

Creating Arrays

Create simple array	<code>np.array([list])</code>
Array filled with zeros	<code>np.zeros(size)</code>
Array filled with ones	<code>np.ones(size)</code>
Array filled with a number	<code>np.full(size,number)</code>
Array from a range of numbers (last not included)	<code>np.arange(start,stop,step)</code>
Array from a range of numbers (last included)	<code>np.linspace(start,stop,count)</code>
Array filled with random numbers (float)	<code>np.random.random(size)</code>
Array filled with random numbers (int)	<code>np.random.randint(start,stop,size)</code>
Create identity Matrix	<code>np.eye(k)</code>
Create empty Matrix	<code>np.empty(k)</code>

Size can be the following :

1) **one-dimensional (Array)** : where size = k = (k)

eg : `np.zeros(2)` , creates an array of 2 elements

2) **two-dimensional (Matrix)** : where size = (a,b)

eg : `np.zeros((2,3))` , creates a matrix of 2 rows and one column

3) **n-dimensional** : where size = (a,b,c....etc) > 2

eg : `np.zeros((2,3,5))` , creates a 3 dimensional array

Reading Array From Text File

Inspecting the file (without reading it)	<code>!head file_source</code>	<code>!head data/array_float.csv</code>
Loading File Into Array	<code>np.loadtxt(source, delimiter, dtype)</code>	<code>B = np.loadtxt("data/array_int.csv", delimiter = ",", dtype=int)</code>
Loading Array Into File	<code>np.savetxt(source, array, fmt ,delimiter)</code>	<code>np.savetxt("data/my_array.csv", B, fmt="%3i", delimiter=",")</code>

Make sure that you are typing the right datatype of the text file or you will receive an error

The **fmt** keyword is used to specify the spacing in our text file where :

- `fmt = " %3i "` means 3 spaces between row elements (numbers are integers)

- `fmt = " %5.1 i "` means 5 spaces between row elements and only one number after floating point

Numpy Array Attributes

Returns number of dimensions	<code>array.ndim</code>
Returns size of each dimension	<code>array.shape</code>
Returns number of elements	<code>array.size</code>
Returns type of elements	<code>array.dtype</code>
Returns size of each element in bytes	<code>array.itemsize</code>
Returns size of all elements in bytes	<code>array.nbytes</code>

Array Indexing

Accessing array element	<code>array_name[index]</code>	<code>x1[5]</code>
Modify array element	<code>array_name[index] = new_value</code>	<code>x1[5] = 69</code>



Array Indexing (cont)

Accessing matrix element	matrix_name[row,column]	x2[2, -1]
Modify matrix element	matrix_name[row,column] = new_value	x2[2, -1] = -5
Accessing sub-arrays	array_name[start,stop,step=1]	x[1:5] x[1:5:3]
Accessing multi-dim arrays	array_name[row_range,column_range]	x2[:, -1, ::-1]
Creating subarrays as no-copy views	sub_array = original_array[slice]	x2_sub = x2[:, :2]
Creating subarrays as copy views	sub_array = original_array[slice].copy()	x2_sub_copy = x2[:, :2].copy()

We can access multiple matrix rows and columns when both row and columns are lists :

x2[[1,2],[1,1]] -> this means access # Second and Third row ([1,2]) , Second and Second column

Reshaping of Arrays

Reshape array	array.reshape((rows,columns))	x2 = x1.reshape((3, 3))
Reshape into row vector	array.reshape((1,columns))	x2 = x1.reshape((1, 3))
Reshape into row vector using np.newaxis	array[np.newaxis, :]	x2 = x1[np.newaxis, :]
Reshape into column vector	array.reshape((rows,1))	x2 = x1.reshape((3, 1))
Reshape into column vector using np.newaxis	array[:, np.newaxis]	x2 = x1[:, np.newaxis]

It is very important to note that number of rows multiplied by the number of columns must be equal to the number of array elements

Array Concatenation and Splitting

Concatenate Arrays vertically (default)	np.concatenate([x1,x2,x3....])	np.concatenate([x, y])
Concatenate Arrays horizontally	np.concatenate([x1,x2,x3....] , axis=1)	np.concatenate([x, y], axis=1)
Stack Arrays vertically	np.vstack([x1, x2,x3....])	np.vstack([x, y])
Stack Arrays horizontally	np.hstack([x1, x2,x3....])	np.hstack([x, y])
Splitting Arrays	np.split(array, [first_split_point, second_split_point , third_split_point....])	x1, x2, x3 = np.split(x, [3, 5])
Splitting Matrixes vertically	np.vsplit(matrix, [frist_row_index,second_row_index....])	upper, lower = np.vsplit(grid, [2])



By Taissir Boukrouba
(taissir2002)

Not published yet.
Last updated 8th November, 2023.
Page 2 of 8.

Sponsored by **Readable.com**
Measure your website readability!
<https://readable.com>

Array Concatenation and Splitting (cont)

Splitting Matrixes horizontally `np.hsplit(matrix, [frist_col_index,second_col_index....])` `left, right = np.hsplit(grid, [2])`

NOTE :

concatenate or stack vertically : must have same number of columns

concatenate or stack horizontally : must have same number of rows

NOTE 2 :

The number of split points when splitting an array depends on the number of arrays generated where :

Number of split points = number of arrays outputted - 1

`x1, x2, x3 = np.split(x, [3, 5])` -> split on the third element to x1 , split on the fifth element to x2 (from forth element) , give the rest to x3

Numpy Operations & UFuncs

Addition	Array + number	<code>x = [0 1 2 3]</code> <code>x+5</code> <i>output : [5 6 7 8]</i>
Substraction	Array - number	<code>x = [5 6 7 8]</code> <code>x-5</code> <i>output : [0 1 2 3]</i>
Multiplication	Array * number	<code>x*2</code> <i>output : [0 2 4 6]</i>
Division	Array / number	<code>x/2</code> <i>output : [0,0.5,1,1.5]</i>
Floor Division	Array // number	<code>x//2</code> <i>output : [0,0,1,1]</i>
Exponential	Array ** number	<code>x *2</code> <i>output : [0 1 4 9]*</i>
Modulus	Array % number	<code>x % 2</code> <i>output : [0 1 0 1]</i>
Dot Multiplication	<code>matrix1 @ matrix2</code>	check the notes
UFunc Addition	<code>np.add(array,number)</code>	<code>np.add(x,5)</code> <i>output : [5 6 7 8]</i>
UFunc Substraction	<code>np.subtract(array,number)</code>	<code>np.subtract(x,5)</code> <i>output : [0 1 2 3]</i>
UFunc Multiplication	<code>np.multiply(array,number)</code>	<code>np.multiply(x,2)</code> <i>output : [0 2 4 6]</i>
UFunc Dot	<code>np.dot(matrix1,matrix2)</code>	check the notes
UFunc Absolute Value	<code>np.abs(x)</code>	<code>x = np.array([-2, -1, 0, 1, 2])</code> <code>np.abs(x)</code> <i>output : [2,1,0,1,2]</i>
UFunc Sinus	<code>np.sin(x)</code>	self-explanatory



Numpy Operations & UFuncs (cont)

UFunc Cosinus	<code>np.cos(x)</code>	self-explanatory
UFunc Tangent	<code>np.tan(x)</code>	self-explanatory
UFunc Arcsinus (inverse)	<code>np.arcsin(x)</code>	self-explanatory
UFunc Arccosinus (inverse)	<code>np.arccos(x)</code>	self-explanatory
UFunc Arctangent (inverse)	<code>np.arctan(x)</code>	self-explanatory

For matrix dot operation, the number of columns in the first matrix (array) must be equal to the number of rows in the second matrix (array)

Aggregates Types

array.ufunc.reduce()	This aggregate help apply the universal function between elements and returns (reduces) the result to one value (last value)	<code>x = np.arange(1, 6)</code> <code>np.add.reduce(x)</code> result : 15
array.ufunc.accumulate()	This aggregate help apply the universal function between elements but returns the result of each element inside a list	<code>x = np.arange(1, 6)</code> <code>np.add.accumulate(x)</code> result : array([1, 3, 6, 10, 15])
array.ufunc.outer()	This aggregate help apply the ufunc operation to all pairs (a, b) with a in A and b in B.	<code>np.multiply.outer([1, 2, 3], [4, 5, 6])</code> result : array([[4, 5, 6], [8, 10, 12], [12, 15, 18]])

The aggregate for multiplication is : **array.multiply.reduce()**

The aggregate for subtraction is : **array.subtract.reduce()**

The aggregate for division is : **array.divide.reduce()**

The aggregate for addition **array.add.reduce()** is equivalent to **np.sum(array)**

The aggregate for multiplication **array.multiply.reduce()** is equivalent to **np.prod(array)**

Other Aggregate Function

<code>np.sum</code>	<code>np.nansum</code>	Compute sum of elements
<code>np.prod</code>	<code>np.nanprod</code>	Compute product of elements
<code>np.mean</code>	<code>np.nanmean</code>	Compute mean of elements
<code>np.std</code>	<code>np.nanstd</code>	Compute standard deviation
<code>np.var</code>	<code>np.nanvar</code>	Compute variance
<code>np.min</code>	<code>np.nanmin</code>	Find minimum value
<code>np.max</code>	<code>np.nanmax</code>	Find maximum value
<code>np.argmin</code>	<code>np.nanargmin</code>	Find index of minimum value
<code>np.argmax</code>	<code>np.nanargmax</code>	Find index of maximum value
<code>np.median</code>	<code>np.nanmedian</code>	Compute median of elements



By **Taissir Boukrouba**
(taissir2002)

Not published yet.

Last updated 8th November, 2023.

Page 4 of 8.

Sponsored by **Readable.com**

Measure your website readability!

<https://readable.com>

Other Aggregate Function (cont)

np.percentile	np.nanpercentile	Compute rank-based statistics of elements
np.any(condition)	N/A	Evaluate whether any elements are true
np.all(condition)	N/A	Evaluate whether all elements are true

Python Broadcasting

Broadcasting allows you to perform operations on arrays with different shapes in a way that makes sense, without the need to explicitly reshape or replicate the arrays

Rules of Broadcasting :

Not any operation is permissible when both of arrays have different shapes where :

Rule 01 - Padded Dimensions :

when two have different dimensions we can always add a padding dimension to the left , meaning pushing the first one to the right (5,) => (1,5) (adding new axis)

Rule 02 - Stretching Dimensions :

if any of the aligned dimensions is 1 , then we can stretch that dimension to match the other one

A.shape = (2,3) B.shape = (1,3) => we can stretch B to make it (2,3)

Example :

A.shape = (3, 1)

B.shape = (3,)

We can padd B => B.shape = (1,3)

This means we can stretch both A and B (Both have 1 as aligned dimension) where :

A.shape = (3, 1) => (3,3)

B.shape = (1,3) => (3,3)

if any of these rules doesn't apply then broadcasting is not possible

Numpy Fancy Indexing

Select Multiple Elements Using Fancy Indexing	select_elements = array[list_of_indexes]	select_elements = array1[[1, 2, 5, 7]]
Fancy Indexing for Sorting NumPy Array	sorted_array = array[np.argsort(array)]	np.argsort retruns indexes of array sorted (to make it reverse we just write np.argsort(- array))
Fancy Indexing to Assign New Values to Specific Elements	array[list_of_indices] = new_values	array1[[1, 3, 6]] = [10, 20, 30]
Fancy Indexing on N-d Arrays	new_array = array1[list_of_row_indices, list_of_col_indices]	selected_rows = array1[[0,2], :]

In fancy indexing we can either put a list of indices or numpy array of indices

Numpy Sorting

Items sort 1	np.sort(array)	by default it sorts the values in ascending order
--------------	----------------	---



By Taissir Boukrouba
(taissir2002)

Not published yet.
Last updated 8th November, 2023.
Page 5 of 8.

Sponsored by **Readable.com**
Measure your website readability!
<https://readable.com>

Numpy Sorting (cont)

Items sort 2	<code>array.sort()</code>	this one changes the content of the original array
Index Sorting	<code>np.argsort(array)</code>	this returns the order of the indexes sorted ascendingly
Item Sorting using Index sorting	<code>array[np.argsort(array)]</code>	this returns the items sorted ascendingly
Matrix Sorting	<code>np.sort(matrix,axis=0/1)</code>	this sorts either each matrix columns (axis=0) or each matrix row (axis=1)
Partitioning (Partial Sorting)	<code>np.partition(array, n)</code>	this takes the smallest n numbers to the left in no particular order (3 numbers if n = 3)
Matrix Partitioning	<code>np.partition(array, n, axis=0/1)</code>	this takes the smallest n numbers (of each row or column) to the left in no particular order
Index Partitioning	<code>np.argpartition(array, n, axis=0/1)</code>	this is like <code>argsort()</code> but for matrixes

Array as a Data Structure

Empty Structured Arrays	<code>np.zeros(struct_size, dtype={'names':(col1,col2,col3), 'formats':(form1, form1, form1)})</code>	<code>data = np.zeros(4, dtype={'names':('name ', 'age', 'weight'), 'formats':('U10', 'i4', 'f8')})</code> #output : <code>[(" ", 0, 0.) (" ", 0, 0.) (" ", 0, 0.) (" ", 0, 0.)]</code>
Adding Items to Structured Arrays	# List should be same size as the array structure <code>struc_array[colname] = list</code>	<code>data['name'] = name4</code> <code>data['age'] = age4</code> <code>data['weight'] = weight4</code> #output: <code>[('Alice', 25, 55.) ('Bob', 45, 85.5) ('Cathy', 37, 68.) ('Doug', 19, 61.5)]</code>
Accessing array strucure columns	<code>struc_array[colname]</code>	<code>data['name']</code>
Accessing array strucure rows	<code>struc_array[index]</code>	# Accessing last row <code>data[-1]</code>



By Taissir Boukrouba
(taissir2002)

Not published yet.
Last updated 8th November, 2023.
Page 6 of 8.

Sponsored by **Readable.com**
Measure your website readability!
<https://readable.com>

Array as a Data Structure (cont)

Sorting array structure	<code>np.sort(struct_array, order=[columns])</code>	<code>np.sort(data, order=['age', 'name'])</code>
Partitioning Structured Arrays	<code>np.partition(struct_array, n, order=[columns])</code>	# getting the smallest 2 weights to the left <code>np.partition(data, 2, order=['weight'])</code> <code>array([('Alice', 25, 55.), ('Doug', 19, 61.5), ('Cathy', 37, 68.), ('Bob', 45, 85.5)])</code>
Defining a new data type	<code>np.dtype([tuple(col_name,type)])</code>	<code>np.dtype([('name', 'U10'), ('age', 'i4'), ('weight', 'f8')])</code>
Appending new row	<code>np.append(struct_array, numpy_array, dtype=dt)</code>	<code>np.append(data, np.array([('John',64,89)], dtype=dt)</code> <code>[('Alice', 25, 55.) ('Bob', 45, 85.5) ('Cathy', 37, 68.) ('Doug', 19, 61.5) ('John', 64, 89.)]</code>

The Format For Empty Structured Arrays are as follows :

- **Integer** : `np.int32` or `'i4'`
- **Float** : `np.float32` or `'f8'`
- **String** : `np.str_` or `'U10'`



By **Taissir Boukrouba**
(taissir2002)

cheatography.com/taissir2002/

Not published yet.

Last updated 8th November, 2023.

Page 8 of 8.

Sponsored by **Readable.com**

Measure your website readability!

<https://readable.com>

