chapter-1 NUMPY

2-1 Numpy

Preparing numpy

Page No.
Date :

```
import numpy as np
np.random.seed  # seed for reproducibility
x1 = np.random.randint(10, size=6)  # 1-D array
x2 = np.random.randint(10, size=(3,4))  # 2-D array
x3 = np.random.randint(10, size=(3,4,3))  # 3-D array
```

→ Attribute of Arrays :-
Determining the size, shape, memory consumption and data types of arrays.

→ Indexing of Arrays :-
Getting and setting the values of individual array elements

→ Slicing of Arrays :-
Getting and setting the ~~values~~ smaller subarrays within a larger array.

→ Reshaping of arrays :-
Changing the shape of a given array

→ Joining and Splitting of Arrays :-
Combining multiple arrays into one, and splitting one array into many

→ Each array has attributes ndim (the number of dimensions)
→ shape (the size of each dimension), and
size (the total size of the array).

Date :

```
print("x3 ndim: ", x3.ndim)
print("x3 shape: ", x3.shape)
print("x3 size: ", x3.size)
print("dtype:", x3.dtype)
```

-> Other attributes include itemsize which are (in bytes) of each array element, and nbytes which lists the total size of the array:

```
print("itemsize:", x3.itemsize, "bytes")
print("nbytes:", x3.nbytes, "bytes")
```

-> Array indexing : Accessing Single Elements
You can access the ith value (counting from zero)

```
x1
```

output array([5, 0, 3, 3, 7, 9])

```
x1[0]
```

output 5

```
x1[4]
```

output 7

To index from the end of the array

```
x1[-1]
```

output 9

```
x1[-2]
```

output 7

In multidimensional array, you can access item using a comma-sep tuple of indices

```
n2
array([[3,5,2,4],
       [7,6,8,8],
       [1,6,7,7]])
```

```
n2[0,0]
3
```

```
n2[2,0]
1
```

```
n2[2,-1]
7
```

mp

n[start : stop : step]

start = 0,
stop = size of dimension
step = 1.

Python lists, Numpy arrays have a fixed type. It means that if you attempt to insert a floating-point value to an integer array, the value will be silently truncated.

```
x1[0] = 3.1459  # It will be truncated.
x1[calling it]
```

Array Slicing : Accessing Subarrays

We can use access them to sub-arrays with the slice notation, marked by colon.

n[start : stop : step]

1-D Array

```
x = np.arange(10)
x
Array([0,1,2,3,4,5,6,7,8,9])
x[:5] # first five elements
Array([0,1,2,3,4])
x[5:] # elements after index 5
Array([5,6,7,8,9])
x[4:7] # middle subarray
Array([4,5,6])
x[::2] # Every other element
array([0,2,4,6,8])
x[1::2] # Every other element, starting at index 1
array([1,3,5,7,9])
```

Now, when the step value is negative. The default for start and stop are swapped

```
x[::-2)
array([9,8,7,6,5,4,3,2,1,0])
x[5::-2] # reversed every other from index 5
```

Multidimensional Array

In multidimensional slices work in the same way, with multiple slices separated by commas.

```
n2
output array([[12, 5, 2, 4],
              [7, 6, 8, 8],
              [1, 6, 7, 7]]).

n2[:2, :3]  # Two rows, Three columns
output array([[12, 5, 2],
              [7, 6, 8]]).

n2[:3, ::2]  # All Rows, Every other columns
output array([[12, 2],
              [7, 8],
              [1, 7]]).

n2[::-1, ::-1]
output array([[7, 7, 6, 1],
              [8, 8, 6, 7],
              [4, 2, 5, 12]]).
```

Accessing array rows and columns.

Date :

→ Accessing single rows or columns of an array you can do this by combining indexing and slicing, using an empty slice marked by a single colon (:).

→ Print(x2[:,0]) # first column of x2.
[12 7 1].

→ Print(x2[0,:]) # first row of x2.
[12 5 2 4].

→ Print(x2[0]) # Equivalent to x2[0,:].
[12 5 2 4].

Subarrays as no-copy views

→ Important and useful thing is about array slices is that they return views rather than copies of the array data. This is one area in which numpy array slicing differs from python list slicing: in lists slices will be copies

For Example :-

→ Print(x2)
[[12 5 2 4]
[7 6 8 8]
[1 6 7 7]]

Lets extract at 2×2 subarray from this
x2_sub[0,0] = 99.
Print(x2_sub), Print(x2).

[[99 5]
[7 6]]

Creating a copy of array

Page No. 1
Date :

→ `x2_sub_copy = x2[:2, :2].copy()`
`print(x2_sub_copy)`

```
[[99  5
  7  6]]
```

modifying this subarray, the org array is not touched

→ `x2_sub_copy[0,0] = 42`
`print(x2_sub_copy)`

```
[[42  5
  7  6]]
```

→ `print(x2)`

```
[[99  5  2 4]
 [7  6  8 8]
 [1  6  7 7]]
```

Reshaping of Arrays

→ You can Reshape the array by using reshape() method.
→ If you want to put the numbers 1 through 9 in a 3×3 Grid

→ `grid = np.arange(1,10).reshape((3,3))`
→ `print(grid)`

output

```
[[1  2  3
 4  5  6
 7  8  9]]
```

→ The size of initial array must match the size of the reshaped array.

→ x = np.array([1,2,3]).
→ x.reshape((1,3)). # Row Vector Via reshape.
output array([[1,2,3]]).

→ x[np.newaxis,:] # Row Vector Via newaxis
output array([[1,2,3]]).

→ x.reshape((3,1)) # Column Vector Via reshape
output array([[1],
[2]
[3]]).

→ x[:, np.newaxis].
output array([[1],
[2]
[3]]).

Array Concatenation and Splitting.

Concat of arrays :-

→ methods using in concatenation are (i) np.concatenate (ii) np.vstack (iii) np.hstack (iv) np.concatenate

x = np.array([1,2,3]).
y = np.array([3,2,1]).
np.concatenate([x,y])
output array([1,2,3,3,2,1]).

→ we can concatenate more than two arrays at once.
z = [99,99,99]
print(np.concatenate([x,y,z]))

Concat two -D Arrays

→ grid = np.array([[1,2,3],
[4,5,6]]).

→ np.concatenate([grid, grid]) # Concat at first axis.
array([grid, grid]).

output Array([[1,2,3],
[4,5,6],
[1,2,3],
[4,5,6]]).

→ np.concatenate([grid, grid], axis=1).
Concat along the second axis (zero-indexed).

output array([[1,2,3,1,2,3],
[4,5,6,4,5,6]]).

For arrays with mixed dimensions.
use np.vstack and (vertical stack)
np.hstack (horizontal stack).

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

Vertically stack the arrays.
→ np.vstack([x, grid]).

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

horizontally stack the arrays.

→ y = np.arrays([[99],
[99]]).

→ np.hstack([grid, y]).

output array([[9, 8, 7, 99],
[6, 5, 4, 99]]).

Similarly np.dstack will stack arrays along the third axis

Splitting of arrays

The opposite of concatenation is splitting.

Functions used to split arrays are:-
(i) np.split
(ii) np.hsplit
(iii) np.vsplit

→ x = [1, 2, 3, 99, 99, 3, 2, 1].
→ x1, x2, x3 = np.split(x, [3, 5]).
→ Print(x1, x2, x3)
Output [1 2 3] [99 99] [3 2 1]

→ grid = np.arange(16).reshape((4, 4)).
→ grid
Output array([[0, 1, 2, 3],
[4, 5, 6, 7],
[8, 9, 10, 11],
[12, 13, 14, 15]]).

→ upper, lower = np.vsplit(grid, [2])

→ print(upper)

→ print(lower)

output

```
[[0 1 2 3]
 [4 5 6 7]]

[[8 9 10 11]
 [12 13 14 15]]
```

→ left, right = np.hsplit(grid, [2])

→ print(left)

print(right)

Computation on numpy arrays: Universal function

→ Talking about speed of numpy and how time% works

→ For making numpy arrays working fast or such things you can actually use Vectorized operations.

→ Generally they are implemented through numpy universal function.

For Example.

```
import numpy as np
np.random.seed(0)

def compute_reciprocals(values):
    output = np.empty(len(values))
    for i in range(len(values)):
        output[i] = 1.0 / values[i]
    return output
values = np.random.randint(1,10, size=5)
compute_reciprocals(values)
```

```
big_array = np.random.randint(1,100,size=1000000)
%time compute_reciprocals(big_array)
```

→ Vectorized operations in Numpy are implemented via ufuncs. purpose is to quickly execute repeated operations on values in numpy arrays

Implementing This Ufuncs with two Arrays.

→ np.arange(5) / np.arange(1,6) # 2 Arrays.
output array([0., 0.5, 0.66667, 0.75, 0.8])

→ x = np.arange(9).reshape((3,3))
→ 2 ** x
output array([[1, 2, 4],
[8, 16, 32],
[64, 128, 256]]).

Exploring Numpy's Ufuncs

→ Ufuncs exist in two flavors:-
(i) Unary functions :- operates on single input.
(ii) binary functions:- operates on two inputs.

Using funcs in Array arithmetic

→ x = np.arange(4).
→ print("x =", x)
→ print("x+5 =", x+5).
→ print("x-5 =", x-5)
→ print("x*2 =", x*2)
→ print("x/2 =", x/2)
→ print("x//2 =", x//2) # floor division.

unary has funcs for negations, a ** operator for exponentiation, and a % operator for modulus.

→ Print ("-x =", -x)
→ Print ("x ** 2 =", x ** 2)
→ Print ("x % 2 =", x % 2).

Python absolute value functions

→ x = np.array([-2, -1, 0, 1, 2]).
→ abs(x).

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

→ It can be used for complex data in which absolute value returns the magnitude:

→ x = np.array([3-4j, 4-3j, 2+0j, 0+2j]).
np.abs(x).

output array([5., 5., 2., 2.]).

Ninja functions

→ theta = np.linspace(0, np.pi, 3).
→ Print ("theta =", theta)
→ Print ("sin(theta) =", np.sin(theta))
→ Print ("cos(theta) =", np.cos(theta))
→ print ("tan(theta) =", np.tan(theta)).

Specialized ufuncs

- hyperbolic trig functions.
- bitwise arithmetic.
- comparison operators.
- conversions from radians to degrees

Another ufuncs is the submodule scipy.special.

```
from scipy import special.
x = [1, 5, 10] # gamma fns and related functions.
print("gamma(x)     =", special.gamma(x))
print("ln|gamma(x)| =", special.gamma_ln(x))
print("beta(x,2)    =", special.beta(x, 2)).
```

Summing the values in an array

```
import numpy as np
L = np.random.random(100)
sum(L)
big_array = np.random.rand(1000000)
%timeit sum(big_array)
%timeit np.sum(big_array)
```

np:- sum and np.sum are diff they have diff abilities and speed to perform the calculations

→ min(big_array), max(big_array).
→ np.min(big_array), np.max(big_array).
→ %timeit min(big_array)
→ %timeit np.min(big_array)

Boolean operator using and/OR operator. &/|

→ bool(42), bool(0).
→ bool(42 and 0).
→ bool(42 or 0).

→ bin(42).
→ bin(59).
→ bin(42&59).
→ bin(42|59).

Fast sorting in numpy: np.sort and np.argsort

→ x = np.array([2,1,4,3,5]).
→ np.sort(x)
output array([1,2,3,4,5]).
→ x.sort()
print(x)

★ A Related function is argsort, which instead returns the indices of the sorted elements.

→ x = np.array([2,1,4,3,5])
→ i = np.argsort(x).
→ print(i).

output [1 0 3 2 4]

→ x[i]
output:- array([1,2,3,4,5])

Partial Sorts : Partitioning

```
x = np.array([7,2,3,1,6,5,4])
np.partition(x,3)
```

output: array([2,1,3,4,6,5,7])

Here the value of the array is starting from the smallest numbers which are 2,1,3; the result is a new array with the smallest k-values to the left of partition

```
np.partition(X, 2, axis=1)
x = rand.rand(10,2)
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn; seaborn.set() # plot styling
plt.scatter(X[:,0], X[:,1], s=100);
```