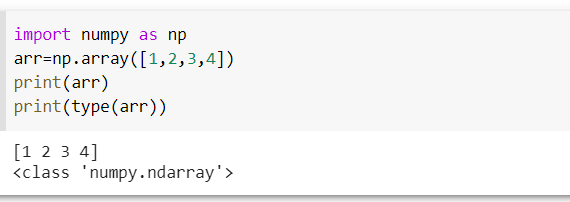
**Numpy in Python**

NumPy is a Python library.

NumPy is used for working with arrays.

NumPy is short for "Numerical Python".



## What is NumPy?

NumPy is a Python library used for working with arrays.

It also has functions for working in domain of linear algebra, fourier transform, and matrices.

NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely.

NumPy stands for Numerical Python.

## Why Use NumPy?

In Python we have lists that serve the purpose of arrays, but they are slow to process.

NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.

The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy.

Arrays are very frequently used in data science, where speed and resources are very important.

****Data Science:**** is a branch of computer science where we study how to store, use and analyze dat

## 1-D Arrays

An array that has 0-D arrays as its elements is called uni-dimensional or 1-D array.

These are the most common and basic arrays.

### Example

Create a 1-D array containing the values 1,2,3,4,5:

import numpy as np  
  
arr = np.array([1, 2, 3, 4, 5])  
  
print(arr)

## 2-D Arrays

An array that has 1-D arrays as its elements is called a 2-D array.

These are often used to represent matrix or 2nd order tensors.

NumPy has a whole sub module dedicated towards matrix operations called numpy.mat

### Example

Create a 2-D array containing two arrays with the values 1,2,3 and 4,5,6:

import numpy as np  
  
arr = np.array([[1, 2, 3], [4, 5, 6]])  
  
print(arr)

## Access 3-D Arrays

To access elements from 3-D arrays we can use comma separated integers representing the dimensions and the index of the element.

### Example

Access the third element of the second array of the first array:

import numpy as np  
  
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])  
  
print(arr[0, 1, 2])

[Try it Yourself »](https://www.w3schools.com/python/numpy/trypython.asp?filename=demo_numpy_array_access3d" \t "https://www.w3schools.com/python/numpy/_blank)

### Example Explained

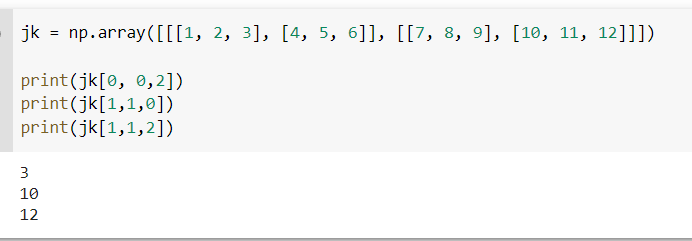
arr[0, 1, 2] prints the value 6.

And this is why:

The first number represents the first dimension, which contains two arrays:  
[[1, 2, 3], [4, 5, 6]]  
and:  
[[7, 8, 9], [10, 11, 12]]  
Since we selected 0, we are left with the first array:  
[[1, 2, 3], [4, 5, 6]]

The second number represents the second dimension, which also contains two arrays:  
[1, 2, 3]  
and:  
[4, 5, 6]  
Since we selected 1, we are left with the second array:  
[4, 5, 6]

The third number represents the third dimension, which contains three values:  
4  
5  
6  
Since we selected 2, we end up with the third value:  
6



## Slicing arrays

Slicing in python means taking elements from one given index to another given index.

We pass slice instead of index like this: [start:end].

We can also define the step, like this: [start:end:step].

If we don't pass start its considered 0

If we don't pass end its considered length of array in that dimension

If we don't pass step its considered 1

### Example

Slice elements from index 1 to index 5 from the following array:

import numpy as np  
  
arr = np.array([1, 2, 3, 4, 5, 6, 7])  
  
print(arr[1:5])

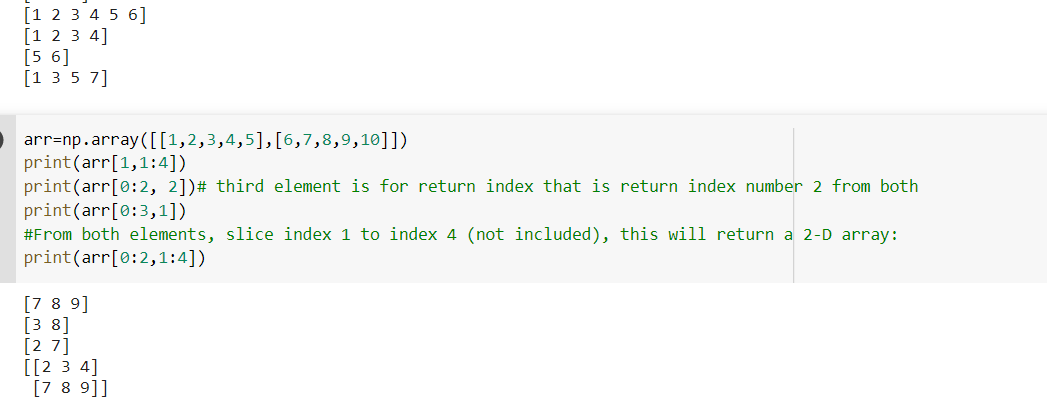
## Negative Slicing

Use the minus operator to refer to an index from the end:

### Example

Slice from the index 3 from the end to index 1 from the end:

import numpy as np  
  
arr = np.array([1, 2, 3, 4, 5, 6, 7])  
  
print(arr[-3:-1])



## Data Types in Python

By default Python have these data types:

* strings - used to represent text data, the text is given under quote marks. e.g. "ABCD"
* integer - used to represent integer numbers. e.g. -1, -2, -3
* float - used to represent real numbers. e.g. 1.2, 42.42
* boolean - used to represent True or False.
* complex - used to represent complex numbers. e.g. 1.0 + 2.0j, 1.5 + 2.5j

## Data Types in NumPy

NumPy has some extra data types, and refer to data types with one character, like i for integers, u for unsigned integers etc.

Below is a list of all data types in NumPy and the characters used to represent them.

* i - integer
* b - boolean
* u - unsigned integer
* f - float
* c - complex float
* m - timedelta
* M - datetime
* O - object
* S - string
* U - unicode string
* V - fixed chunk of memory for other type ( void )

## Converting Data Type on Existing Arrays

The best way to change the data type of an existing array, is to make a copy of the array with the astype() method.

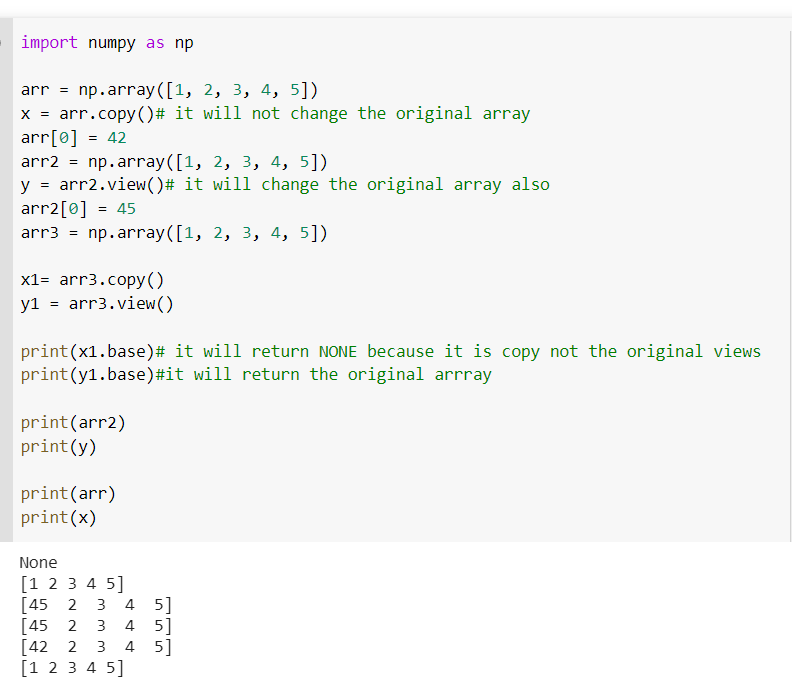
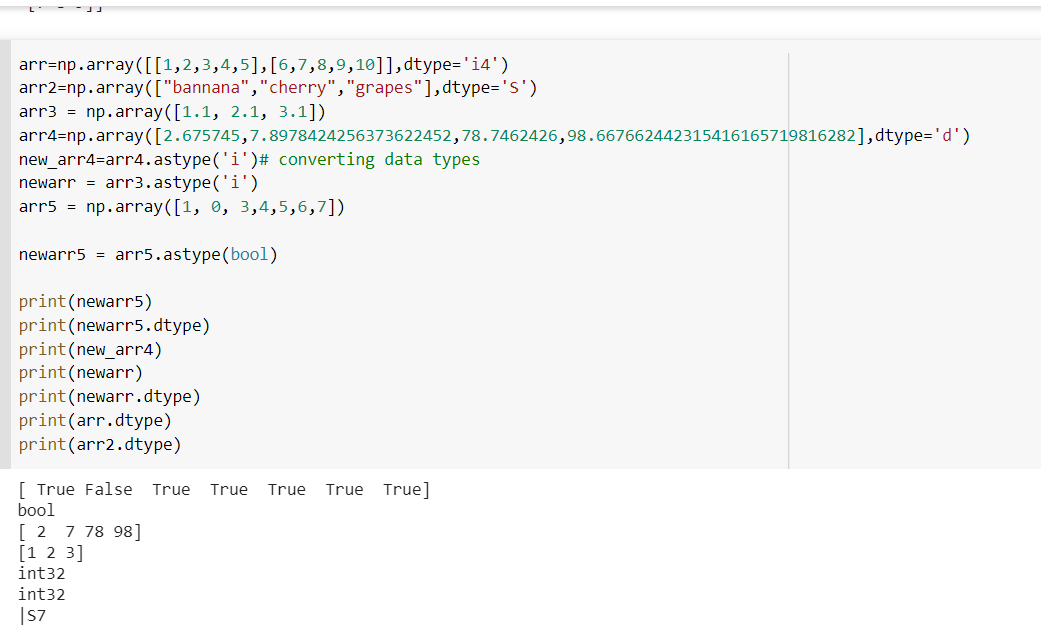
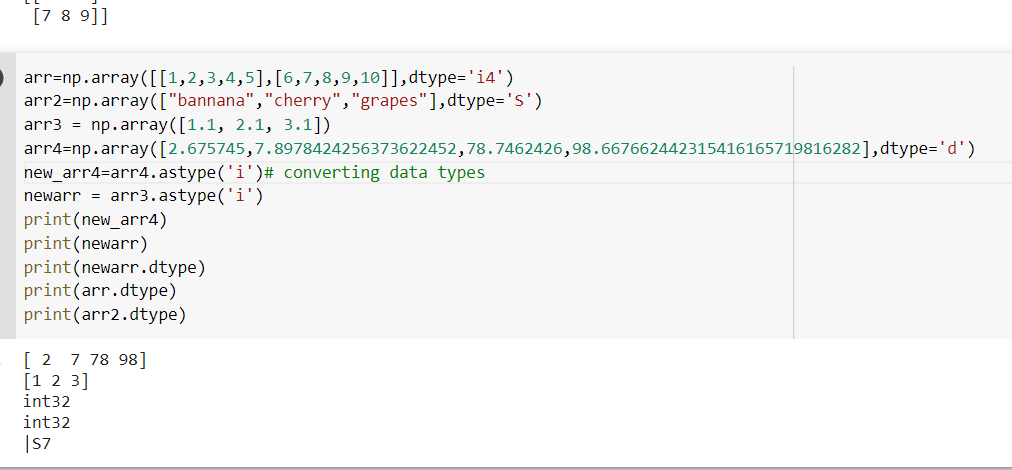
The astype() function creates a copy of the array, and allows you to specify the data type as a parameter.

The data type can be specified using a string, like 'f' for float, 'i' for integer etc. or you can use the data type directly like float for float and int for integer.

### Example

Change data type from float to integer by using 'i' as parameter value:

import numpy as np  
  
arr = np.array([1.1, 2.1, 3.1])  
  
newarr = arr.astype('i')  
  
print(newarr)  
print(newarr.dtype)



## Get the Shape of an Array

NumPy arrays have an attribute called shape that returns a tuple with each index having the number of corresponding elements.

## Reshaping arrays

Reshaping means changing the shape of an array.

The shape of an array is the number of elements in each dimension.

By reshaping we can add or remove dimensions or change number of elements in each dimension.

## Reshape From 1-D to 2-D

### Example

Convert the following 1-D array with 12 elements into a 2-D array.

The outermost dimension will have 4 arrays, each with 3 elements:

import numpy as np  
  
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])  
  
newarr = arr.reshape(4, 3)  
  
print(newarr)

## Unknown Dimension

You are allowed to have one "unknown" dimension.

Meaning that you do not have to specify an exact number for one of the dimensions in the reshape method.

Pass -1 as the value, and NumPy will calculate this number for you.

### Example

Convert 1D array with 8 elements to 3D array with 2x2 elements:

import numpy as np  
  
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])  
  
newarr = arr.reshape(2, 2, -1)  
  
print(newarr)

There are a lot of functions for changing the shapes of arrays in numpy flatten, ravel and also for rearranging the elements rot90, flip, fliplr, flipud etc. These fall under Intermediate to Advanced section of numpy.

import numpy as np

#Convert the following 1-D array with 12 elements into a 2-D array.

arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])

arr1=np.array([11,12,13,14,15,16,17,18,19,20,1,2,3,4,5,6,7,8,9,21,22,23,24,25])# 1-D array

new\_arr1=arr1.reshape(8,3)# 8 arrays with each 3 elements

newarr = arr.reshape(4, 3)#The outermost dimension will have 4 arrays, each with 3 elements:

arr2 = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])#  1-D array with 12 elements into a 3-D array.

newarr2 = arr2.reshape(2, 3, 2)# The outermost dimension will have 2 arrays that contains 3 arrays, each with 2 elements:

arr3 = np.array([1, 2, 3, 4, 5, 6, 7, 8])

arr4 = np.array([[1, 2, 3], [4, 5, 6]])

newarr4 = arr4.reshape(-1)

print(newarr4)

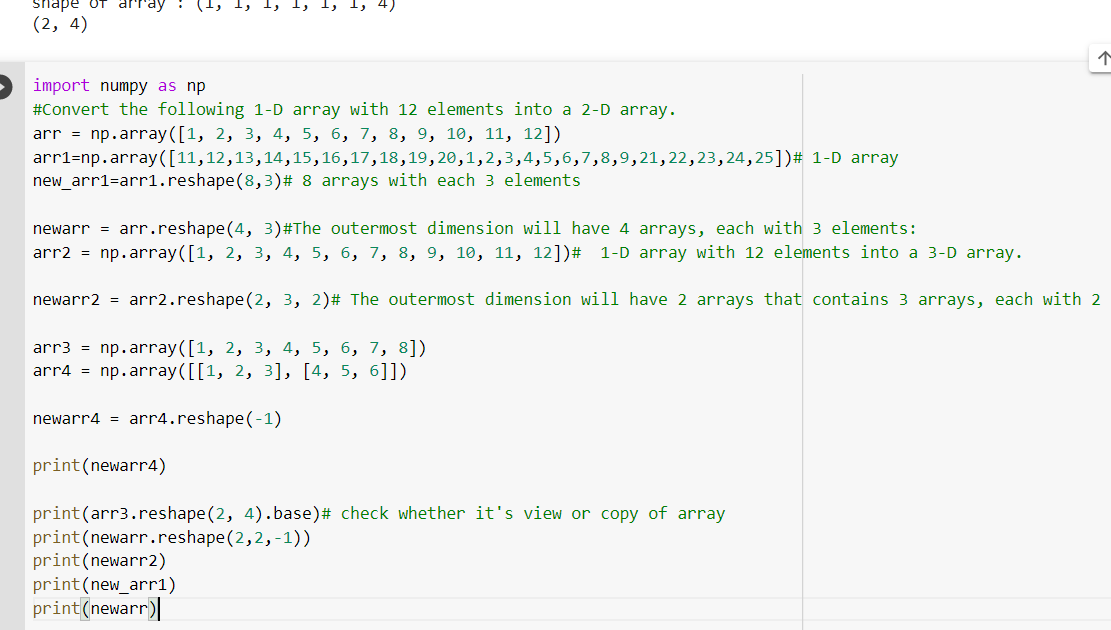
print(arr3.reshape(2, 4).base)# check whether it's view or copy of array

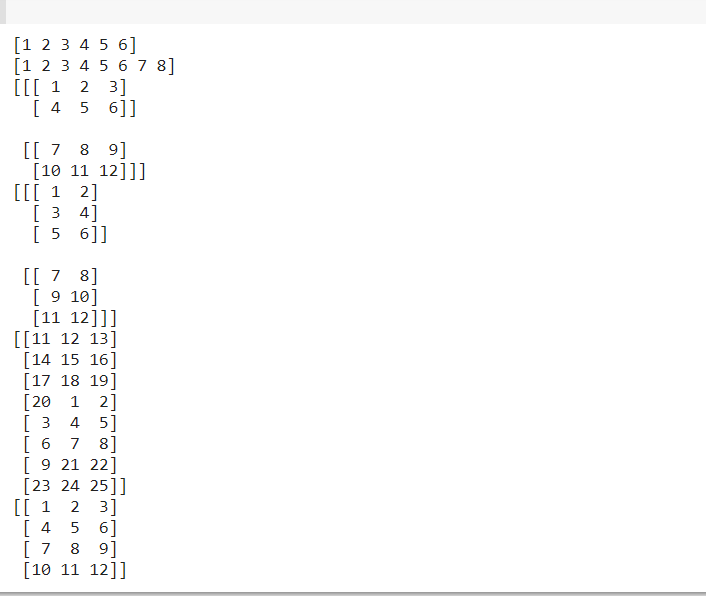
print(newarr.reshape(2,2,-1))

print(newarr2)

print(new\_arr1)

print(newarr)





## Iterating Arrays

Iterating means going through elements one by one.

As we deal with multi-dimensional arrays in numpy, we can do this using basic for loop of python.

If we iterate on a 1-D array it will go through each element one by one.

## Iterating Arrays Using nditer()

The function nditer() is a helping function that can be used from very basic to very advanced iterations. It solves some basic issues which we face in iteration, lets go through it with examples.

### Iterating on Each Scalar Element

In basic for loops, iterating through each scalar of an array we need to use n for loops which can be difficult to write for arrays with very high dimensionality.

### Example

Iterate through the following 3-D array:

import numpy as np  
  
arr = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])  
  
for x in np.nditer(arr):  
  print(x)

## Enumerated Iteration Using ndenumerate()

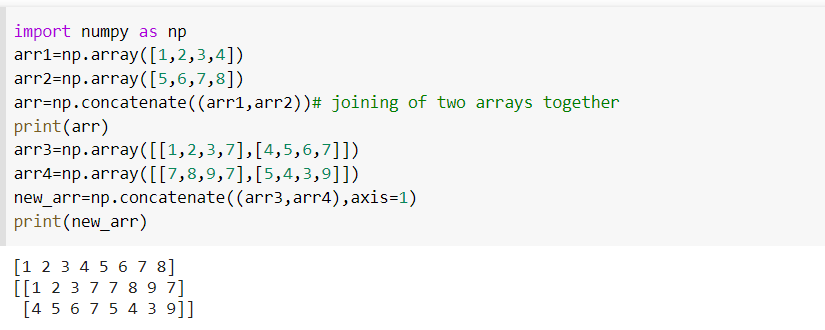
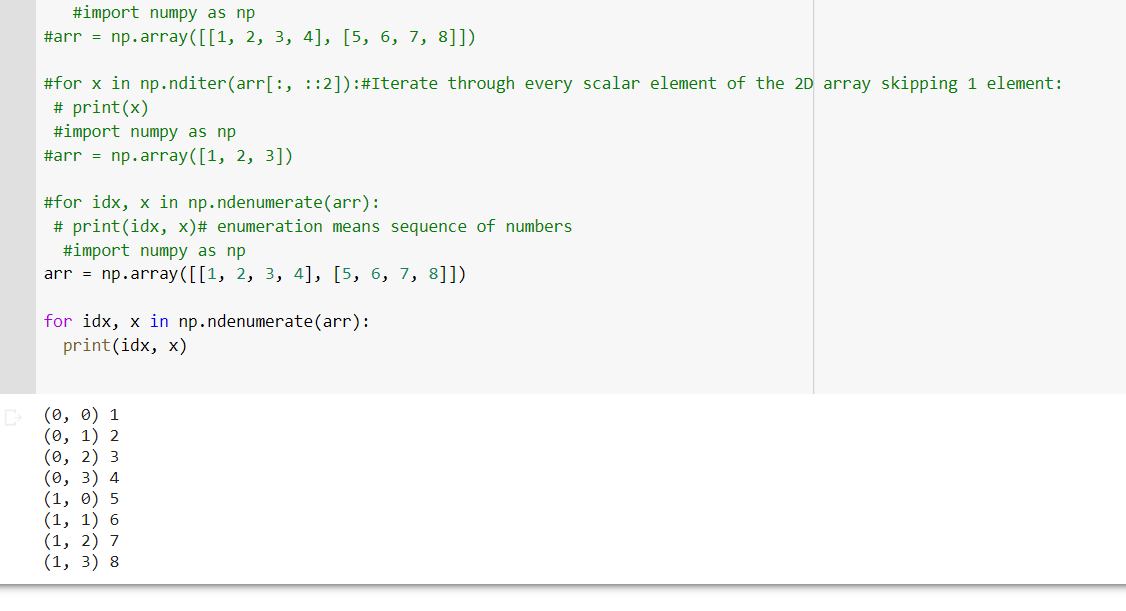
Enumeration means mentioning sequence number of somethings one by one.

Sometimes we require corresponding index of the element while iterating, the ndenumerate() method can be used for those usecases.

### Example

Enumerate on following 1D arrays elements:

import numpy as np  
  
arr = np.array([1, 2, 3])  
  
for idx, x in np.ndenumerate(arr):  
  print(idx, x)



## Joining Arrays Using Stack Functions

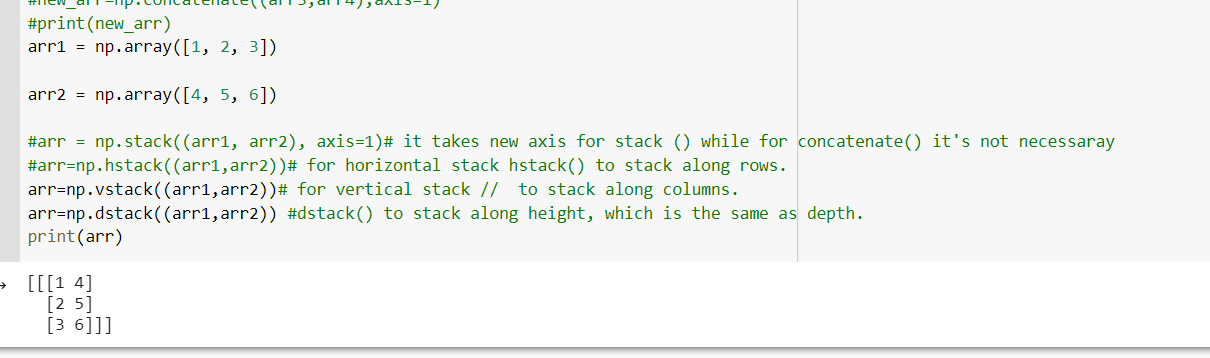
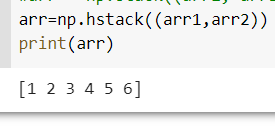
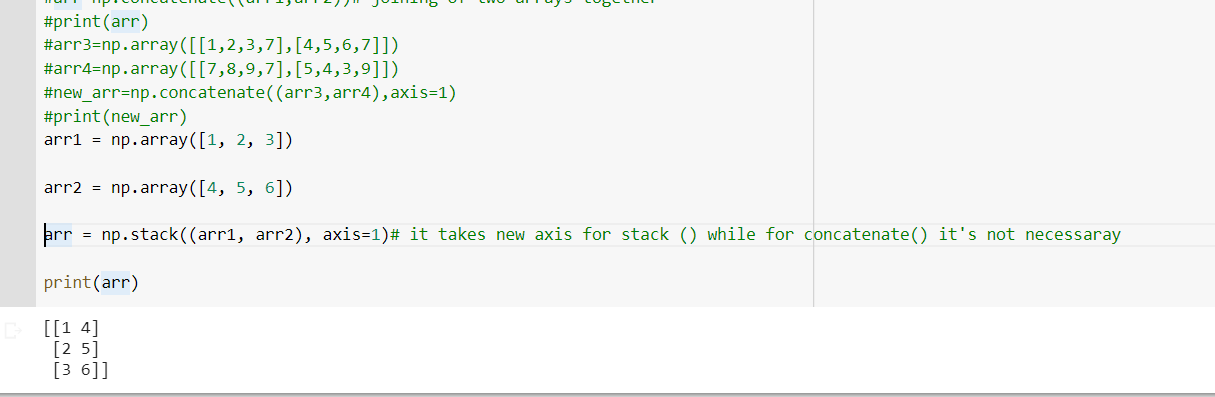
Stacking is same as concatenation, the only difference is that stacking is done along a new axis.

We can concatenate two 1-D arrays along the second axis which would result in putting them one over the other, ie. stacking.

We pass a sequence of arrays that we want to join to the stack() method along with the axis. If axis is not explicitly passed it is taken as 0.

### Example

import numpy as np  
  
arr1 = np.array([1, 2, 3])  
  
arr2 = np.array([4, 5, 6])  
  
arr = np.stack((arr1, arr2), axis=1)  
  
print(arr)

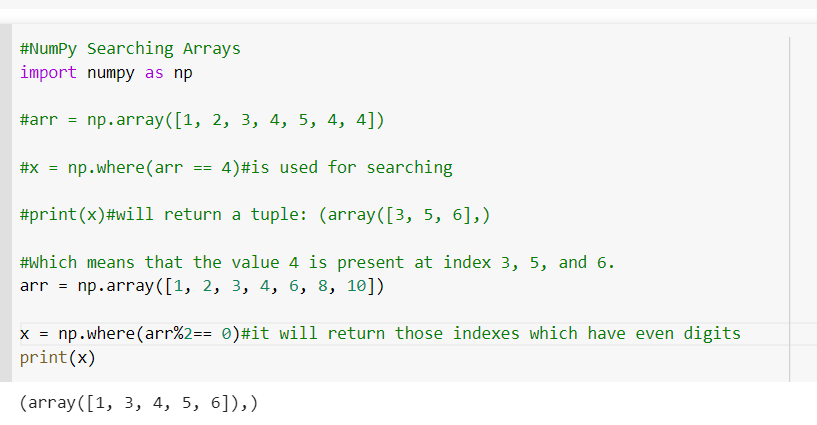
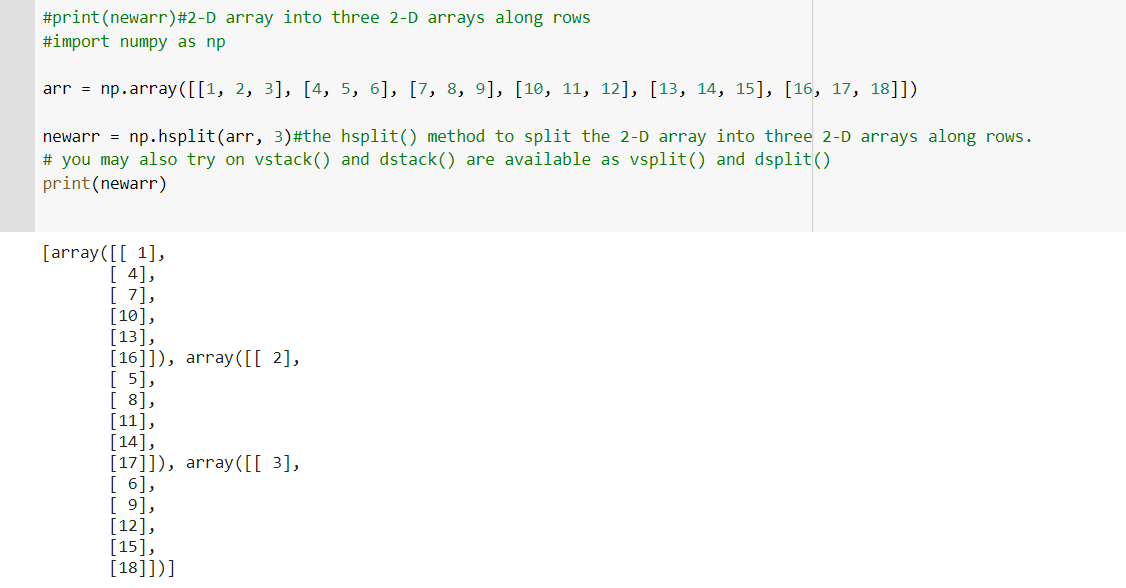
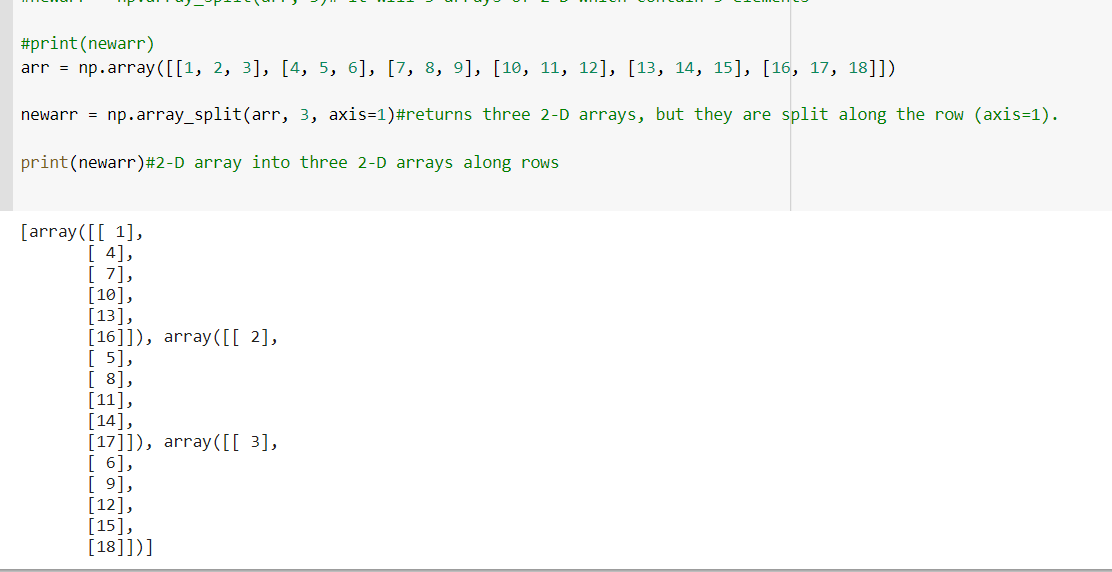
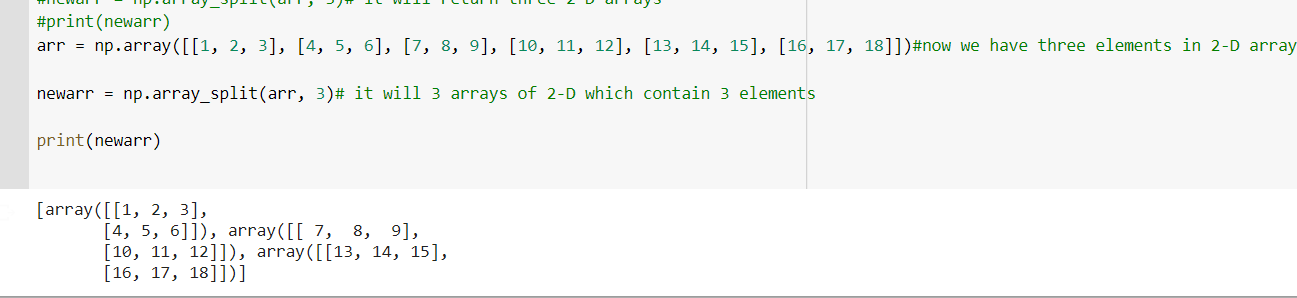
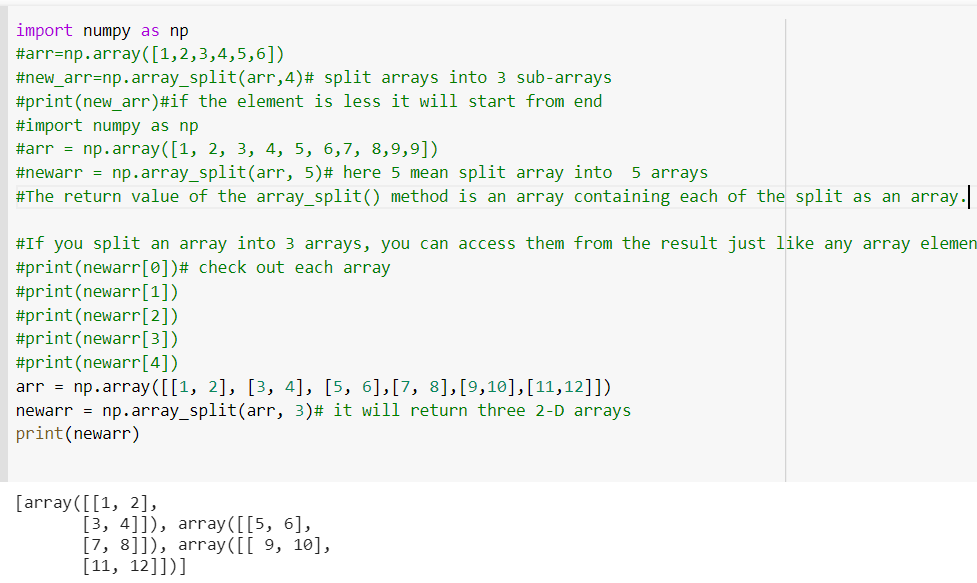
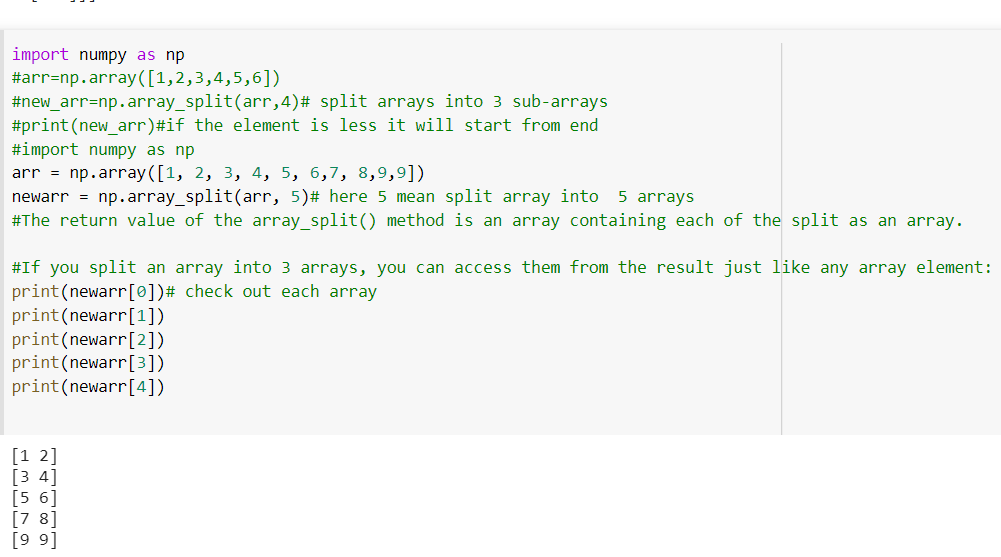
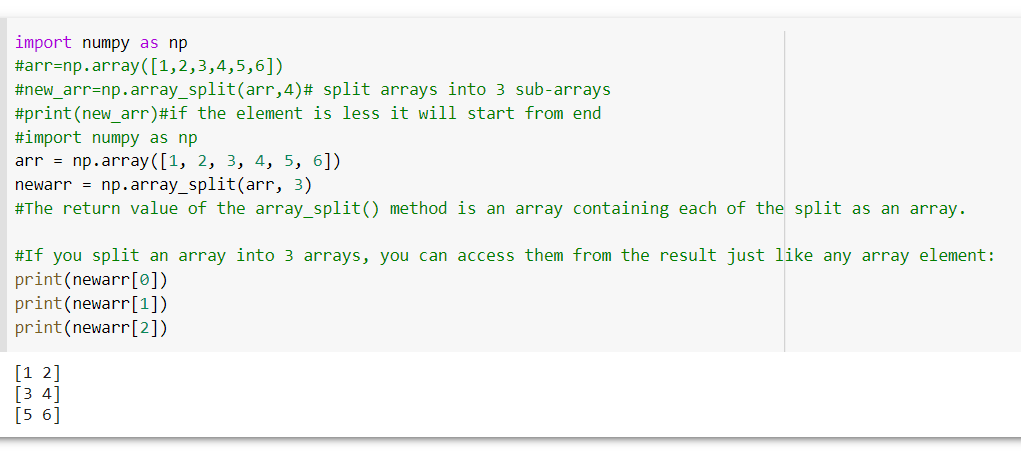
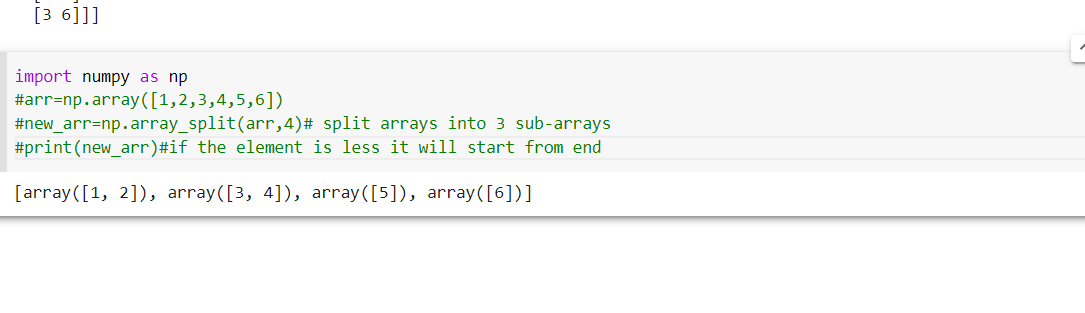


## Splitting NumPy Arrays

Splitting is reverse operation of Joining.

Joining merges multiple arrays into one and Splitting breaks one array into multiple.

We use array\_split() for splitting arrays, we pass it the array we want to split and the number of splits.



Find the indexes where the values are odd:

import numpy as np  
  
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])  
  
x = np.where(arr%2 == 1)  
  
print(x)

## Search Sorted

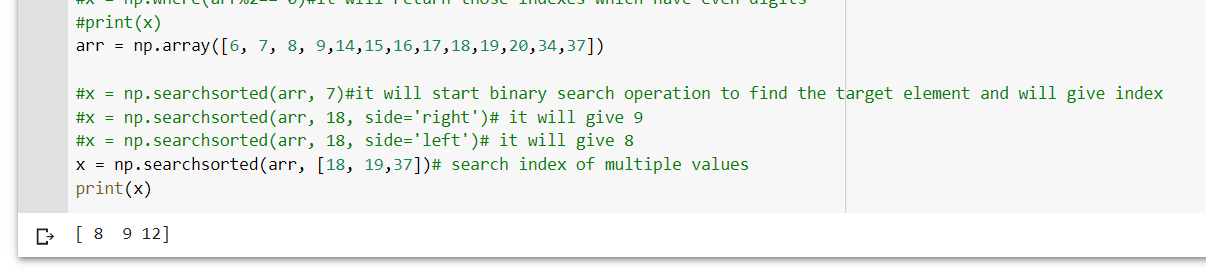
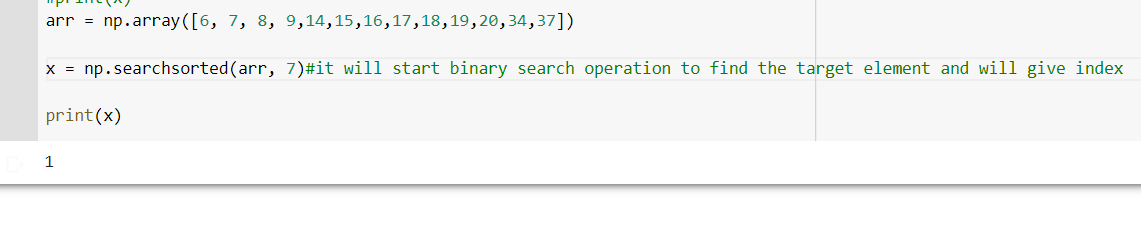
There is a method called searchsorted() which performs a binary search in the array, and returns the index where the specified value would be inserted to maintain the search order.

The searchsorted() method is assumed to be used on sorted arrays.

### Example

Find the indexes where the value 7 should be inserted:

import numpy as np  
  
arr = np.array([6, 7, 8, 9])  
  
x = np.searchsorted(arr, 7)  
  
print(x)



## Sorting Arrays

Sorting means putting elements in an ordered sequence.

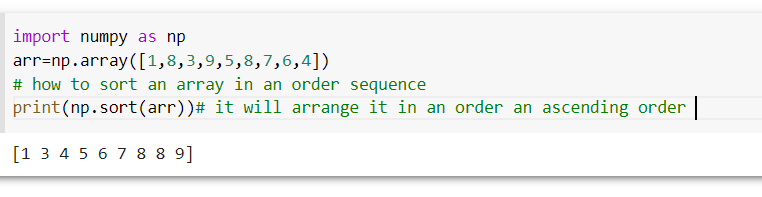
Ordered sequence is any sequence that has an order corresponding to elements, like numeric or alphabetical, ascending or descending.

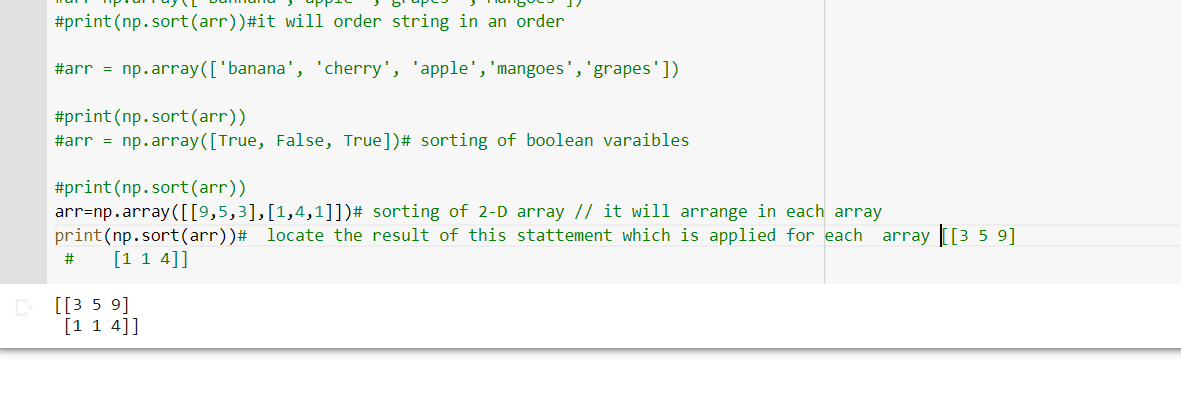
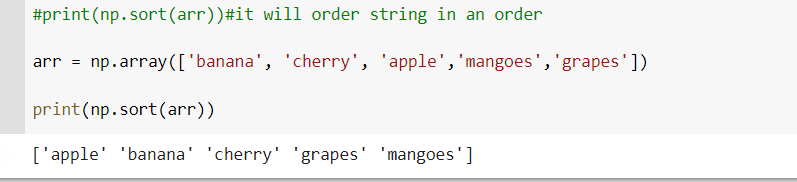
The NumPy ndarray object has a function called sort(), that will sort a specified array.

### Example

Sort the array:

import numpy as np  
  
arr = np.array([3, 2, 0, 1])  
  
print(np.sort(arr))





## Filtering Arrays

Getting some elements out of an existing array and creating a new array out of them is called filtering.

In NumPy, you filter an array using a boolean index list.

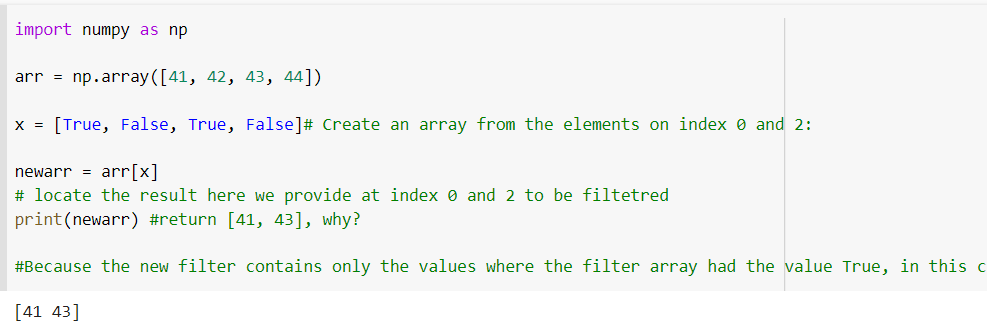
A boolean index list is a list of booleans corresponding to indexes in the array.

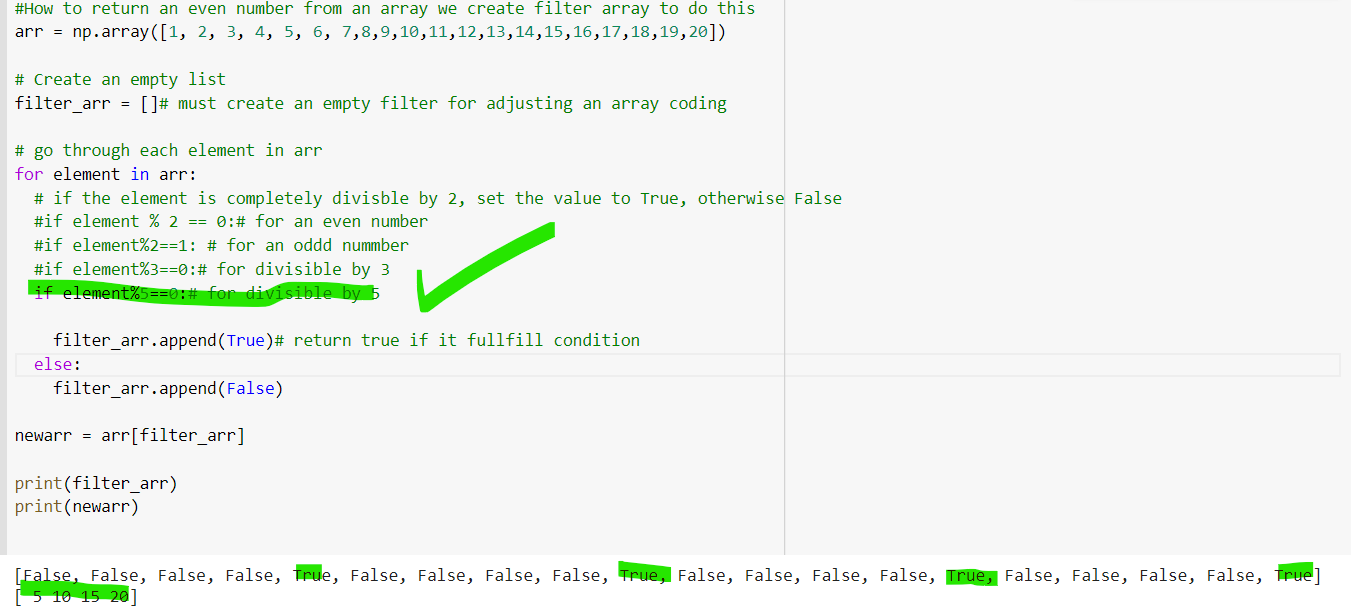
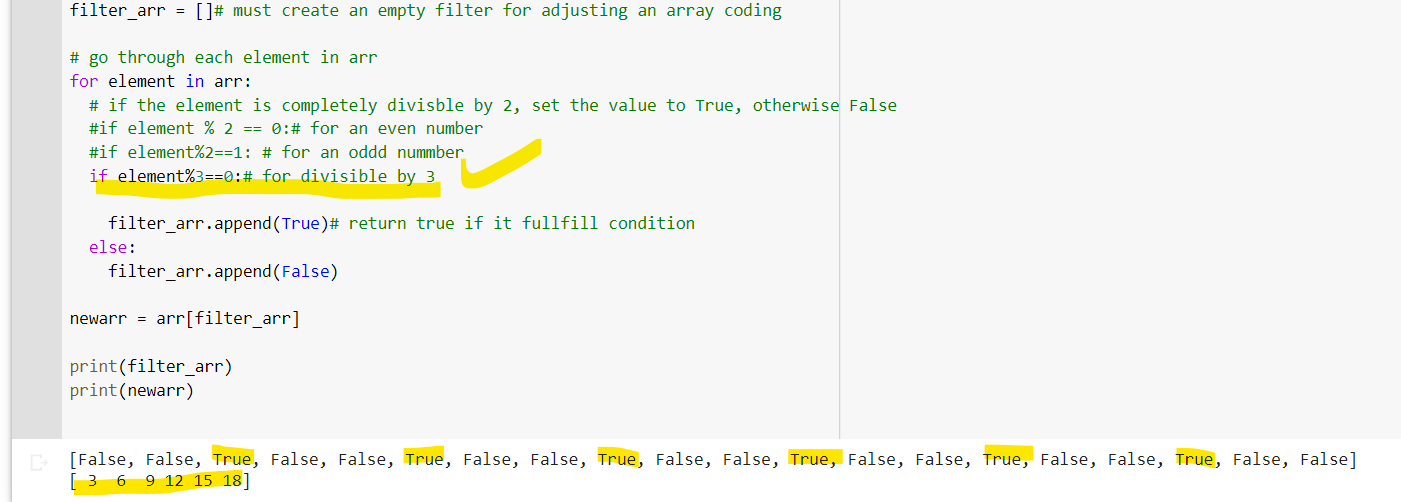
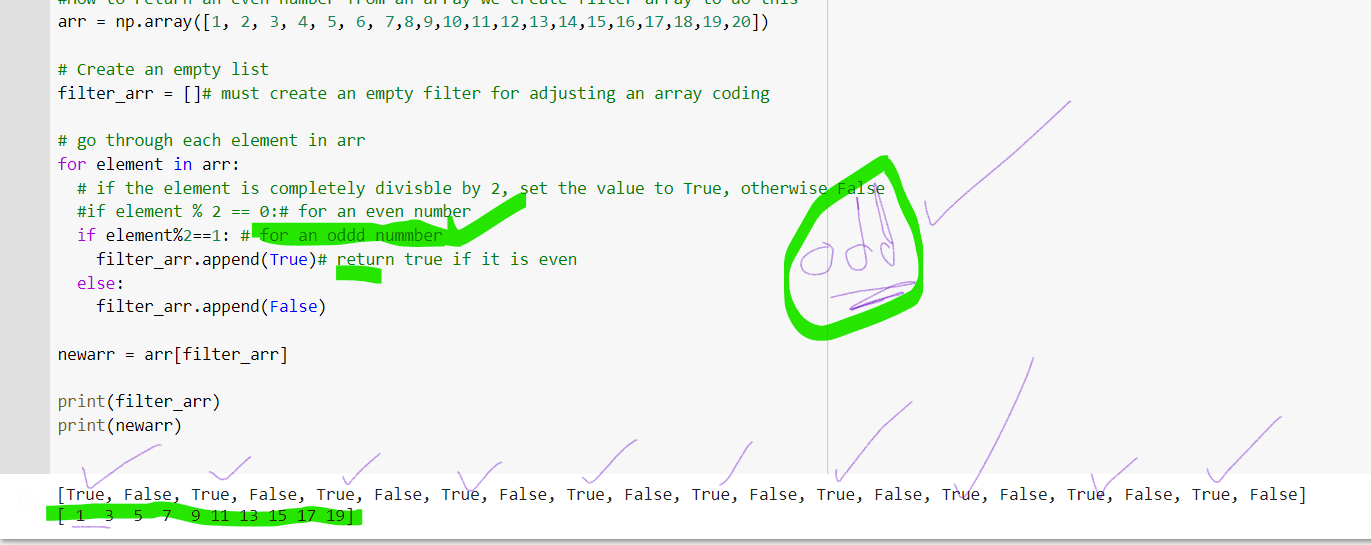
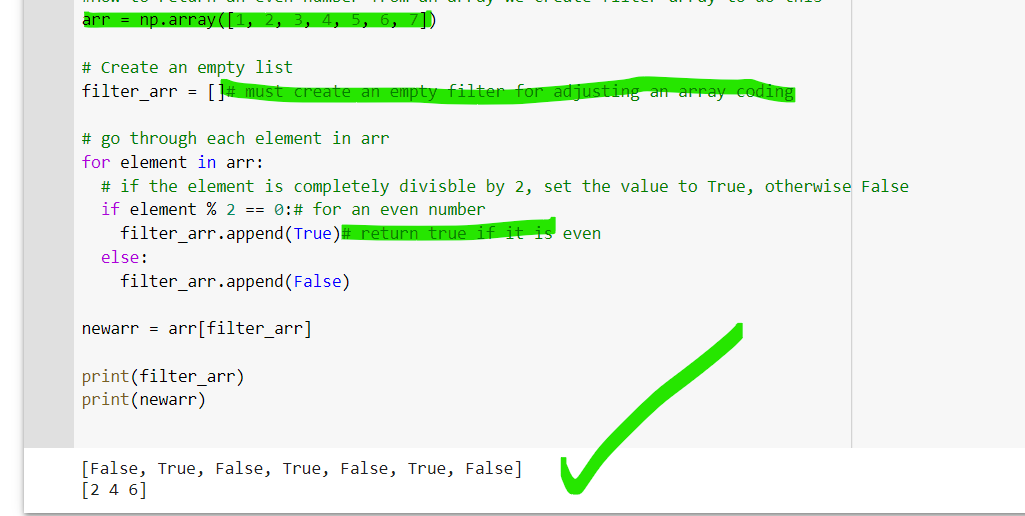
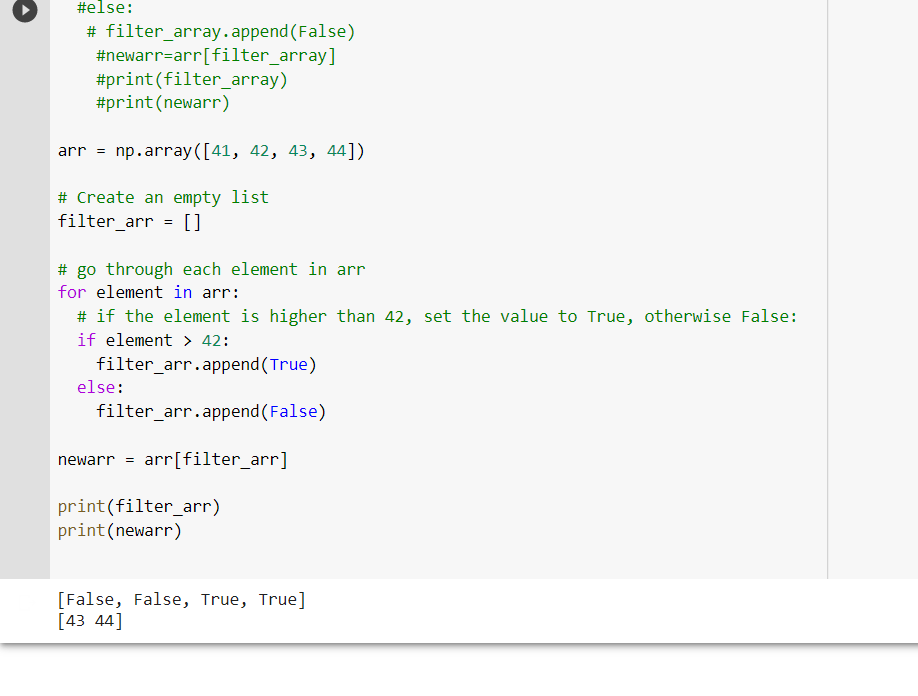
If the value at an index is True that element is contained in the filtered array, if the value at that index is False that element is excluded from the filtered array.

### Example

Create an array from the elements on index 0 and 2:

import numpy as np  
  
arr = np.array([41, 42, 43, 44])  
  
x = [True, False, True, False]  
  
newarr = arr[x]  
  
print(newarr)





## What is a Random Number?

Random number does NOT mean a different number every time. Random means something that can not be predicted logically.

## Pseudo Random and True Random.

Computers work on programs, and programs are definitive set of instructions. So it means there must be some algorithm to generate a random number as well.

If there is a program to generate random number it can be predicted, thus it is not truly random.

Random numbers generated through a generation algorithm are called pseudo random.

Can we make truly random numbers?

Yes. In order to generate a truly random number on our computers we need to get the random data from some outside source. This outside source is generally our keystrokes, mouse movements, data on network etc.

We do not need truly random numbers, unless its related to security (e.g. encryption keys) or the basis of application is the randomness (e.g. Digital roulette wheels).

In this tutorial we will be using pseudo random numbers.

