How to Create NumPy Arrays

There are two common approaches for creating NumPy arrays:

a. Using the numpy.array function
b. Using array creation routines

Let us learn both approaches one by one.

a. numpy.array function:

The basic ndarray is created using array function in NumPy as follows –

```
numpy.array
```

The **numpy.array** function is a powerful constructor in NumPy, allowing the creation of arrays with various parameters. Below is an overview of its parameters:

```
numpy.array(object, dtype = None, copy = True, order = None,
subok = False, ndmin = 0)
```

S. No.	Parameter	Description
1.	Object	An array is returned by any object that exposes the array interface method, or alternatively, any (nested) sequence.
2.	Dtype	Desired data type of the array, optional.
3.	Copy	Optional specification of the desired data type for the array.
4.	Order	Specify either C (row major), F (column major), or A (any) to determine the array layout.
5.	Subok	By default, the array that is returned is coerced into being a base class array. If set to true, sub-classes will be allowed to pass through without coercion.
6.	ndmin	Defines the minimum dimensions of the resulting array.

Let us make Python programs to understand the ndarray better.

1. **Creating 1-D Array:** Python program to make a one-dimensional array.

```
import numpy as np
a = np.array([4,6,8])
print(a)

Output
[4 6 8]
```

This code creates a NumPy array named a, with the elements 4, 6, and 8, and then prints the array to the console.

2. **Creating 2-D Array**: Python program to create two-dimensional array.

```
# multi- dimension arrays
import numpy as np
a = np.array([[3, 4], [9, 6]])
print(a)

Output
[[3 4]
 [9 6]]
```

This code creates a 2-dimensional NumPy array named a with elements arranged in two rows and two columns and then prints the array to the console.

3. **Creating Array with minimum dimension**: You can create an array with a minimum dimension using the ndmin parameter when creating the array. This parameter specifies the minimum number of dimensions that the resulting array should have.

```
# minimum dimensions
import numpy as np
a = np.array([1, 2, 3,4,5], ndmin = 2)
print(a)

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

This code uses array function to create a 2-dimensional array with elements 1, 2, 3, 4, and 5. The ndmin=2 parameter ensures that the array

has a minimum of two dimensions, even though the input is a 1-dimensional list. The output is a 2D array with a single row and five columns.

4. **Creating Array of Complex Numbers:** You can create an array of complex numbers by specifying the dtype parameter as complex when creating the array.

```
# dtype parameter
import numpy as np
a = np.array([1, 2, 3], dtype = complex)
print(a)

Output
[1.+0.j 2.+0.j 3.+0.j]
```

The dtype parameter specifies the data type of array elements. Here, dtype=complex is used to create an array of complex numbers from the integer values 1, 2, and 3.

A new ndarray object can be generated using one of the **array creation routines** or through the low-level ndarray constructor. Let us learn about these array creation routines.

b. NumPy - Array Creation Routines

NumPy Array Creation Routines refer to a set of functions provided by the NumPy library that allow users to create arrays with specific characteristics, shapes, and initial values.

The routines are the following:

1. numpy.empty
2. numpy.zeros
3. numpy.ones
4. numpy.asarray
5. numpy.arange
6. numpy.linspace
7. numpy.logspace

Let us discuss these routines one by one.

1. **numpy.empty**: It is a routine that creates an uninitialized array with a specified shape and data type. The constructor parameters include shape, dtype (default is float), and order (default is 'C').

   Syntax:

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

   The constructor takes the following parameters.

Sr.No.	Parameter & Description
1	Shape Shape of an empty array in int or tuple of int
2	Dtype Desired output data type. Optional
3	Order 'C' for C-style row-major array, 'F' for FORTRAN style column-major array

   Example Code:

```
import numpy as np
x = np.empty([3,2], dtype = int)
print(x)

Output
[[0 0]
 [0 0]
 [0 0]]
```

   The numpy.empty function creates an array with uninitialized values. This code is creating a 3x2 array with integer dtype.

2. **numpy.zeros**: numpy.zeros produces a new array with the given size, where all elements are set to zero.

   Syntax:

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

The constructor takes the following parameters:

Sr.No.	Parameter & Description
1	Shape - Shape of an empty array in int or sequence of int
2	Dtype - Desired output data type. (Optional)
3	Order - 'C' for C-style row-major array, 'F' for FORTRAN style column-major array

Example Code 1:

```
# array of three zeros. The default dtype is float
import numpy as np
x = np.zeros(3)
print(x)

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

This code creates a NumPy array named **x** with three elements, all initialized to zero, and the default data type is float.

Example Code 2:

```
# custom type
import numpy as np
arr = np.zeros((2,2), dtype = [('a', 'i4'), ('b', 'i4')])
print(arr)

Output
[[(0, 0) (0, 0)]
 [(0, 0) (0, 0)]]
```

The code creates a 2x2 NumPy array with a custom data type, where each element is a tuple with two integer fields ('a' and 'b'). The array is initialized with zeros, and the output shows the array with the specified data type.

3. **numpy.ones**: It makes a new array with the specified size and data type, where all elements are set to one.

Syntax:

```
numpy.ones(shape, dtype = None, order = 'C')
```

The constructor takes the following parameters.

Sr.No.	Parameter & Description
1	**Shape** Shape of an empty array in int or tuple of int
2	**Dtype** Desired output data type. Optional
3	**Order** 'C' for C-style row-major array, 'F' for FORTRAN style column-major array

Example Code 1:

```
# Create an array of five ones. The Default dtype is float
import numpy as np
arr = np.ones(5)
print(arr)

Output
[1. 1. 1. 1. 1.]
```

The code uses NumPy to create an array of five floating-point elements, all set to 1. The **np.ones(5)** function is used, and the resulting array is printed.

Example Code 2:

```
import numpy as np
arr = np.ones([2,2], dtype = int)
print(arr)

Output
[[1 1]
 [1 1]]
```

The code is creating a variable **arr**. It is a NumPy array with dimensions 2x2, and all the elements in the array are set to the value 1 using np.ones() function.

Let us learn to create NumPy arrays from existing data.

4. **numpy.asarray:** This function is used to convert input data, such as lists, tuples, or other array-like objects, into a NumPy array.

Syntax:

```
numpy.asarray(a, dtype = None, order = None)
```

The constructor accepts the following parameters:

S.No.	Parameter Name	Parameter Description
1.	a	Input data in various forms, including lists, lists of tuples, tuples, tuple of tuples, or tuple of lists.
2.	dtype	By default, the data type of the input data is assigned to the resulting ndarray.
3.	order	C (row major) or F (column major). The default is C.

Example Code 1:

```
# convert the list to ndarray
import numpy as np

x = [4,5,6]
arr = np.asarray(x)
print(arr)

Output
[4 5 6]
```

The Python list named **x** is converted into a NumPy array named **arr**. The **np.asarray()** function is employed for this conversion.

Example Code 2:

```
# Making ndarray from a tuple
import numpy as np

x = (1,2,3)
arr = np.asarray(x)
print(arr)

Output
[1 2 3]
```

The code is converting a tuple named x into a NumPy array named arr using the as array() function.

Now that you know how to create a ndarray from existing data. Let us learn how we can create an array from numerical ranges.

5. **numpy.arange (ndarray from Numerical Ranges):** This function generates an ndarray object with evenly spaced values within a specified range.

Syntax:

```
numpy.arange(start, stop, step, dtype)
```

The constructor accepts the following parameters.

- start: The beginning of the interval. If not provided, it defaults to 0.
- stop: The end of the interval (exclusive).
- step: The spacing between values, with a default of 1.
- dtype: The data type of the resulting ndarray. If unspecified, the data type of the input is utilized.

Example Code 1:

```
import numpy as np
arr = np.arange(3)
print(arr)

Output
[0 1 2]
```

The code creates an array (**arr**) with elements ranging from 0 to 2 using the **arange** function.

Example Code 2:

```
import numpy as np
# dtype set
arr= np.arange(3, dtype = float)
print(arr)

Output
[0. 1. 2.]
```

A NumPy array 'arr' is created with values 0.0, 1.0, 2.0, having a data type of float. It's generated using **arrange** function.

Example Code 3:

```
# start and stop parameters set
import numpy as np
arr = np.arange(30,40,2)
print (arr)

Output
[30 32 34 36 38]
```

NumPy array 'arr' is created with values from 30 to 40 (exclusive), incrementing by 2. It's printed using **arrange**.

6. **numpy.linspace:** This function bears similarity to the arange() function. Rather than specifying a step size, this function defines the number of evenly spaced values within the given interval.

Syntax:

```
numpy.linspace(start, stop, num, endpoint, retstep, dtype)
```

The constructor accepts the following parameters.

Sr.No.	Parameter	Description
1.	start	The initial value of the sequence.
2.	stop	The final value of the sequence, included in the sequence if the endpoint is set to true.
3.	num	The quantity of evenly spaced samples to generate. The default is 50
4.	endpoint	True by default, meaning the stop value is encompassed in the sequence. If false, it is excluded.
5.	retstep	If true, it provides samples and the step between consecutive numbers.
6.	dtype	The data type of the output ndarray.

Example Code:

```
import numpy as np
arr= np.linspace(20,40,5)
print(arr)

Output
[20. 25. 30. 35. 40.]
```

The code uses NumPy to create an array (arr) containing 5 evenly spaced values between 20 and 40 (inclusive). It then prints the ndarray.

7. **numpy.logspace:** The function yields an ndarray comprising numbers evenly distributed on a logarithmic scale. The start and stop endpoints on the scale are indices of the base, typically set to 10.

Syntax:

```
numpy.logspace(start, stop, num, endpoint, base, dtype)
```

The output of the logspace function is influenced by the following parameters.

S.No.	Parameter	Description
1.	start	The initiation point of the sequence is denoted as "basestart."
2.	stop	The concluding value of the sequence is referred to as "basestop."
3.	num	The quantity of values within the specified range. The default value is set to 50.
4.	endpoint	If set to true, the stop value becomes the ultimate value in the range.
5.	base	The fundamental value of the logarithmic space; by default, it is set to 10.
6.	dtype	The data type of the resulting array. If not explicitly provided, it is determined by other input parameters.

Example Code:

```
# set base of log space to 2
import numpy as np
arr = np.logspace(1,10,num = 10, base = 2)
print(arr)

Output
[   2.    4.    8.   16.   32.   64.  128.  256.  512.
1024.]
```

This code creates an array (arr) with ten logarithmically spaced values between 2 and 1024. The logarithmic scale has a base of 2, achieved by setting the base parameter in the logspace function.

You now know how to create NumPy arrays. Let us know characteristics of NumPy arrays, including details such as data type, size, memory usage, etc.

ndArray Attributes

Attributes of ndarrays are properties or characteristics associated with these arrays that provide information about their structure and content. These attributes offer insights into the array's dimensions, data type, size, and memory usage etc.

Following are the main attributes of ndArrays:

1. ndarray.shape
2. ndarray.ndim
3. numpy.itemsize
4. numpy.flags

1. **ndarray.shape**: This array property provides a tuple containing the dimensions of the array and can also be utilized for resizing the array.

Example Code 1:

```
import numpy as np
a = np.array([[4,5,6],[7,8,9]])
print(a.shape)

Output
(2,3)
```

The code creates a 2x3 array named a, and prints its shape, which is (2, 3), indicating 2 rows and 3 columns.

Example Code 2:

```
# this resizes the ndarray
import numpy as np

a = np.array([[4,5,6],[7,8,9]])
a.shape = (3,2)
print(a)

Output
[[4 5]
 [6 7]
 [8 9]]
```

This code snippet creates a 2x3 array **a**, then resizes it to a 3x2 array using the **shape** property. Finally, it prints the resized array.

NumPy offers a reshape function for resizing arrays as well.

Example Code:

```
import numpy as np
a = np.array([[4,5,6],[7,8,9]])
b = a.reshape(3,2)
print(b)

Output
[[4 5]
 [6 7]
[8 9]]
```

The code creates a NumPy array **a** with shape (2,3), then reshapes it to a new array **b** with shape (3,2), and finally prints the reshaped array b.

2. **ndarray.ndim**: This array attributes provides the number of dimensions (axes) in the array.

Example Code:

```
import numpy as np

# Create a 1-dimensional array
arr_1d = np.array([1, 2, 3])
print(arr_1d.ndim)

# Create a 3-dimensional array
arr_3d = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
print(arr_3d.ndim)

Output
1
3
```

In the code **arr_1d** is a 1-dimensional array, and **arr_3d** is a 3-dimensional array. The **ndim** attribute is used to query the number of dimensions of each array.

3. **numpy.itemsize:** This array attributes provides the size of each element in the array, measured in bytes**.**

```
# dtype of array is float32 (4 bytes)
import numpy as np
x = np.array([1,2,3,4,5], dtype = np.float32)
print(x.itemsize)

Output
4
```

Array x contains 32-bit floating-point numbers, so each element occupies 4 bytes.

4. **numpy.flags:** This function returns the current values of the attributes of the ndarray object.

Sr.No.	Attribute & Description
1	C_CONTIGUOUS (C) - The data is organized in a single, C-style contiguous segment.
2	F_CONTIGUOUS (F) - The data is organized in a single, Fortran-style contiguous segment.
3	OWNDATA (O) - The array either owns the memory it uses or borrows it from another object.
4	WRITEABLE (W) - The data area is writable; setting to False locks the data, making it read-only.
5	ALIGNED (A) - Both data and elements are appropriately aligned for the hardware.
6	UPDATEIFCOPY (U) - This array serves as a copy of another array; deallocation updates the base array with its contents.

Example Code:

```
import numpy as np
x = np.array([3,4,5,6,7])
print(x.flags)

Output
C_CONTIGUOUS : True
F_CONTIGUOUS : True
OWNDATA : True
WRITEABLE : True
ALIGNED : True
WRITEBACKIFCOPY : False
```

The code creates a NumPy array 'x' with elements [3,4,5,6,7] and prints its flags indicating memory layout and properties.