python-numpy

September 28, 2023

1 ————————-NumPy————————

NumPy is a Python library.

NumPy is used for working with arrays.

NumPy is short for “Numerical Python”

NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.

[3]:
```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
print(arr)
print(type(arr))
print(np.__version__)
```

```
[1 2 3 4 5]
<class 'numpy.ndarray'>
1.24.3
```

[4]:
```
#Number of Dimensions
import numpy as np

a = np.array(42) #0 d array
b = np.array([1, 2, 3, 4, 5]) #1d- array
c = np.array([[1, 2, 3], [4, 5, 6]]) #2d-array
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]]) #3-D arrays

print(a.ndim)
print(b.ndim)
print(c.ndim)
print(d.ndim)
```

```
0
1
2
3
```

2 NumPy Array Slicing

[5]:
```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[1:5])
print(arr[4:])
print(arr[:4])
print(arr[-3:-1])
print(arr[1:5:2])
print(arr[::2])
print(arr[0:2])
```

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

3 NumPy Array Copy vs View

The main difference between a copy and a view of an array is that the copy is a new array, and the view is just a view of the original array.

The copy SHOULD NOT be affected by the changes made to the original array.

The view SHOULD be affected by the changes made to the original array.

[7]:
```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])
x = arr.copy()
arr[0] = 42

print(arr)
print(x)
```

```
[42  2  3  4  5]
[1 2 3 4 5]
```

[10]:
```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])
x = arr.view()
arr[0] = 42
```

```
print(arr)
print(x)
```

```
[42  2  3  4  5]
[42  2  3  4  5]
```

[9]:
```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])
x = arr.view()
x[0] = 31

print(arr)
print(x)
```

```
[31  2  3  4  5]
[31  2  3  4  5]
```

4 Joining NumPy Arrays

[14]:
```
import numpy as np

arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
arr = np.concatenate((arr1, arr2)) #Join two arrays
arr6 = np.stack((arr1, arr2), axis=1) #Joining Arrays Using Stack Functions
arr7 = np.hstack((arr1, arr2)) #NumPy provides a helper function: hstack() to␣
 ↪stack along rows.
arr8 = np.vstack((arr1, arr2)) #NumPy provides a helper function: vstack()  to␣
 ↪stack along columns.
arr9 = np.dstack((arr1, arr2)) #NumPy provides a helper function: dstack() to␣
 ↪stack along height, which is the same as depth.

print(arr)
print("------********-------")
print(arr6)
print("------********-------")
print(arr7)
print("------********-------")
print(arr8)
print("------********-------")
print(arr9)
```

```
[1 2 3 4 5 6]
------********-------
[[1 4]
 [2 5]
```

```
 [3 6]]
-------**********-------
[1 2 3 4 5 6]
-------**********-------
[[1 2 3]
 [4 5 6]]
-------**********-------
[[[1 4]
  [2 5]
  [3 6]]]
```

5 Searching Arrays

[19]:
```
import numpy as np

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

x1 = np.where(arr == 4) #To search an array, use the where() method.
x2 = np.where(arr%2 == 0) #Find the indexes where the values are even:
x3 = np.where(arr%2 == 1) #Find the indexes where the values are odd:
x4 = np.searchsorted(arr, 6) #The searchsorted() method is assumed to be used
 ↪on sorted arrays.Find the indexes where the value 6 should be inserted:

print(x1)
print(x2)
print(x3)
print(x4)
```

```
(array([3], dtype=int64),)
(array([1, 3, 5, 7], dtype=int64),)
(array([0, 2, 4, 6], dtype=int64),)
5
```

6 Sorting Arrays

Sorting means putting elements in an ordered sequence.

[20]:
```
import numpy as np

arr1 = np.array([3, 2, 0, 1])
arr2 = np.array([[3, 2, 4], [5, 0, 1]])
arr3 = np.array(['banana', 'cherry', 'apple'])
arr4 = np.array([True, False, True])

print(np.sort(arr1))
print(np.sort(arr2))
print(np.sort(arr3))
```

```
print(np.sort(arr4))
```

```
[0 1 2 3]
[[2 3 4]
 [0 1 5]]
['apple' 'banana' 'cherry']
[False  True  True]
```

[ ]: