NumPy part-2

26-02-2024

By Anam Suri, NET-JRF

Creating arrays

ndarrays are typically created using two main methods:

1.From existing data (usually lists or tuples) using np.array(), already done

2.Using built-in functions such as np.arange(), np.linspace(), np.zeroes etc.

```
np.arange(1, 5)  # from 1 inclusive to 5 exclusive
np.arange(0, 11, 2)  # step by 2 from 1 to 11
np.linspace(0, 10, 5)  # 5 equally spaced points between 0 and 10
np.ones((2, 2))  # an array of ones with size 2 x 2
np.zeros((2, 3))  # an array of zeros with size 2 x 3
np.full((3, 3), 3.14)  # an array of the number 3.14 with size 3 x 3
np.full((3, 3, 3), 3.14)  # an array of the number 3.14 with size 3 x 3 x 3
```

```
x = np.random.rand(5, 2) # generates a NumPy array of shape (5,2) filled with random float numbers between 0 and 1
Print(x)

x.transpose()
x.mean()
x.astype(int) #resulting array will contain array  containing integer part of every number
```

If you want to generate random numbers within a specific range, such as between 3 and 5, you can use:

```
np.random.uniform(3,5, size = (5,2))
```

Array shapes

.ndim: the number of dimensions of an array

.shape: the number of elements in each dimension (like calling len() on eah dimension

.size: the total number of elements in an array (i.e., the product of .shape())

```
array_1d = np.ones(3)
print(f"Dimensions: {array_1d.ndim}")
print(f"    Shape: {array_1d.shape}")
print(f"     Size: {array_1d.size}")
```

Task: Turn the above print actions into a function and call that function with different other arrays (Function name = array_nd).

```
array_2d = np.ones((3, 2))
print_array(array_2d) #Call function

np.array_equal(x,y) #to check if the size of two arrays is equal
```

```
def print_array(x):
    print(f"Dimensions: {x.ndim}")
    print(f"    Shape: {x.shape}")
    print(f"     Size: {x.size}")
    print("")
    print(x)
```

NumPy Slice Is a Reference

- The result of NumPy slice is a reference and not a copy of the original array.
  - Try creating an array, slice it and assign the slice to any variable (say *b*).
  - Change one of the elements in *b*.
  - Now check if the original array has the modified element or not.

- Note: The result of slicing depend on how you slice it
  - Check: b1 = arr[2:, :] and b2 = arr[2, :]
  - Print the shapes of both the arrays

NumPy Data Types

NumPy has some extra data types, and refer to data types with one character, like i for integers, u for unsigned integers etc.
Below is a list of all data types in NumPy and the characters used to represent them.

- i - integer
- b - boolean
- u - unsigned integer
- f - float
- c - complex float
- m - timedelta
- M - datetime
- O - object
- S - string
- U - unicode string
- V - fixed chunk of memory for other type (void)

Checking the Data Type of an Array

- The NumPy array object has a property called dtype that returns the data type of the array.
- Example:
- Get the data type of an array object:
-
```
import numpy as np

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

print(arr.dtype)
```
- Try checking data type of array of string values.
- The key difference between type and dtype is: type is a general Python concept used to determine the type of an object (e.g., a variable), while dtype is specific to NumPy arrays and describes the type of data stored within those arrays.

Creating Arrays With a Defined Data Type

- We use the array() function to create arrays, this function can take an optional argument: dtype that allows us to define the expected data type of the array elements.
- For example, Creating an array with data type string:
-
```
import numpy as np
arr = np.array([1, 2, 3, 4], dtype='S')
print(arr)
print(arr.dtype)
```

- For i, u, f, S and U we can define size as well.

For example:

```
arr = np.array([1, 2, 3, 4], dtype='i4')
```

Note : What if a Value Can Not Be Converted?

- If a type is given in which elements can't be casted then NumPy will raise a ValueError.
- Try converting a non integer string like 'a' to an integer.

-
```
import numpy as np

arr = np.array(['a', '2', '3'], dtype='i')
```

Converting Data Type on Existing Arrays

- The best way to change the data type of an existing array, is to make a copy of the array with the astype() method.

- The astype() function creates a copy of the array, and allows you to specify the data type as a parameter.

- The data type can be specified using a string, like 'f' for float, 'i' for integer etc. or you can use the data type directly like float for float and int for integer.

- Example 1: Change data type from float to integer by using 'i' as parameter value:

```
import numpy as np
arr = np.array([1.1, 2.1, 3.1])
newarr = arr.astype('i')
print(newarr)
print(newarr.dtype)
```

- Example 2: Change data type from float to integer by using int as parameter value:

```
import numpy as np
arr = np.array([1.1, 2.1, 3.1])
newarr = arr.astype(int)
print(newarr)
print(newarr.dtype)
```

Array operations and broadcasting

- Elementwise operation

```
x = np.ones(4) #Creating an array
y=x+1
x-y
x = =y
x*y
x**y
x/y
np.array_equal(x, y)
```

Broadcasting

ndarrays with different sizes cannot be directly used in arithmetic operations:

```
a = np.ones((2, 2))
b = np.ones((3, 3))
a + b ERROR
```

- Broadcasting describes how NumPy treats arrays with different shapes during arithmetic operations. The idea is to wrangle data so that operations can occur element-wise.

- Let's see an example. Say I sell pies on my weekends. I sell 3 types of pies at different prices, and I sold the following number of each pie last weekend. I want to know how much money I made per pie type per day.

Pie Cost	
	Cost ($)
Apple	20
Blueberry	15
Pumpkin	25

Number of Pies Sold			
	Friday	Saturday	Sunday
Apple	2	3	1
Blueberry	6	3	3
Pumpkin	5	3	5

EXAMPLE

```
cost = np.array([20, 15, 25]) #1D array
print("Pie cost:")
print(cost)
sales = np.array([[2, 3, 1], [6, 3, 3], [5, 3, 5]]) #2D array
print("\nPie sales (#):")
print(sales)
```

o/p:

```
cost shape: (3, 1)
sales shape: (3, 3)
```

How can we multiply these two arrays together? We could use a loop:

```
total = np.zeros((3, 3))  # initialize an array of 0's
for col in range(sales.shape[1]):
    total[:, col] = sales[:, col] * cost
total
```

Or we could make them the same size, and multiply corresponding elements “elementwise”:

```
cost = np.repeat(cost, 3).reshape((3, 3))
print(cost)

cost * sales
```

```
cost = np.array([20, 15, 25]).reshape(3, 1)
print(f" cost shape: {cost.shape}")

sales = np.array([[2, 3, 1], [6, 3, 3], [5, 3, 5]])
print(f"sales shape: {sales.shape}")

sales * cost
```

Why should you care about broadcasting? Well, it’s cleaner and faster than looping and it also affects the array shapes resulting from arithmetic operations.

Of course, not all arrays are compatible! NumPy compares arrays element-wise. It starts with the trailing dimensions, and works its way forward. Dimensions are compatible if:

- **they are equal**, or
- **one of them is 1**.

Use the code below to test out array compatibitlity:

```
a = np.ones((3, 2))
b = np.ones((3, 2, 1))
print(f"The shape of a is: {a.shape}")
print(f"The shape of b is: {b.shape}")
print("")
try:
   print(f"The shape of a + b is: {(a + b).shape}")
except:
   print(f"ERROR: arrays are NOT broadcast compatible!")
```

Reshaping Arrays

- We can reshape an array to another dimension using the reshape() function.
- There are three reshaping methods:
  - np.reshape()
  - np.newaxis
  - np.ravel()/ np.flatten()
- Example reshape() function:
  - x = np.full((4, 3), 3.14)
    x.reshape(6, 2)
  - x.reshape(2, -1) # -1 indicates that you will leave it to reshape() function to create the correct number of columns

np.newaxis

- Sometimes you'll want to add dimensions to an array for broadcasting purposes.
- We can do that with np.newaxis (note that None is an alias for np.newaxis). We can add a dimension to an array to make the arrays compatible.
- For example:

```
a = np.ones(3)
print_array(a)
b = np.ones((3, 2))
print_array(b)
```

- If you want to add these two arrays you won't be able to because their dimensions are not compatible:
  - a + b
  - print_array(a[:, np.newaxis]) # same as a[:, None]
  - a[:, np.newaxis] + b
- Finally, sometimes you'll want to "flatten" arrays to a single dimension using .ravel() or .flatten()
- Create a 2d array, use flatten() / ravel() function to change its dimension.

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 owns the data and any changes made to the copy will not affect original array, and any changes made to the original array will not affect the copy.
- The view does not own the data and any changes made to the view will affect the original array, and any changes made to the original array will affect the view.
- Try, making a copy and view, change the original array, and display both arrays.

use .copy() function to make a copy of the original array.

use .view() function to make a view of the original array.

```
import numpy as np

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

print(arr)
print(x)
```

```
arr2 = np.array([1, 2, 3, 4, 5])
y = arr2.view()
arr2[0] = 42

print(arr2)
print(y)
```

Make Changes in the VIEW:

- Example
- Make a view, change the view, and display both arrays:

```
import numpy as np

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

print(arr)
print(x)
```

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

Copying by reference

- a2 = a1, changing the shape of original array will affect the copied array.
- a2 = a1.view(), creates copy of a1 by reference; but changes in dimension in a1 will not affect a2 {shallow copy}
- a2 = a1.copy()

Check if Array Owns its Data

Copies owns the data, and views does not own the data, but how can we check this?

- Every NumPy array has the attribute *base* that returns *None* if the array owns the data.
- Otherwise, the base attribute refers to the original object.
- For example:
  - `arr = np.array([1, 2, 3, 4, 5])`

```
x = arr.copy()
y = arr.view()

print(x.base)
print(y.base)
```

```
# Printing the value of the base attribute to check if an array owns it's data or not
```

The copy returns None.

The view returns the original array.