Numpy → vectorized library similar to MATLAB

→ ndarray → multidimensional array that supports array-oriented arithmetic and flexible broadcasting capabilities

→ vectorized functions

→ linear algebra, random number generation

→ stack of data-science, scientific computing tools in python.

numpy arrays or the default data type for all of those

```
import numpy as np

x = np.array([1, 2, 3])
y = np.array([3, 0, 3])
```

x ⟹ array([1,2,3])

⟹ generic multidimensional container
for homogeneous data → all elements
have the same data type

data2 = [[1,2,3,4], [5,6,7,8]]

arr2 = np.array(data2)

arr2.ndim ⟹ 2

arr2.shape ⟹ (2,4)

⟹ can create arrays of zeros and ones

np.zeros(10) np.ones(10)

np.zeros((3,6)) np.ones((3,6))

⟹ np.arange(15)

array([0, 1, 2, ..., 14])

np.linspace(start, stop, num)

np.linspace(0, 3, 5)

arr([0, 0.7, 1., 2.2, 3.0])

→ arithmetic

→ access (modify) subsets

→ apply functions to the array

→ linear algebra

arithmetic

⟹ given two numpy arrays with the same shape

a = np.arr([1,2]) b = np.arr([3,4])

a+b ⟹ arr([4,6])

a*b ⟹ arr([3, 8])

⟹ perform an arithmetic operation between arr and scalar

1/a = arr([1, 1/2])

a*2 = arr([1,4])

a > 1 = array([False, True])

Indexing / slicing

arr = np.array([1,2,3])

arr[1] = 2

arr[1:] = array([2,3])

arr[1:] = 12

arr ⇒ array([1,12,12])

arr_slice = arr[1:]

arr_slice = 7

arr ⇒ array([1,7,7])

arr2 = arr.copy()

---

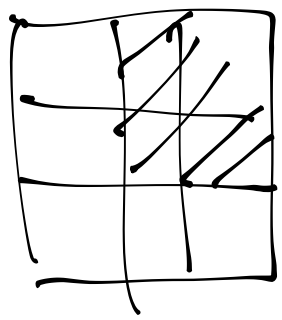
arr2d = np.array([[1,2,3], [4,5,6],
[7,8,9]])

arr2d[2] ⇒ [7, 8, 9]

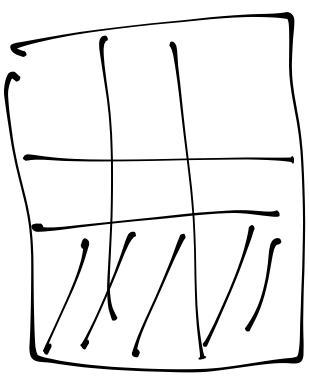
arr2d[2,0] ⇒ 7

arr2d[:2] ⇒ [[1,2,3],
[4,5,6]]

arr2d[:2,1:] ⇒
[[2,3],
[5,6]]



arr[:2,1:] (2,2)



arr[2] (3,)

arr[2:,:] (1,3)

Boolean indexing

arr = np.arr([1,2,3,4,5])

arr > 3 ⟹ array([F, F, F, T, T])

arr[arr > 3] ⟹ array([4, 5])

arr[arr != 3] ⟹ array([1, 2, 4, 5])

Can't use and

or

instead use & or |

arr[(arr != 3) & (arr != 2)]

= array([1, 4, 5])

Universal functions: element-wise operations on data in ndarray

np.sqrt(x) ⟹ returns the square root of every element

np.exp(x) ⟹ e^x for each element

np.maximum(x, y) ⟹ element-wise compares, returns element-wise max

x = [1, 2, 3]

y = [0, 4, 7] np.maximum(x, y)

= [1, 4, 7]

np.sum()

np.mean()

arr = [[0, 1],
[2, 3]]

np.sum(arr) ⇒ 6

np.sum(arr, axis=0) ⇒ [2, 4]

np.sum(arr, axis=1) ⇒ [1, 5]