

 Search the docs ...

Array objects

Array API Standard

Compatibility

Constants

Universal functions ( **ufunc** )

**Routines** ^

Array creation routines

**Array manipulation routines** ^

[numpy.copyto](#)

[numpy.shape](#)

[numpy.reshape](#)

[numpy.ravel](#)

[numpy.ndarray.flat](#)

[numpy.ndarray.flatten](#)

[numpy.moveaxis](#)

[numpy.rollaxis](#)

[numpy.swapaxes](#)

[numpy.ndarray.T](#)

[numpy.transpose](#)

[numpy.atleast\\_1d](#)

[numpy.atleast\\_2d](#)

[numpy.atleast\\_3d](#)

[numpy.broadcast](#)

[numpy.broadcast\\_to](#)

[numpy.broadcast\\_arrays](#)

[numpy.expand\\_dims](#)

[numpy.squeeze](#)

[numpy.asarray](#)

[numpy.asanyarray](#)

[numpy.asmatrix](#)

[numpy.asfarray](#)

[numpy.asfortranarray](#)

[numpy.ascontiguousarray](#)

[numpy.asarray\\_chkfinite](#)

[numpy.require](#)

**[numpy.concatenate](#)**

[numpy.stack](#)

[numpy.block](#)

[numpy.vstack](#)

[numpy.hstack](#)

[numpy.dstack](#)

[numpy.column\\_stack](#)

[numpy.row\\_stack](#)

[numpy.split](#)

# numpy.concatenate

**numpy.concatenate**((a1, a2, ...), axis=0, out=None, dtype=None, casting="same\_kind")

Join a sequence of arrays along an existing axis.

**Parameters:** a1, a2, ... : *sequence of array\_like*

The arrays must have the same shape, except in the dimension corresponding to *axis* (the first, by default).

**axis** : *int, optional*


The axis along which the arrays will be joined. If axis is None, arrays are flattened before use. Default is 0.

**out** : *ndarray, optional*

If provided, the destination to place the result. The shape must be correct, matching that of what concatenate would have returned if no out argument were specified.


**dtype** : *str or dtype*

If provided, the destination array will have this dtype. Cannot be provided together with *out*.

 **New in version 1.20.0.**

**casting** : {'no', 'equiv', 'safe', 'same\_kind', 'unsafe'}, *optional*

Controls what kind of data casting may occur. Defaults to 'same\_kind'.

 **New in version 1.20.0.**

**Returns:** res : *ndarray*

The concatenated array.

### See also

#### [ma.concatenate](#)

Concatenate function that preserves input masks.

#### [array\\_split](#)

Split an array into multiple sub-arrays of equal or near-equal size.

#### [split](#)

Split array into a list of multiple sub-arrays of equal size.

#### [hsplit](#)

Split array into multiple sub-arrays horizontally (column wise).

#### [vsplit](#)

Split array into multiple sub-arrays vertically (row wise).

#### [dsplit](#)

Split array into multiple sub-arrays along the 3rd axis (depth).

#### [stack](#)

Stack a sequence of arrays along a new axis.

#### [block](#)

Assemble arrays from blocks.

#### [hstack](#)

Stack arrays in sequence horizontally (column wise).

#### [vstack](#)

Stack arrays in sequence vertically (row wise).

#### [dstack](#)

Stack arrays in sequence depth wise (along third dimension).

#### [column\\_stack](#)

Stack 1-D arrays as columns into a 2-D array.

### Notes

When one or more of the arrays to be concatenated is a `MaskedArray`, this function will return a `MaskedArray` object instead of an `ndarray`, but the input masks are *not* preserved. In cases where a `MaskedArray` is expected as input, use the `ma.concatenate` function from the masked array module instead.

### Examples

```
>>> a = np.array([[1, 2], [3, 4]])
>>> b = np.array([[5, 6]])
>>> np.concatenate((a, b), axis=0)
array([[1, 2],
       [3, 4],
       [5, 6]])
>>> np.concatenate((a, b.T), axis=1)
array([[1, 2, 5],
       [3, 4, 6]])
>>> np.concatenate((a, b), axis=None)
array([1, 2, 3, 4, 5, 6])
```

This function will not preserve masking of `MaskedArray` inputs.

```
>>> a = np.ma.arange(3)
>>> a[1] = np.ma.masked
>>> b = np.arange(2, 5)
>>> a
masked_array(data=[0, --, 2],
             mask=[False,  True,  False],
             fill_value=999999)
>>> b
array([2, 3, 4])
>>> np.concatenate([a, b])
masked_array(data=[0, 1, 2, 2, 3, 4],
             mask=False,
             fill_value=999999)
>>> np.ma.concatenate([a, b])
masked_array(data=[0, --, 2, 2, 3, 4],
             mask=[False,  True,  False,  False,  False,  False],
             fill_value=999999)
```

[< Previous](#)  
[numpy.require](#)

[Next >](#)  
[numpy.stack](#)