NumPy Basics - Creating NumPy Arrays and Array Attributes

WE'LL COVER THE FOLLOWING

1. Creating Arrays

There are two ways to create arrays in NumPy:

a. Arrays From Lists

The first step when working with packages is to define the right “imports”. We can import NumPy like so:

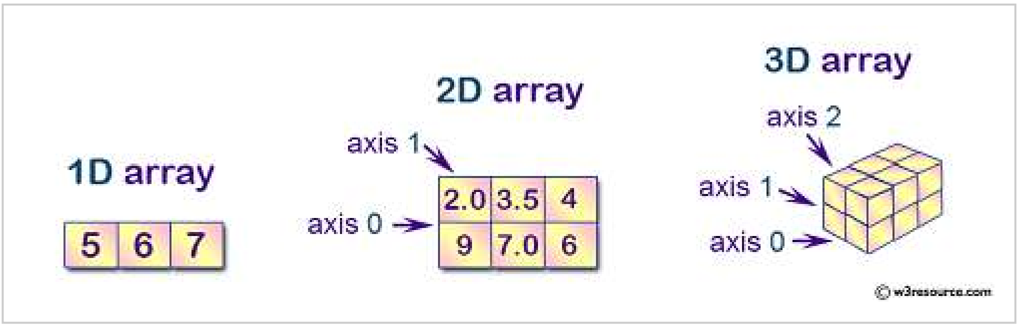
```
import numpy as np
```

Notice that *np* is the standard shorthand for NumPy.

There are multiple ways to create arrays. Let’s start by creating an array from a list using the `np.array` function.

We can create a one-dimensional (1D) array from a list by passing the list as input parameters to the *np.array* function:

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



What does 1D, 2D,3D array mean? Image Credits: www.3resource.com

Run the code in the widget below and **inspect** the output values. In particular, observe the `type` of the created array from the result of the print statement.

```
# Import the numpy package
import numpy as np

# Create a 1d integer array from a list
arr1 = np.array([1, 2, 3, 4])

# Print the array and its type
print(arr1)
print(type(arr1))
```

If we want to explicitly set the data type of the resulting array, we can use the `dtype` keyword. Some of the most commonly used numpy dtypes are: *'float'*, *'int'*, *'bool'*, *'str'*, and *'object'*. Say we want to create an array of floats, we can define it like so:

```
# Create a 1d float array
arr2 = np.array([1, 2, 3, 4], dtype='float32')

# Print the array and its type
print(type(arr2))
print(arr2)
```

In the examples above, we have seen one-dimensional arrays. We can also define two and three-dimensional arrays.

```
# Create a 2d array from a list of lists
lists = [[0,1,2], [3,4,5], [6,7,8]]
arr2d = np.array(lists)

print(arr2d)
```

A key difference between an array and a list is that arrays allow you to perform *vectorized* operations while a list is not designed to handle vector operations. A vector operation means a function gets applied to every item in the array.

Say we have a list and we want to multiply each item by 2. We cannot do element-wise operations by simply saying "*mylist * 2*". However, we can do so on a NumPy array. Let's see some code-examples:

```
arr1 = np.array([1, 2, 3, 4])
print(arr1)

# Vector (element-wise) operations
print(arr1 * 2)
print(arr1 + 2)
print(arr1 * arr1)
```

Some other key differences between Python built-in lists and NumPy arrays are:

- Array size cannot be changed after creation, you will have to create a new array or overwrite the existing one to change size.
- Unlike lists, all items in the array must be of the same *dtype*.
- An equivalent NumPy array occupies much less space than a Python list of lists.

b. Arrays From Scratch

Now instead of using lists as a starting point, let's learn to create arrays from scratch. For large arrays, it is more efficient to create arrays using routines already built into NumPy. Here are several examples:

already built into NumPy. Here are several examples.

```
# Create an integer array of length 100 filled with zeros
np.zeros(100, dtype=int)

# Create a 3x3 floating-point array filled with 1s
np.ones((3, 3), dtype=float)

# Create an array filled with a linear sequence
# Starting at 0, ending at 20, stepping by 3
# (this is similar to the built-in range() function)
np.arange(0, 20, 3)

# Create an array of hundred values evenly spaced between 0 and 1
np.linspace(0, 1, 100)

# Create a 3x3 array of uniformly distributed random values between 0 and 1
np.random.random((3, 3))

# Create a 3x3 array of random integers in the interval [0, 10)
np.random.randint(0, 10, (3, 3))

# Create a 3x3 array of normally distributed random values
# with mean 0 and standard deviation 1
np.random.normal(0, 1, (3, 3))

np.random.randint(10, size=6)  # One-dimensional array of random integers
np.random.randint(10, size=(3, 3))  # Two-dimensional array of random integers
np.random.randint(10, size=(3, 3, 3))  # Three-dimensional array of random integers
```

2. Array Attributes

Each array has the following attributes:

- `ndim`: the number of dimensions
- `shape`: the size of each dimension
- `size`: the total size of the array
- `dtype`: the data type of the array
- `itemsize`: the size (in bytes) of each array element
- `nbytes`: the total size (in bytes) of the array

Run the code below and observe the output.

```
import numpy as np

# Create a 3x3 array of random integers in the interval [0, 10)
x = np.random.randint(0, 10, (3, 3))
```

```
x = np.random.randint(0, 10, (3, 5))

print("ndim: ", x.ndim)
print("shape:", x.shape)
print("x size: ", x.size)
print("dtype:", x.dtype)
print("itemsize:", x.itemsize, "bytes")
print("nbytes:", x.nbytes, "bytes")
```