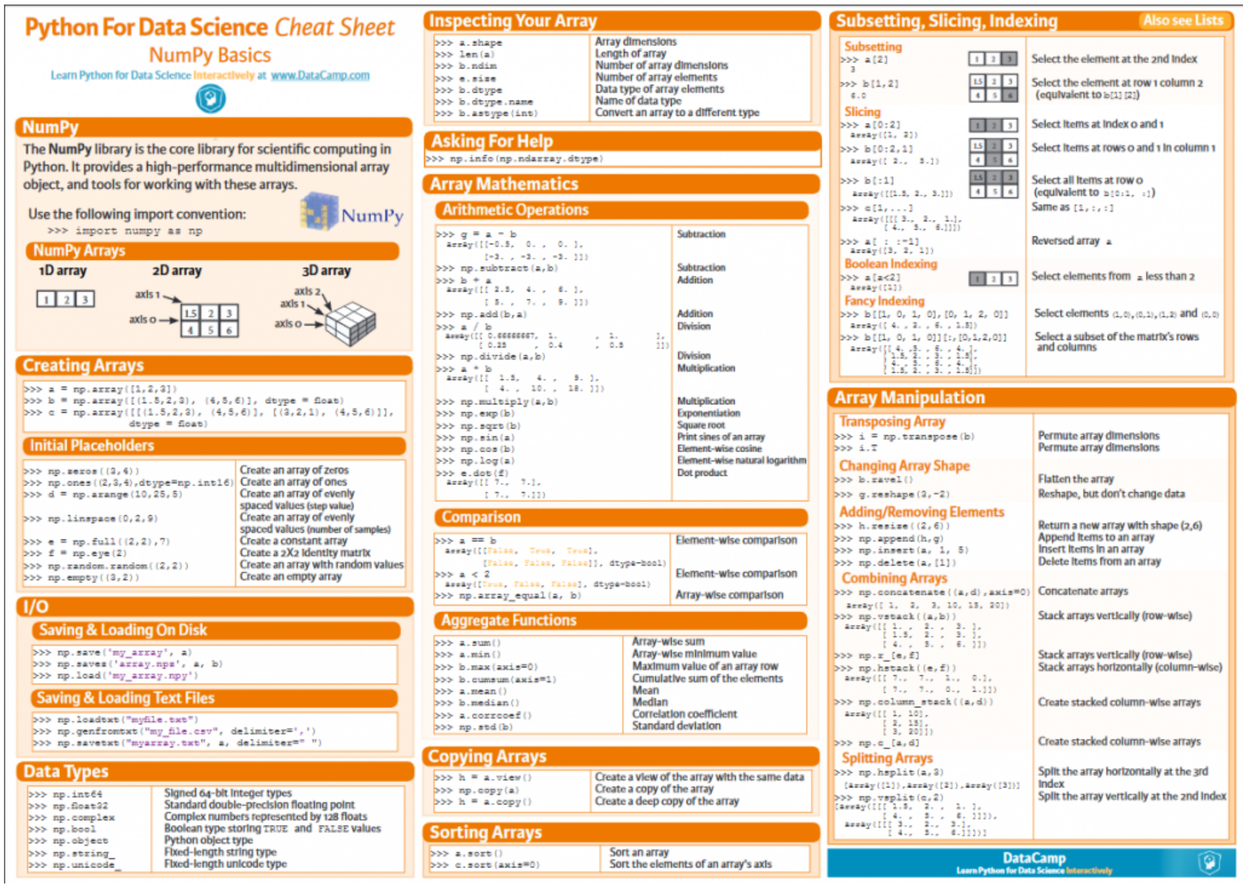
Define a 5x5 array with "Superman, Batman, Jim Hammond, Captain America, Green Arrow, Aquaman, Wonder Woman, Martian Manhunter, Barry Allen, Hal Jordan, Hawkman, Ray Palmer, Spider Man, Thor, Hank Pym, Solar, Iron Man, Dr. Strange, Daredevil, Ted Kord, Captian Marvel, Black Panther, Wolverine, Booster Gold, Spawn " and print it out. Then:

- Slice the first column and print it out
- Slice the third row and print it out
- Slice 'Wolverine' and print it out
- Slice a 3x3 array with 'Wonder Woman, Ray Palmer, Iron Man, Martian Manhunter, Spider Man, Dr. Strange, Barry Allen, Thor, Daredevil'

In [22]:
```
import numpy as np          #import numpy
Superheroes = np.array([['Superman', 'Batman', 'Jim Hammond', 'Captain America', 'Green
                        ['Aquaman', 'Wonder Woman', 'Martian Manhunter', 'Barry Allen', 'Hal Jor
                        ['Hawkman', 'Ray Palmer', 'Spider Man', 'Thor', 'Hank Pym'],
                        ['Solar', 'Iron Man', 'Dr. Strange', 'Daredevil', 'Ted Kord'],
                      ['Captian Marvel', 'Black Panther', 'Wolverine', 'Booster Gold', 'Spawn'
print(Superheroes) #Step1
print(Superheroes[:,0])
print(Superheroes[2,:])
print(Superheroes[4,2])
print(Superheroes[1:4,1:4])
```

```
[['Superman' 'Batman' 'Jim Hammond' 'Captain America' 'Green Arrow']
 ['Aquaman' 'Wonder Woman' 'Martian Manhunter' 'Barry Allen' 'Hal Jordan']
 ['Hawkman' 'Ray Palmer' 'Spider Man' 'Thor' 'Hank Pym']
 ['Solar' 'Iron Man' 'Dr. Strange' 'Daredevil' 'Ted Kord']
 ['Captian Marvel' 'Black Panther' 'Wolverine' 'Booster Gold' 'Spawn']]
['Superman' 'Aquaman' 'Hawkman' 'Solar' 'Captian Marvel']
['Hawkman' 'Ray Palmer' 'Spider Man' 'Thor' 'Hank Pym']
Wolverine
[['Wonder Woman' 'Martian Manhunter' 'Barry Allen']
 ['Ray Palmer' 'Spider Man' 'Thor']
 ['Iron Man' 'Dr. Strange' 'Daredevil']]
```

This is a Numpy Cheat Sheet- similar to the one you had on top of this notebook!



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

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

Check out this link for more:

https://blog.finxter.com/collection-10-best-numpy-cheat-sheets-every-python-coder-must-own/

Exercise: Python List vs. Numpy Arrays?

- What are some differences between Python lists and Numpy arrays?

Put your answer in the empty markdown cell below; Make sure to cite any resources that you may use.

*Use this markdown cell for your answer

array can be applied math directly whilst a list cannot. array is faster and uses less memory list can have different data types.

Here are some of the resources used for creating this notebook:

- Johnson, J. (2020). Python Numpy Tutorial (with Jupyter and Colab). Retrieved September 15, 2020, from https://cs231n.github.io/python-numpy-tutorial/
- Willems, K. (2019). (Tutorial) Python NUMPY Array TUTORIAL. Retrieved September 15, 2020, from https://www.datacamp.com/community/tutorials/python-numpy-tutorial?utm_source=adwords_ppc
- Willems, K. (2017). NumPy Cheat Sheet: Data Analysis in Python. Retrieved September 15, 2020, from https://www.datacamp.com/community/blog/python-numpy-cheat-sheet
- W3resource. (2020). NumPy: Compare two given arrays. Retrieved September 15, 2020, from https://www.w3resource.com/python-exercises/numpy/python-numpy-exercise-28.php

Here are some great reads on this topic:

- **"Python NumPy Tutorial"** available at *https://www.geeksforgeeks.org/python-numpy-tutorial/
- **"What Is NumPy?"** a collection of blogs, available at *https://realpython.com/tutorials/numpy/
- **"Look Ma, No For-Loops: Array Programming With NumPy"** by **Brad Solomon** available at *https://realpython.com/numpy-array-programming/
- **"The Ultimate Beginner's Guide to NumPy"** by **Anne Bonner** available at *https://towardsdatascience.com/the-ultimate-beginners-guide-to-numpy-f5a2f99aef54

Here are some great videos on these topics:

- **"Learn NUMPY in 5 minutes - BEST Python Library!"** by **Python Programmer** available at *https://www.youtube.com/watch?v=xECXZ3tyONo
- **"Python NumPy Tutorial for Beginners"** by **freeCodeCamp.org** available at *https://www.youtube.com/watch?v=QUT1VHiLmmI
- **"Complete Python NumPy Tutorial (Creating Arrays, Indexing, Math, Statistics, Reshaping)"** by **Keith Galli** available at *https://www.youtube.com/watch?v=GB9ByFAIAH4
- **"Python NumPy Tutorial | NumPy Array | Python Tutorial For Beginners | Python Training | Edureka"** by **edureka!** available at *https://www.youtube.com/watch?v=8JfDAm9y_7s

In [ ]: