Notes for NumPy

General Theory

- Python is min-inclusive, max-exclusive
- **Axis notation**
    - Python sets axis=0 to the outermost grouping. When we add a new dimension, the added dimension is the outermost group.
        - We start with columns(not actually how it is) →add rows: therefore going down row by row is axis=0 and axis=1 is columns
        - If we add another grouping(3D→depth) →going deeper in depth is axis=0 and axis=1 is now rows(which was axis=0 before we added a new grouping level) etc.
        - For example: np.ndarray((depth_size,row_size,column_size))
- When putting '**arg**' in front of commands, it returns the index at which the command holds true

Ufuncs→allow Python to compute individual operations on large arrays of data efficiently

- For all ufuncs, the output array can be specified using 'out='
- The 'reduce' method of any ufunc will continuously operate until only one value remains(good for summation of all elements, etc.)

and/or	**&/|**
boolean evaluation of an entire object	Boolean evaluation on the individual elements *generally more desired for NumPy arrays

---

On NumPy arrays

Creation Commands

- np.zeros → array full of zeros
- np.empty
- np.full→fills array with the specified value
- np.randint - integers in a given interval
- np.random - random between 0-1
- np.linspace→evenly spaced intervals(number of points)
- np.arange→generate values evenly spaced across an interval(step size)
- np.array()-->create NumPy array
    - 'dtype=' specifies element types

Creating multi-datatype arrays

- multi_dtype_array=np.[c_cmd](#,dtype={'names':(names),'formats':(formats)})*-->creates an array with # elements with each element possessing the variables 'names'(with their respective formats–dtype)*
- multi_dtype_array['specific name']=[array to insert]
- *much better alternative: Pandas Dataframes...

Accessing

- A[index #]-->retrieves element at index #
    - Use negative values to start counting from the back
        - max-exclusive so +1 # than if u were from the front(-1 is the lastmost element, etc.)
    - For multi-dimensional, separate dimensions through ','
        - [:]-->all of specified index(ex: a[:,0]-->first element of every row)
- **Fancy Indexing**
    - Passing through an array of indices instead of a single scalar
- **Slicing**
    - array[start:stop:step]
        - If position, default: start-0, stop-end, step-1
        - If step<0→goes backward (note: start default is the end and the stop default is 0 of original)

- **Concatenate**
    - np.concatenate([arrays],axis=)
        - Axis→0-indexed; specifies which dimension to join arrays in
    - np.vstack(), np.hstack(), np.dstack()
- **Split**
    - np.split(array,[split position])
        - Splits after the position specified
    - np.vsplit(), np.hsplit()-->same principle as np.split

When accessing, we are given views, not changing the actual array→must make <u>copies</u> and specify changes

- array.copy()

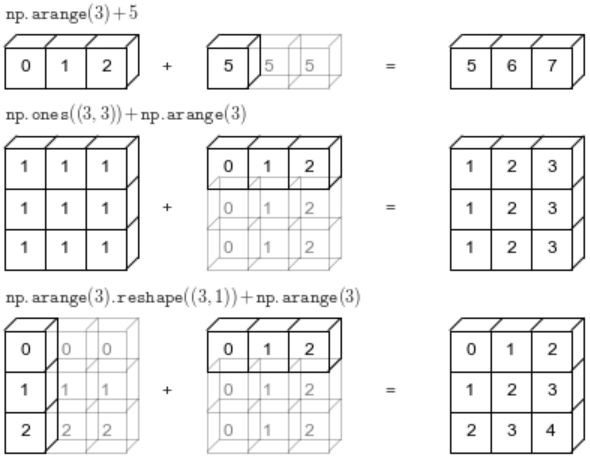
Representing Data + Aggregation

- np.sum(array)-->sums all elements
- np.min(array)/np.max(array)-->finding min/max of the given array
- There are many unlisted functions here
    - <u>Some more useful ones:</u>
        - np.mean(), np.std(), np.variance(), np.argmin/max()-->*returns index of min/max element*, np.median(), np.percentile()

Broadcasting

<u>Rules:</u>

- Rule 1: If the two arrays differ in their number of dimensions, the shape of the one with fewer dimensions is *padded* with ones on its leading (left) side.
- Rule 2: If the shape of the two arrays does not match in any dimension, the array with shape equal to 1 in that dimension is stretched to match the other shape.
- Rule 3: If in any dimension the sizes disagree and neither is equal to 1, an error is raised.



Boolean Masking

~based on some criterion

Counting Entries

- np.count_nonzero([condition])
- np.any([condition])-->True/False
- np.all([condition])-->True/False
- np.where([condition])-->returns index at which condition is true

*Specify axis to perform evaluation → np.[evaluation]([condition],axis=?)

Sorting Arrays

Commands

- np.sort([array])-->sorts in ascending order
- np.argsort([array])-->returns sorted indices
    - Put ",axis=?" to sort along given axis

Command Library

NumPy

Creation

- np.zeros → array full of zeros
- np.empty
- np.full→fills array with the specified value
- np.randint - integers in a given interval
- np.random - random between 0-1
- np.linspace→evenly spaced intervals(number of points)
- np.arange→generate values evenly spaced across an interval(step size)
- np.array()-->create NumPy array
    - 'dtype=' specifies element types

Modification

- array.reshape((dimension))-->reshapes specified array to specified dimension
- np.split()
    - np.vsplit(), np.hsplit()
- np.concatenate()
- np.vstack(), np.hstack(), np.dstack()

Representation + Aggregation

- np.mean()
- np.std()
- np.variance()
- np.argmin/max()-->returns index of min/max element
- np.median()
- np.percentile()

Misc.

- "Array name".copy()

Conditions

- np.count_nonzero([condition])-->returns #
- np.any([condition])-->returns True/False
- np.all([condition])-->returns True/False
- np.where([condition])-->returns index

Sort

- np.sort([array],axis=)

- np.argsort([array],axis=)

Pandas

Creation

- data=pd.series([array],index=[])
- pd.DataFrame([elements],columns=[],index=[])
- ind=pd.index([elements])

Sort

- [data_frame].sort_values(by='[column name]',ascending=True/False) → Sorts by values or by indexes
- pd.Series.sort_values([series name]) →Sorts by values or by indexes

Extraction + Indexing

- [series name].iloc[indices] → extract by position
- [series name].loc[labels] → extract by labels

Notes for Pandas

General Information

*direct assignment: 'key':value

```
states = pd.DataFrame({'population': population,
                       'area': area})
pd.DataFrame([{'a': 1, 'b': 2}, {'b': 3, 'c': 4}])
```

Series - wraps data in a sequence of values associated with indices; indexes are explicitly stated and can be modified to our liking

- data=pd.series([array],index=[])
    - data.values→returns values
    - data.index(start,stop,step)
- Indexing is the same as NumPy arrays

Dataframes - multidimensional arrays with attached row and column labels, often with heterogeneous types and/or missing data

- Similar to a series, but also has a column→two dimensional with index(think of as a row) and columns
- pd.DataFrame([elements],columns=[],index=[])

Example:

```
states = pd.DataFrame({'population': population,
                       'area': area})
states
```

	area	population
California	423967	38332521
Florida	170312	19552860
Illinois	149995	12882135
New York	141297	19651127
Texas	695662	26448193

- ind=pd.index([elements])
    - Creates an array of Pandas indexes

- Immutable but largely functions like an array

Indexing

Extraction

- [series name].iloc[indices] → extract by position
- [series name].loc[labels] → extract by labels