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 **arange**.

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.