1. bool –

>>> True \* 12

12

>>> 0 and 1

0

2. assignment to numbers-

>>> 9x = 2 # Not allowed, must begin with a letter

>>> x9 = 2 # ok

>>> \_x = 2 # ok, but special

>>> if = 2 # must not be keyword

>>> x, y = 2, 3

>>> x, y = y, x # No temporary variables needed

>>> x = y = z = 1

>>> x += 1 # Shorthand; but no x++

3. pandas vs numpy-

-1d str in –

* Pandas= series
* Numpy = array

-series have more features, but numpy array are simpler

-pandas series are built on numpy array

-numpy array is similar to pandas List-

\* elements are in order and can be accessed by posn.

\*like list you can access range of elements in numpy array using slicing

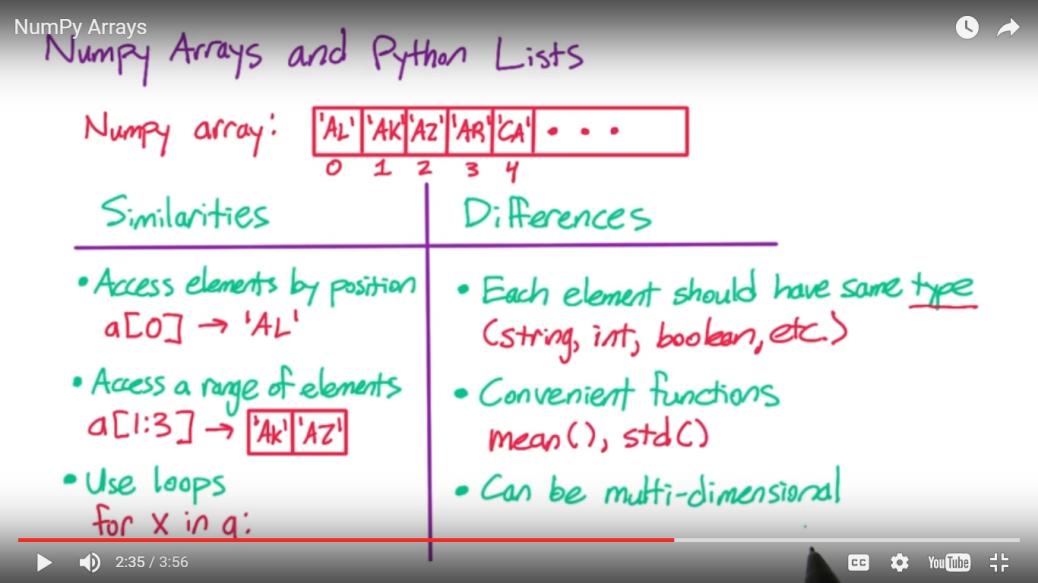
\*you can use for loops in numpy array using “**for x in a:**”

-numpy array is diff from pandas list in following ways-

\*array elements have to be of same data type, unlike list which can be a heterogeneous collection of data type.

\*numpy includes bunch of functions like mean and std; these functions can be applied to list as well, but in case of them being applied to numpy array these functions will be faster.

\*numpy can be multi dimensional; this is somewhat similar to making list of list in python;



4. numpy func-

Employment = numpy array;

Employment.max() – gives max value of employment

x=Employment.argmax 🡪 gives posn in array of element with max value; use this to get corresponding country value.

def max\_emp(countries, employment):

i=employment.argmax()

return(countries[i],employment[i])

5. to create numpy array from list-

a=np.array( [give list] )

6. print countries.dtype

o/t: |S22 🡪 means an array of Strings, with max length of a string being 22.

Print np.array([True,False,True] ).dtype

o/t: bool

7. format function-

x=’india’

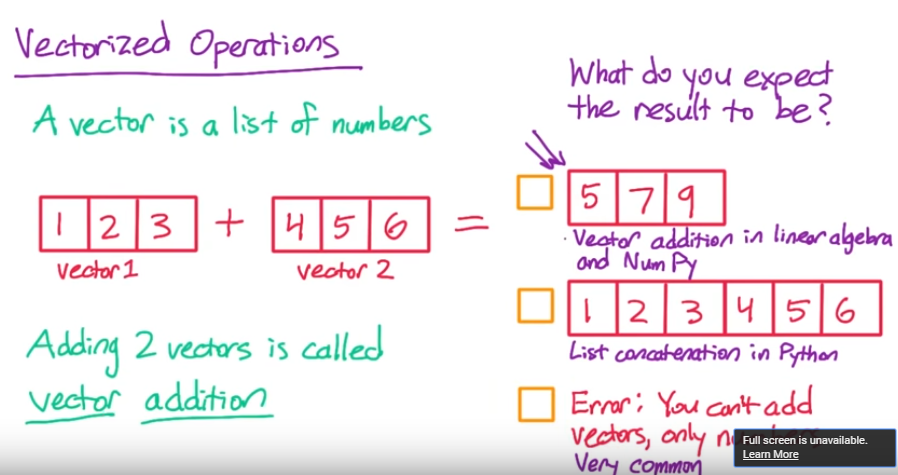
y=’40’

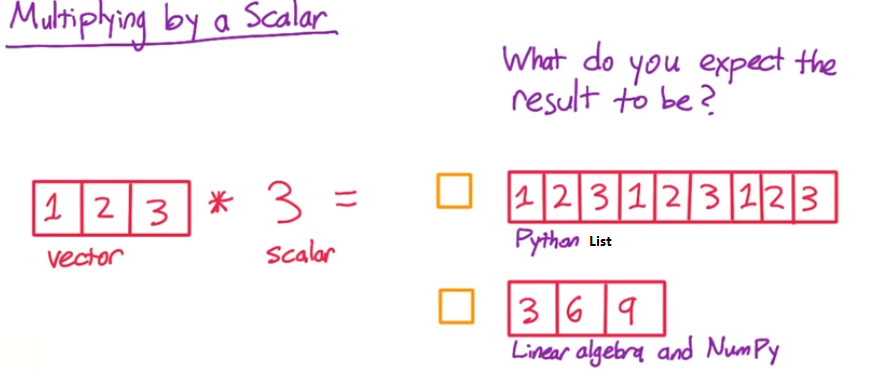
print ‘examining country {} with employment {}’.format(x,y)

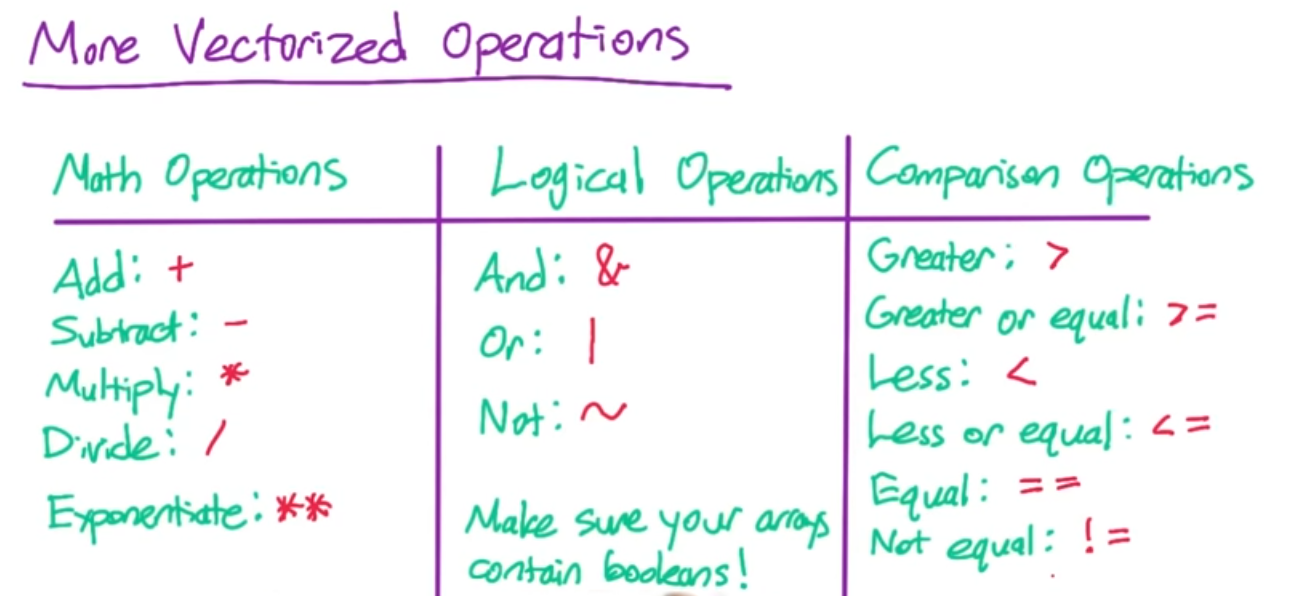
o/t: examining country India with employment 40.

8. Vectorized operations-

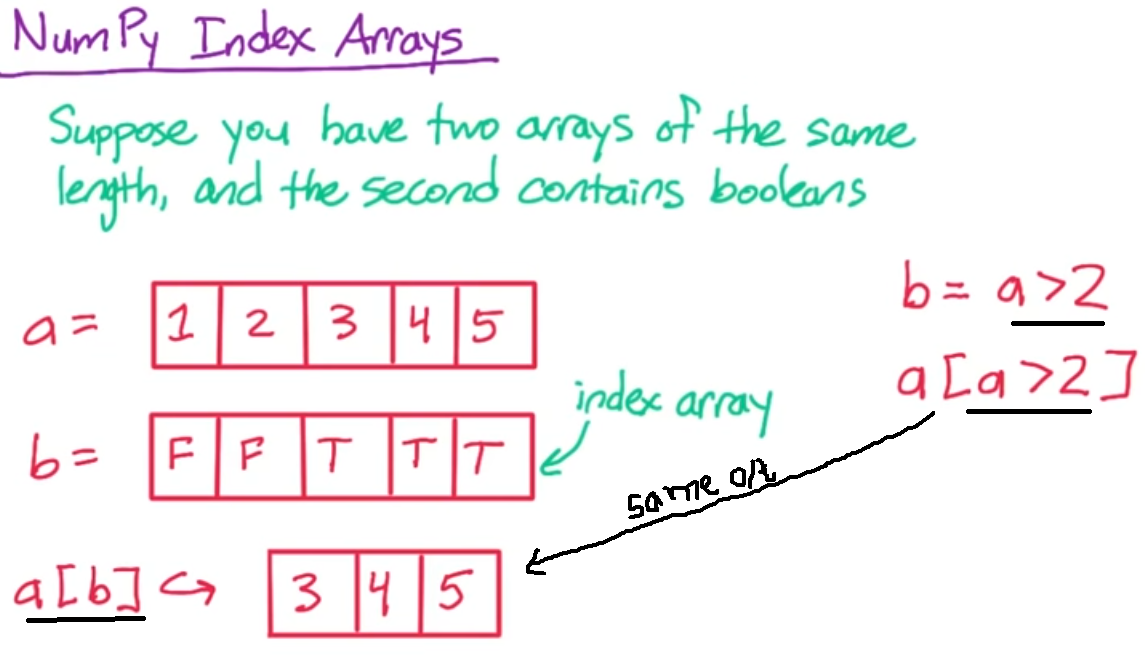
Numpy array support vector operations.



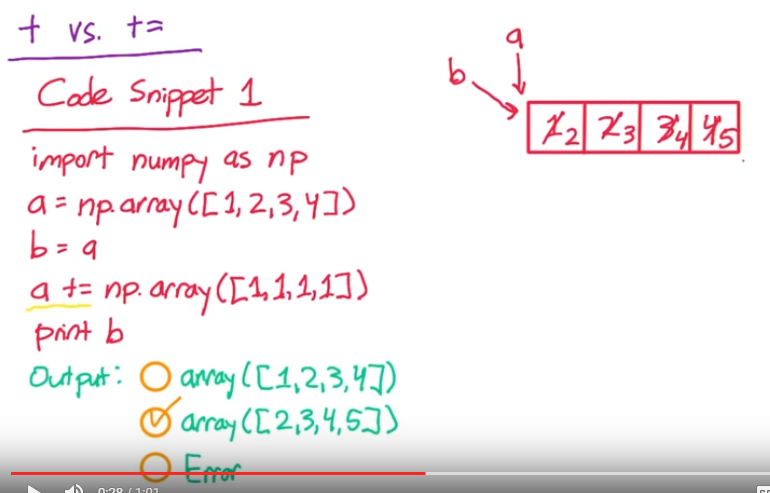


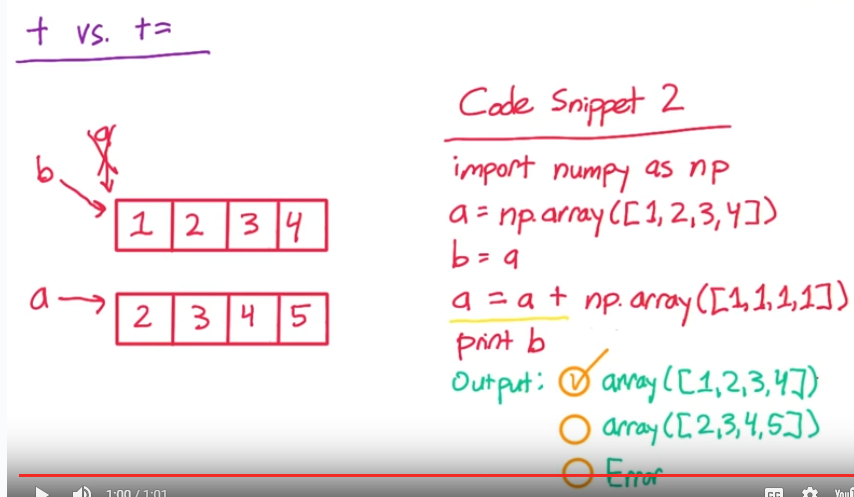


When in case of logical operations, we have 2 arrays which are not Boolean, then it results in what we call as bitwise operations.



9. **+ Vs +=**

****

****

**In this case, a=a+np.array() creates as new array to which array ‘a’ points.**

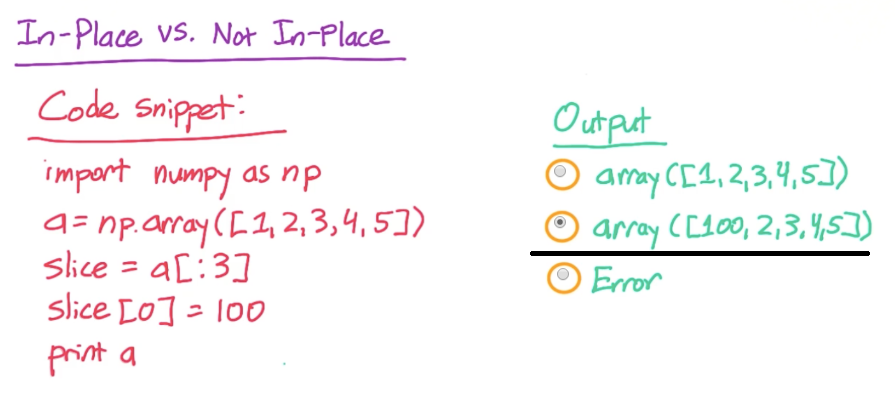
**Array ‘b’ remains unchanged and so output is [1,2,3,4]**

**‘+=’ modifies in-place**

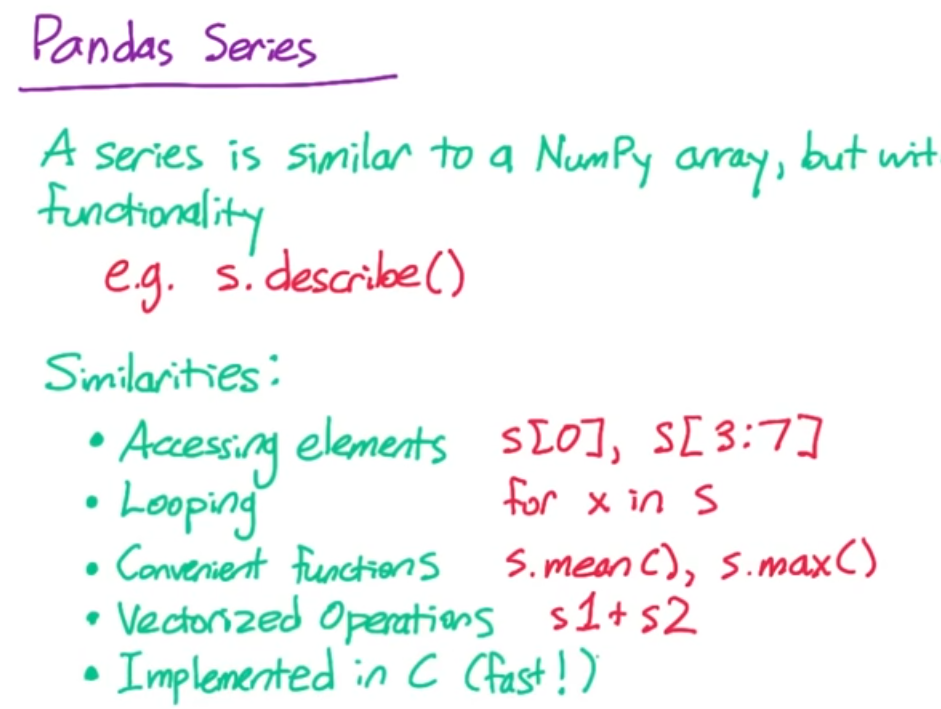
**‘+’ doesn’t modify in-place, and instead creates a new array.**

**Operations that are not in-place are much easier to deal with.**

**i.e. use ‘+’ for vectorized addition, and not ‘+=’**

****

**10. Pandas Series-**

****

**In python, you can treat Boolean as 1 and 0 for True and False.**

**So –**

**True + True =2**

**True + false=1**

**11. pandas series has indexes-**

**What are benefits of using pandas series over numpy array?**

**a. Pandas series have built-in functions that are not available with numpy array.**

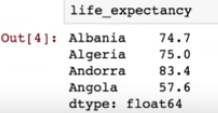
**Eg – myseries.describe 🡪 will give descriptive stats on my series – mean, min, max, p25, etc**

**b. pandas series have indexes-**

****

**In numpy array, we have to maintain 2 arrays- countries and life\_expectancy**

**In Pandas Series, this can be done in single series, with indexes as countries name.**



As seen above pandas is matching each index value to corresponding data point.

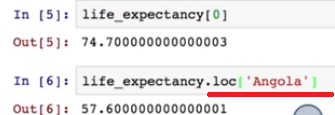
**Further, Pandas series is a cross bw a list and dictionary.**

How-

In a list, elements are stored in order, and can be accessed by their posn.

In Series, elements are stored in order, and can be accessed by their **indexes.**

In a dictionary, you have a key(like country name here) and you access its values using key(life expectancy here).



You have function ‘loc’ to access value based on index(i.e. key).

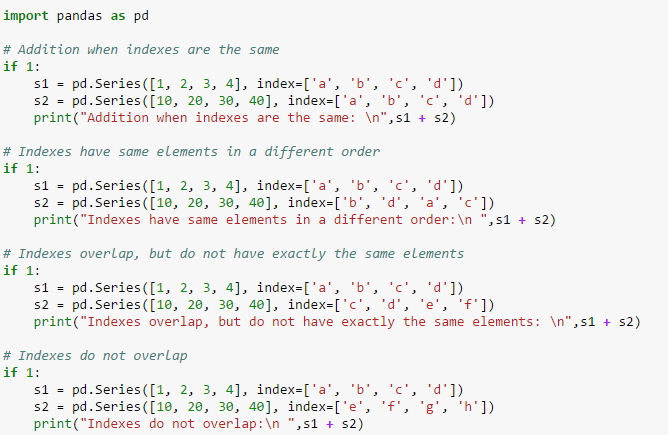
So you can access value corresponding to index angola w/o needing to know its posn in list.

Note- life\_expectancy[0] = accessing series value by posn.

Same can be done using “iloc()’ function 🡪 life\_expectancy.iloc[0]

You must remember that in pandas series, index and posn are not same.

12. vectorized operations on Series are done not by posn, but by indexes-



Output:

Addition when indexes are the same:

a 11

b 22

c 33

d 44

dtype: int64

Indexes have same elements in a different order:

a 31

b 12

c 43

d 24

dtype: int64

Indexes overlap, but do not have exactly the same elements:

a NaN

b NaN

c 13.0

d 24.0

e NaN

f NaN

dtype: float64

Indexes do not overlap:

a NaN

b NaN

c NaN

d NaN

e NaN

f NaN

g NaN

h NaN

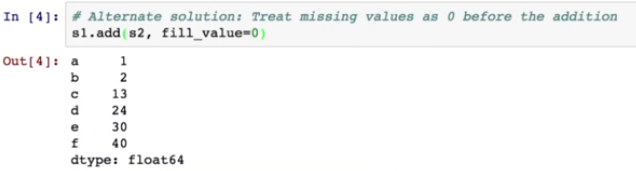
dtype: float64

13. Treating missing values in a series-

dropna() function



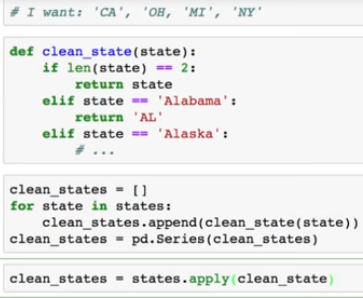
**Other way to do this is to treat missing values as 0s- by using add() and fill\_value= option**



14. apply() –

-to apply user defined function in pandas series-





Syntax: myseries.apply(mufunction)

## 15. Plotting in Pandas

If the variable data is a NumPy array or a Pandas Series, just like if it is a list, the code

import matplotlib.pyplot as plt

plt.hist(data)

will create a histogram of the data.

Pandas also has built-in plotting that uses matplotlib behind the scenes, so if data is a Series, you can create a histogram using data.hist().

There's no difference between these two in this case, but sometimes the Pandas wrapper can be more convenient. For example, you can make a line plot of a series using data.plot(). The index of the Series will be used for the x-axis and the values for the y-axis.

Pandas dataframes and 2d numpy array