NumPy Array Creation and Manipulation

This document provides a comprehensive guide to essential NumPy functions for array creation, manipulation, and data analysis.

by dorira Mohamed

np.array() - Creating Arrays from Existing Data

The **np.array()** function is fundamental for converting various data structures into NumPy arrays.

np.array(object, dtype=None, copy=True, order='K', subok=False, ndmin=0, like=None)

object	Data to be converted to array	---	[1, 2, 3]	Missing → Error
dtype	Force data type	None	dtype=int	Incompatible → TypeError
copy	Copy data or not	True	copy=False	Shared memory effects
order	Memory layout ('C', 'F', 'K')	'K'	order='F'	N/A
subok	Allow subclass return	False	subok=True	Rare usage
ndmin	Force minimum number of dimensions	0	ndmin=3	N/A
like	Copy structure of another array (advanced)	None	like=another_array	Rare usage

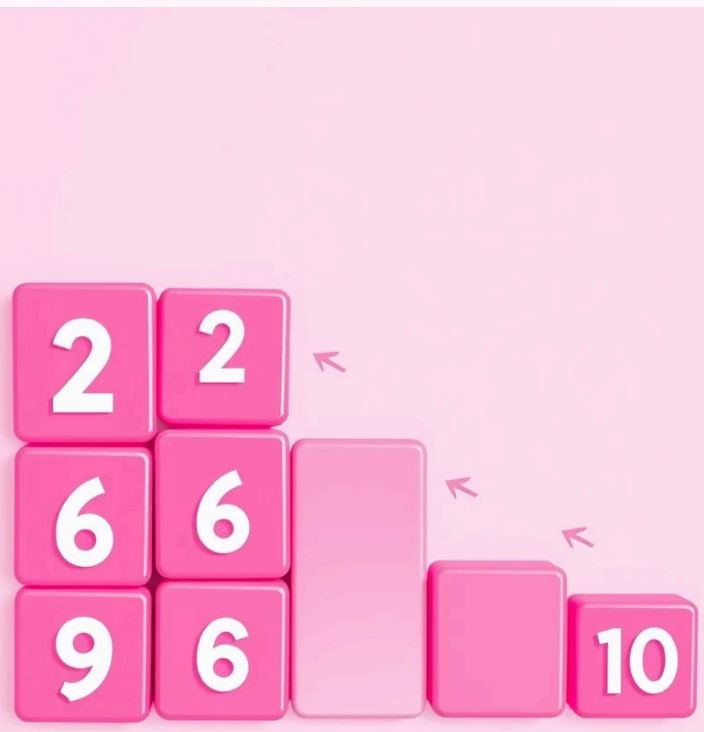
np.arange() - Generating Sequences

The **np.arange()** function is used to generate arrays with regularly spaced values within a given interval.

```
np.arange(start, stop, step, dtype=None, like=None)
```

start	Number (int/float)	The number from which the array starts	❌ (Default = 0)	Determines where counting begins
stop	Number (int/float)	The number at which generation stops (but not included)	✅ Mandatory	Determines where counting ends
step	Number (int/float)	The difference between each number in the array	❌ (Default = 1)	Controls counting speed or direction
dtype	Data type	The data type of returned elements (like int32, float64...)	❌	If you want to control element type inside the array
like	ndarray	Used to make the array similar to an existing array (advanced usage)	❌	Rarely used at beginner levels

np.arange(5)	[0 1 2 3 4]	Simple sequence
np.arange(2, 8)	[2 3 4 5 6 7]	Sequence starting from 2
np.arange(0, 10, 2)	[0 2 4 6 8]	Sequence with step 2
np.arange(5, 0, -1)	[5 4 3 2 1]	Descending sequence
np.arange(1, 4, dtype=float)	[1. 2. 3.]	Converting data type to float



np.zeros(), np.ones(), and np.full() - Initializing Arrays

These functions are used to create new arrays filled with specific values, useful for initializing data structures.

np.zeros(shape, dtype=float, order='C', like=None)

shape	int or tuple	Size or dimensions of required array (e.g., 3×3 or 2×4 or 1×5)	✅ Mandatory	Determines number of rows and columns
dtype	Data type	Type of elements inside array (like: int, float, bool, etc.)	❌ Optional	Controls value type inside each cell
order	str	Data arrangement method in memory: 'C' = rows, 'F' = columns	❌ Optional	Affects performance (especially in operations)
like	array-like	If you want to build array with same properties as another array (advanced usage)	❌ Optional	Rarely used at the beginning

np.ones(shape, dtype=float, order='C', like=None)

shape	int or tuple	Size or dimensions of required array (e.g., 3×3 or 2×4 or 1×5)	✅ Mandatory	Determines number of rows and columns
dtype	Data type	Type of elements inside array (like: int, float, bool, etc.)	❌ Optional	Controls value type inside each cell
order	str	Data arrangement method in memory: 'C' = rows, 'F' = columns	❌ Optional	Affects performance (especially in operations)
like	array-like	If you want to build array with same properties as another array (advanced usage)	❌ Optional	Rarely used at the beginning

np.full(shape, fill_value, dtype=None, order='C', like=None)

shape	int or tuple	Required array dimensions (e.g., 3×3 or 2×4 or 1×5)	✅ Mandatory	Determines array shape
fill_value	Any data type	The value with which all array elements are filled	✅ Mandatory	Can be number, text, True/False
dtype	Data type	Data type inside array (like: int, float, str)	❌ Optional	Automatically recognized if not specified
order	'C' or 'F'	Storage order in memory (rows = 'C' or columns = 'F')	❌ Optional	Only affects performance
like	array-like	Uses properties of existing Array (advanced usage)	❌ Optional	Usually not necessary for beginners

np.zeros()

All values = 0

When starting with empty numeric array

np.ones()

All values = 1

For model testing/fixed experiment

np.full()

All values = what you choose

If you want specific value to repeat

np.random.rand() and np.random.randn() - Generating Random Numbers

NumPy provides functions to generate arrays with random numbers from different distributions.

np.random.rand(d0, d1, ..., dn)

d0, d1...	int	Dimensions you want to create array with	✅ Yes	Can pass more than one dimension

numpy.random.randn(d0, d1, ..., dn)

d0, d1, ..., dn	int	One or more integers specifying the dimensions of the output array.

np.random.rand()	Numbers between 0.0 and 1.0 (uniform distribution)	For testing or generating random data
np.random.randint()	Random integers within specific range	When needing random integers
np.random.randn()	Numbers with normal distribution (Gaussian distribution)	When needing to simulate natural data

Distribution	Uniform [0, 1)	Normal (Gaussian), mean=0, std=1
Use Case	Random values between 0 and 1	Simulate real-world "natural" randomness

📌 When to Use:

- You want random positive values between 0 and 1 → **np.random.rand()**
- You want random values that include negatives and positives around 0 (like in real data) → np.random.randn()

numpy.reshape() and np.resize() - Changing Array Shapes

These functions allow you to modify the dimensions of an array, which is crucial for data preparation in many applications.

numpy.reshape(a, newshape, order='C')

a	ndarray	The array to be reshaped. (When using numpy.reshape(a, ...))
newshape	int or tuple of ints	The new shape you want. One dimension can be -1, meaning it will be automatically inferred.
order	str, optional	'C' (row-major), 'F' (column-major), or 'A' (any) --- affects how data is read from memory. Default is 'C'.

ndarray	A new array with the same data but new shape.

'C'	Row-major order (default) --- like C language
'F'	Column-major order --- like Fortran
'A'	Any order (let NumPy decide based on memory)

Preparing data for ML models	You often need to reshape arrays to 2D: (samples, features)
Visualizing images	Images often need to be reshaped to (height, width, channels)
Flattening or restructuring data	Use reshape to switch between 1D/2D/3D easily

Trying to reshape to invalid shape	ValueError: cannot reshape array of size X into shape (...)
Using incompatible shapes or forgetting -1 properly	Unexpected output or wrong shape

Modifies shape	✅ Yes
Copies data?	❌ Usually No (creates a view)
Keeps values	✅ Yes
Keeps original shape	❌ No

np.resize(a, new_shape)

a	array_like	Original array you want to change its shape
new_shape	int or tuple of int	Required new shape (new number of elements, or multidimensional array shape)

If dimensions are incompatible?	❌ Gives error	✅ Automatically repeats elements
Modifies original array?	❌ No (returns new copy)	✅ Possible (if using array.resize() object method)
Appropriate usage	Change shape without changing data	To fill array with new shape even if elements are fewer

🔙 Summary of np.resize() uses:

Need to increase number of elements and repeat	✅ Very suitable
Need to only decrease elements	✅ Fine
Need to change shape without changing data	❌ Use reshape() better
Need high performance and consistency?	❌ reshape() better

📌 Real difference summary:

Fill grid or model with repeated values	✅ Yes	❌ No
Change data shape without changing count	❌ No	✅ Yes
Prepare data for ML model with fixed length	✅ Suitable (for testing only)	❌ Not sufficient
Need to preserve same data?	❌ No	✅ Yes

np.copy() and Slicing - Data Duplication and Extraction

Understanding how to copy arrays and extract subsets is crucial for data integrity and analysis.

np.copy(a, order='K', subok=True)

a	Original array you want to copy	✅ Yes	-
order	Data storage form in memory ('C', 'F', 'A', 'K')	❌ No	'K'
subok	Whether to allow copying subclass (special types of ndarray) or not	❌ No	True

Copy simple Numpy array	✅ Very good	✅
Copy complex Python object (nested)	❌ Not guaranteed	✅ Better
Performance	Faster in Numpy only	Slower but comprehensive

Simple data of Numpy type	np.copy()	Fast and light enough
Nested or complex structure data	copy.deepcopy()	To avoid editing effect on original

array[start:stop:step]

slicing

📌 Practical uses:

- Extract data from specific part in DataFrame
- Do Training/Test split
- Prepare data for ML model
- Edit or analyze Subsets of data

a[0] → single element only	a[0:3] → group of elements
Returns number/element directly	Returns new array
Doesn't accept step	Accepts start:stop:step

❗ Possible errors you might encounter:

IndexError	If you used index outside array bounds
a[1:10] and there aren't 10 elements	No error, but will return existing only
Forgot to write :	Will return single element instead of group

np.random.permutation(), np.random.shuffle(), np.random.randint(), and np.argwhere()

These functions offer powerful capabilities for randomizing data, generating integers, and locating specific elements within arrays.

np.random.permutation(x)

np.random.permutation()	✅ Yes	❌ No
np.random.shuffle()	❌ No (returns None)	✅ Yes

🔍 Parameters:

x	int or array-like	Either a number (defines random ordering range), or array to be shuffled

If you want to keep original copy, use permutation() If you need to modify data itself in place, use shuffle()

Returns copy?	✅ Yes	❌ No
Changes original?	❌ No	✅ Yes
Suitable if?	Want new copy and preserve original	Want to change original data itself
Returns what?	Shuffled copy	None

np.random.randint(low, high=None, size=None, dtype=int)

low	Lower bound for values (inclusive)	✅ Yes	Lowest possible value appearing in results
high	Upper bound (exclusive)	❌ No	If not written → generation from 0 to low
size	Size of resulting array (number of numbers or its shape)	❌ No	If not written → returns single value only
dtype	Data type (default: int)	❌ No	You can choose np.int32 or np.int64 etc.

ValueError: low >= high	If you wrote low greater than or equal to high	Make sure low < high
TypeError: 'float' object...	Used float values instead of int	randint works with integers only

✅ Practical uses:

- Generate random training data
- Choose numbers for algorithm testing
- Create dummy data

✅ Why is randint() important here?

- Because it:
- Gives you control over random range of values
- Helps you simulate realistic data for model testing
- Very fast and suitable for data simulation or bootstrapping

np.argwhere(condition)

📥 Input:

- condition: boolean condition, like:
  - a > 5
  - a == 0
  - a % 2 == 1

📌 It's a condition applied to Numpy array.

📤 Output:

- 2D array containing coordinates [rows, cols] for each element that met the condition.

Search for locations of specific element	np.argwhere(a == 0)	Returns locations containing 0
Extract first location of specific element	np.argwhere(a == 20)[0]	Gives you first location
Know locations of values matching condition	np.argwhere(a % 2 == 0)	Locations of even numbers

✅ How this benefits you as data analyst:

- np.argwhere() allows you to precisely locate positions, which is useful when you want to:
  - Return original value
  - Replace it with average or expected value
  - Mark it for model to ignore