UNIST STEM Camp 2020

Python – NumPy

Numpy

- A library consisting of multidimensional array objects and a collection of routines for processing those arrays.

- How to use?

```
import numpy as np
```

Creating Array

- Syntax

```
numpy.array(object, dtype=None,
copy=True)
```

- object: array_like
  - An array

- dtype: data-type, optional
  - The desired data-type for the array
  - https://docs.python.org/3/library/stdtypes.html

- copy: bool, optional
  - If true, then the object is copied.

Creating Array

- Examples

```
import numpy as np

a = np.array([1,2,3])
b = np.array([(1,2,3,4),(7,8,9,10)],
dtype = int)
```

Create Array

- Syntax

```
numpy.zeros(shape, dtype=float,
order='C')
```

- shape: int or tuple of ints
  - Shape of the new array, e.g., (2, 3) or 2.
- dtype: data-type, optional
  - The desired data-type for the array, e.g., numpy.int8. Default is numpy.float64.
- order: {‘C’, ‘F’}, optional, default: ‘C’
  - Whether to store multi-dimensional data in row-major (C-style) or column-major (Fortran-style) order in memory.

Initial Placeholders

- Examples

```
import numpy as np

np.zeros(3)  # 1D array of length 3 all zeros
    array([0., 0., 0.])
np.zeros((2,3))  # 2D array of all zeros
    array([0., 0., 0.],
                [0., 0., 0.])
```

Initial Placeholders

- Examples

```
import numpy as np

np.zeros((3, 2, 4))
       [[[0. 0. 0. 0.]
         [0. 0. 0. 0.]]

        [[0. 0. 0. 0.]
         [0. 0. 0. 0.]]

        [[0. 0. 0. 0.]
         [0. 0. 0. 0.]]]
```

Initial Placeholders

- Examples

```
np.full((2, 3), 2)
# 2 X 3 array with all values 2
[[2, 2, 2],
 [2, 2, 2]]

np.random.rand(3, 4)
# 3 X 4 array of random floats between 0-1
[[0.18541753 0.02535141 0.60898043 0.12037671]
 [0.86784702 0.24748429 0.83820784 0.37462509]
 [0.94776286 0.56500454 0.18805111 0.53407793]]
```

Initial Placeholders

- Examples

```
import numpy as np

np.ones((3, 4))
# 3 X 4 array with all values 1

np.eye(4)
# 4 X 4 array of 0 with 1 on diagonal
[[1 0 0 0],
 [0 1 0 0],
 [0 0 1 0],
 [0 0 0 1]]
```

Saving and Loading

- On disk

```
import numpy as np

x = np.random.rand(3, 4)

np.save("new_array", x)
# save ndarray x as new_array.npy file

y = np.load("new_array.npy")
# load ndarray from "new_array.npy" file
# and save it in y
```

Saving and Loading

- Text/Comma Separated Values(CSV)

Main difference between text file(.txt) and csv file(.csv) is that delimiter for text file is ‘ ‘ (blank) and which for csv file is ‘,’ (comma).

For example, a = np.array([2, 3, 4]) is saved as

in .txt file	in .csv file
2	2**,**
3	3**,**
4	4

Saving and Loading

- Text/Comma Separated Values(CSV)

```
import numpy as np

np.loadtxt("old_file.txt")
# From a text file
np.genfromtxt('new_file.csv', delimiter=',')
# From a csv to txt
np.savetxt('file.txt', arr, delimiter=' ')
# saving 'arr' as txt
np.savetxt('file.csv', arr, delimiter=',')
# saving 'arr' as csv
```

Saving and Loading

- Properties

```
import numpy as np

array.size
# number of elements in array
array.shape
# dimensions of array (row, columns)
array.dtype
# type of elements in array
# e.g., array.dtype == int

Note, it is not function, so
You should not use like array.size()
```

Copying

- Syntax

```
numpy.copy(a, order='K')
```

- a : array_like
  - Input data.
- order : {‘C’, ‘F’, ‘A’, ‘K’}, optional
  - Controls the memory layout of the copy. ‘C’ means C-order, ‘F’ means F-order, ‘A’ means ‘F’ if a is Fortran contiguous, ‘C’ otherwise. ‘K’ means match the layout of a as closely as possible. (Note that this function and ndarray.copy are very similar, but have different default values for their order= arguments.)

Operations

- Copying

```
import numpy as np

x = np.zeros((3, 4))

y = np.copy(x)
# copies x to new memory array

print(x.view(dtype=int))
# creates view of array elements with dtype
```

Operations

- Copying

```
x = np.zeros((2, 3))

print(x.view(dtype=int))
[[0, 0, 0],
 [0, 0, 0]]

print(x.view(dtype=float))
[[0., 0., 0.],
 [0., 0., 0.]]
```

Operations

- Sorting

```
import numpy as np

x = np.zeros((3, 4))

x.sort()                    # sorts array
x.sort(axis=0)  # sorts by specific axis
x.reshape(4, 3) # reshapes to (4, 3)
                                  # without
changing data
```

Operations

- Sorting

```
import numpy as np

a = np.array([[1,4],[3,1]])
print(np.sort(a, axis=0))
[[1 1]
 [3 4]]
print(np.sort(a, axis=1))
[[1 4]
 [1 3]]
```

UNIST FIRST IN CHANGE

Adding

- Syntax

```
numpy.append(arr, values, axis=None)
```

- arr : array_like
  - Values are appended to a copy of this array.
- values : array_like
  - These values are appended to a copy of arr. It must be of the correct shape (the same shape as arr, excluding axis). If axis is not specified, values can be any shape and will be flattened before use.
- axis : int, optional
  - The axis along which values are appended. If axis is not given, both arr and values are flattened before use.

Adding

- Syntax

```
numpy.insert(arr, obj, values,
axis=None)
```

- arr : array_like
  - Input array.
- obj : int, slice or sequence of ints
  - Object that defines the index or indices before which values is inserted.
- values : array_like
  - Values to insert into arr. If the type of values is different from that of arr, values is converted to the type of arr.
- axis : int, optional
  - The axis along which values are appended. If axis is not given, both arr and values are flattened before use.

Operations

- Adding

```
import numpy as np

x = np.zeros(4)

y = np.append(x, [4, ])
# appends 4 to end of x
z = np.insert(y, 3, values)
# insert values before index 3
```

Removing

- Syntax

```
numpy.delete(arr, obj, axis=None)
```

- arr: array_like
  - Input array
- obj: slice, int or array of ints
  - Indicate which sub-arrays to remove
- axis: int, optional
  - The axis along which to delete the subarray defined by obj. If axis is None, obj is applied to the flattened array.

Operations

- Removing

```
import numpy as np

a = np.array([(1,2,3,4),(7,8,9,10)])

b = np.delete(a, 2, axis=1)
#deletes column(axis=1) on index 2 of a
c = np.delete(a, 1, axis=0)
#deletes column(axis=0) on index 1 of a
```

Combining

- Syntax

```
numpy.concatenate((a1, a2, ...),
axis=0, out=None)
```

- a1, a2, … : sequence of array_like
  - The arrays must have the same shape, except in the dimension corresponding to axis (the first, by default).
- axis : int, optional
  - The axis along which the arrays will be joined. If axis is None, arrays are flattened before use. Default is 0.
- out : ndarray, optional
  - If provided, the destination to place the result.

Operations

- Combining

```
import numpy as np

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

c = np.concatenate((a, b), axis=1)
#adds b as columns(axis=1) to the end of a
d = np.concatenate((a, b), axis=0)
#adds b as rows(axis=0) to the end of a
```

Splitting

- Syntax

```
numpy.split(ary,
indices_or_sections, axis=0)
```

- ary : ndarray
  - Array to be divided into sub-arrays.
- indices_or_sections : int or 1-D array
  - If indices_or_sections is an integer, N, the array will be divided into N equal arrays along axis.
  - If an index exceeds the dimension of the array along axis, an empty sub-array is returned correspondingly.
- axis : int, optional
  - The axis along which to split, default is 0.

Operations

- Splitting

```
import numpy as np

a = np.array([(1,2,3,4),(7,8,9,10)])

b = np.split(a, 2)
#splits a into 2 sub-arrays
```

Operations

- Indexing

```
import numpy as np

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

a[0] = 5
#assigns element of a on index 0 the value 5
b[0,1] = 1
#assigns element of b on index [0][1] the
value 1
```

UNIST FIRST IN CHANGE

Operations

- Subseting

```
import numpy as np

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

a[2]
#returns the element of index 2 in array a
b[1,0]
#returns the array element on index [1][0]
```

Operations

- Slicing

```
import numpy as np

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

a[0:3] #returns elements at indices 0,1,2
b[0:2, 1] #returns elements on rows 0,1 at
column 1
a[:2] #returns elements at indices 0,1
b[:,0] #returns elements at index 0 on all
rows
```

Array Mathematics

- Arithmetic operations

```
import numpy as np

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

np.add(a,b) #adds a to b
np.subtract(a,b) #subtracts b from a
np.multiply(a,b) #multiplies a by b
np.divide(a,b) #divides a by b
np.exp(a) #exponential of a
np.sqrt(b) #square root of b
```

Splitting

- Syntax

```
numpy.array_equal(a1, a2)
```

- a1, a2 : array_like
  - Input arrays.
- b : bool
  - Returns True if the arrays are equal.

Array Mathematics

- Comparison

```
import numpy as np

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

a == b #element-wise comparison
np.array_equal(a,b) #array-wise comparison
```

Functions

- Examples

```
import numpy as np

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

a.sum() #array-wise sum
a.min() #array-wise min value
a.max(axis=0) #array row max value
a.mean() #mean
a.median() #median (not allowed)
```