## Section 1: Course Introduction ☑

Data Science -> #1 Job in Glassdoor

Solve very interesting problems

This course will cover

-> Most popular Python Data Science Libraries

- NumPy
- SciPy
- Pandas
- Seaborn
- Scikit-Learn
- Matplotlib
- Plotly
- PySpark

Course contents

PART I

1) Python Crash Course -> Refreshing Python
2) NumPy -> A scientific computing library for Python -> Allows us to work with data arrays in Python.
3) Pandas -> An excellent data analysis library -> Allows us to read data from multiple sources.

↳ • csv files, • SQL databases.
• Excel
• HTML webpages

4) Matplotlib -> Basic data visualization.

5) Seaborn -> Library used to create beautiful statistical plots.
-> Interactive plotting techniques
↳ • Financial data
• Interactive data for users.
• Geographical plotting

PART II

6) Sci-kit Learn -> Implementing ML algorithms.
(SL) { (LR, LoR, Decision trees, Random Forests)
(UL) { clustering algorithms

7) Big-Data -> AWS

8) TF Basics

↳ * Section Exercises
* Capstone Data Projects
* ML Portfolio Projects

Section II -> Environment setup ☑

Section III -> Jupyter overview ☑

## Virtual Enviroments (7)

↳ using Anaconda

what are virtual environments?

↳ • Allow you to set a virtual installations of Python and libraries in your computer.

• This way -> You can have multiple version of python libraries

Easily activate & De-activate them.

Example 1:

• Sometimes you'll want to program in different Versions of a library

↳ You develop a program in Sci-kit Learn 0.17

Version 0.18 released!

⇓

want to explore 0.18 BUT don't want 0.17 code to break

Example 2:

- Sometimes you'll want to make sure your library installations are in the correct location.
  - You want multiple versions of Python on your computer.
  - You want one environment with Py 2.7 and another with Py 3.5.

- How to setup virtual environments?
  - There is a virtualenv library for normal Python distributions.
  - Anaconda -> has a virtual environment manager that makes the whole process really easy.

To create a new conda environment:

```
$ conda create --name envName packageName
```

- We can also specify the python version we want to install in our virtual environment -> python=2.7
- An specific library version -> scipy=0.15.0

- To remove an environment:

```
$ conda env remove --name envName
```

## Section IV: Python Crash Course

### Introduction:

- General overview of Python Programming.
- Crash Course -> Prior programming experience -> Required.

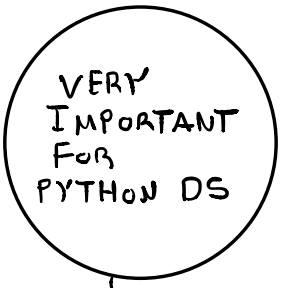
Check List -> General overview

- Data Types
  - Numbers ✓
  - Strings ✓
  - Print Formatting ✓
  - Lists ✓
  - Dictionaries ✓
  - Booleans ✓
  - Tuples & sets ✓
- ✓ Comparison operators
- ✓ If, elif and else statements ✓
- For Loops ✓
- While Loops ✓
- range() ✓
- List Comprehension ✓
- Functions ✓
- Lambda expressions ✓
- Map and Filter ✓

## Section 5: Python for Data Analysis - Numpy

### Introduction to Numpy:

Numpy -> Linear Algebra Library for Python




- Almost all of the libraries in the PyData ecosystem rely on Numpy as one of its main building blocks

- Numpy is also incredibly fast -> It has bindings to C libraries

- Numpy -> Installed either with conda or pip

### Numpy arrays

- Are the main way we are going to use Numpy throughout the course.

- Numpy arrays
  - Vectors (1-d arrays)
  - Matrices (2-d arrays)

Note: A matrix can still have either one row or one column.

# Numpy Arrays

- The main way we are going to use the Numpy library.
- Arrays can either be called matrices or vectors if they are 2d or 1d.

- We import the numpy library as follows:

```
import numpy as np
```

By convention we use np to call the numpy library functions and attributes ♡‿♡ (By my wife CLAU)

A) Casting Python objects into numpy arrays

↳ we can create numpy arrays using Python objects such as lists, e.g.

```
myList = [1,2,3]

myArray = np.array(myList)
```

Returns an array as the container of that list object.

- To obtain a 2d array we can cast a list of lists as follows:

```
my_mat = [ [1,2,3], [4,5,6] ]
```

Then:

```
my_mat_arr = np.array(my_mat)
```

## B) Numpy built-in array generation

Methods:

① Using `arange()`

Similar to Python's built in range() function.

`np.arange(0, 11, 2)`

- 0 → start (Inclusive)
- 11 → stop (non-inclusive)
- 2 → step

② `np.zeros()` -> Generates an array of zeros

-> we pass an integer to specify vector dimension.

-> we pass a tuple to specify the desired dimensions (2d) -> Need to pass the tuple with (a,b) notation.

③ `np.ones()`

↳ Generates an array of ones (we specify the dimensions similarly to `np.zeros()`)

④ np.linspace() → Returns an array of evenly spaced numbers over a specified interval

np.linspace(start, stop, intervals)

Inclusive

⑤ np.eye() → Creates an identity matrix
→ Pass an integer to specify an nxn identity matrix

⑥ RNG

(1), (5,5) No need for tuple notation

Ⓐ np.random.rand(size)
↳ Creates a uniformly distributed array of random numbers.

Ⓑ np.random.randn(size)
↳ Creates an array whose elements are sampled from the standard distribution.

Ⓒ np.random.randint(start, stop, size)
Inclusive Exclusive
↳ Returns a uniformly distributed array of random integers.

- Useful np array attributes and methods

(1) arr. reshape( ) -> Returns an array with a different shape.

The new dimensions of the array must have the same number of elements as the original array.

$r_{old} \times c_{old} = r_{new} \times c_{new}$

original size

New size

(2) arr. max( ) & arr. min( ) -> Return the min and max values of the array of the array.

(3) arr. argmax( ) & arr. argmin( ) -> Return the index of max and min values. (argmax & argmin values)

(4) arr. shape -> Returns the shape of your array.

⑤ arr.dtype -> Returns the data type contained in your array.

* using imports to simplify code:

```
from numpy.random import randInt as rdi
```

rdi (1, 10, 2) -> Returns the same as:

np.randInt (1, 10, 2)

- NumPy Array Indexing

- Bracket indexing and selection

↳ • To extract an element from an np.array we use the same syntax we would use in a python list.

↳ e.g. arr[8] -> Extracts element 8 in our array.

• we can also extract several values with the slice notation

arr[1:5] -> Extracts values from index 1 to index 4 in our array.

(1: Inclusive, 5: non-inclusive)

• We can also omit the start & end indices and they will be set to their default values.

↳ start = 0
end = len(arr) - 1

- arr[:] = arr[0 : len(arr) - 1]

Returns the whole array.

- Broadcasting
  ↳ • one of the main differences between python lists and np arrays.

Broadcasting resizes our array accordingly.

↳ e.g. arr[0:5] = 100

Broadcasts this into a 1x5 array and performs the assignment.

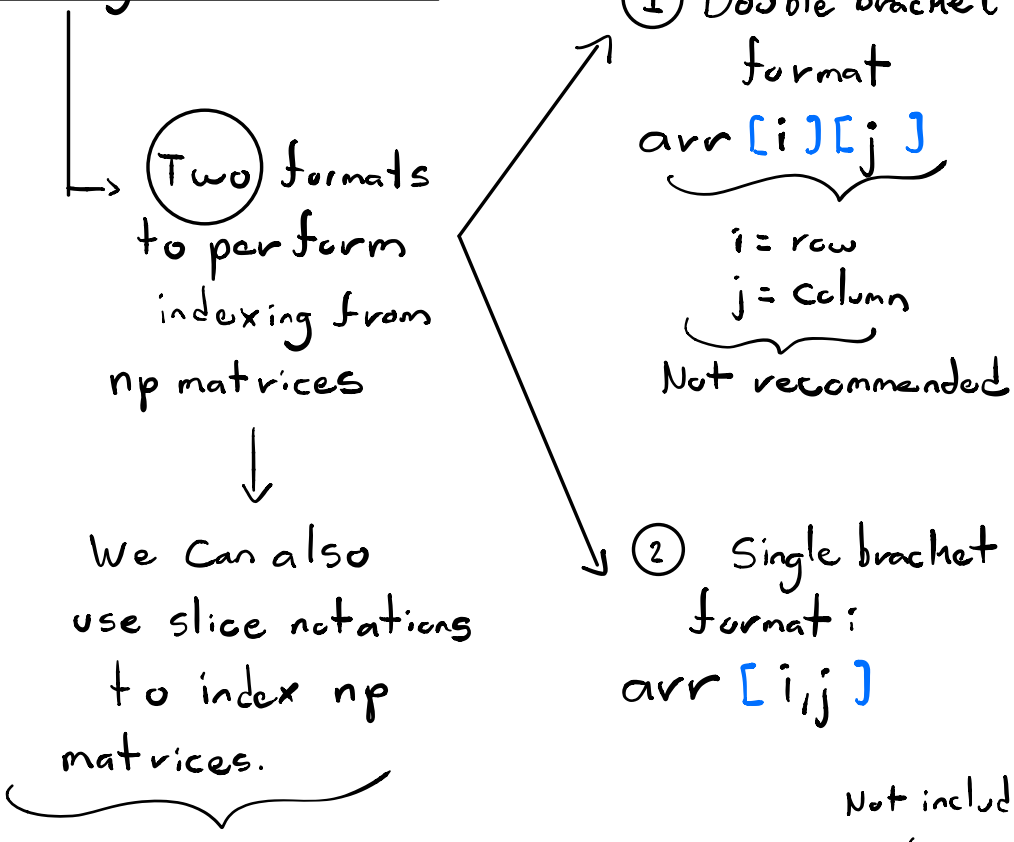
- Be Careful -> Python uses references to point to np arrays in memory. Thus, changes from any variable pointing to the array will be reflected in every variable pointing to the array.
  ↳ Done for memory reasons when

using large np arrays.

- To avoid the above behavior we must use the `.copy()` method as follows:

```
arr_copy = arr.copy()
```

- Indexing np matrices




Not included

`arr_2d[:2, 1:]` -> Selects rows 0:2 and Columns 1: len(arr_2d)

- Conditional Selection

  ↳ we can use boolean/conditional operations to perform selections from arrays as follows:

`arr[ arr > 5 ]` -> outputs on array whose elements are greater than 5.

`arr > 5` : Conditional expression

e.g. `arr = [1, 2, 3, ... 10]`

`arr[arr > 5] = [6, 7, ... 10]`

This is very common, specially when we use the pandas library!

## Numpy Operations

- np operations -> Basic operations we can perform with numpy arrays

  ↳ 1) Array with Array operations

  ↳ We can perform array with array operations using the arithmetic operators found in Python.

  ↳ Element-wise basis

## 2) Array with Scalar operations

↳ Numpy perform broadcasting such that the scalar has an agreeable shape. ⇝ operations are also performed in Element-wise basis

↳ * undefined divisions return the following + warning (Python outputs an Error/Exception)

↳ 0/0 → nan

1/0 → inf

## 3) Universal Array Functions

↳ Are np functions which perform mathematical operations which may be broadcasted across the entire array.

↓ e.g. Element-wise basis

- np.sqrt(arr) → Returns the sqrt of arr
- np.exp(arr) → Returns arr exp
- np.max(arr) → Returns an int → max of arr.
- np.sin(arr) → Returns the sine of arr

UAF Contained in ufunc documentation. optimized!

↳ Implement them when possible!