**NumPy(Numerical Python)**

**NumPy(Numerical Python)** is a fundamental library for Python **numerical computing**. It provides efficient multi-dimensional array objects and various mathematical functions for handling large datasets making it a critical tool for professionals in fields that require heavy computation.

NumPy (Numerical Python) is a **powerful library** for numerical computing in Python. It is mainly used for **working with arrays, matrices, and performing mathematical operations** efficiently.

**Installing NumPy in Python**

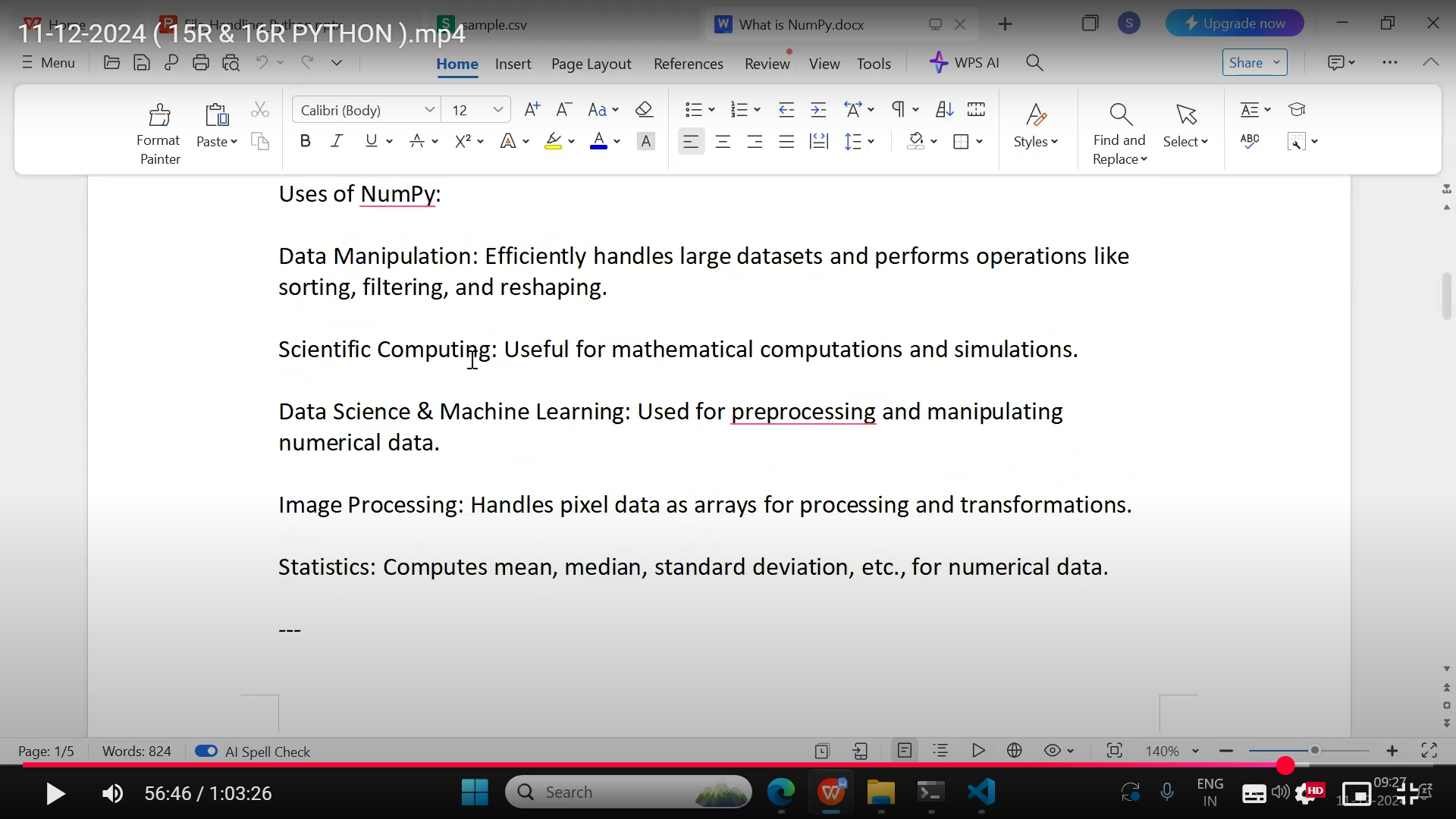
To begin using NumPy, you need to install it first. This can be done through pip command:

***pip install numpy***

**Key Features of NumPy**

NumPy has various features that make it popular over lists.

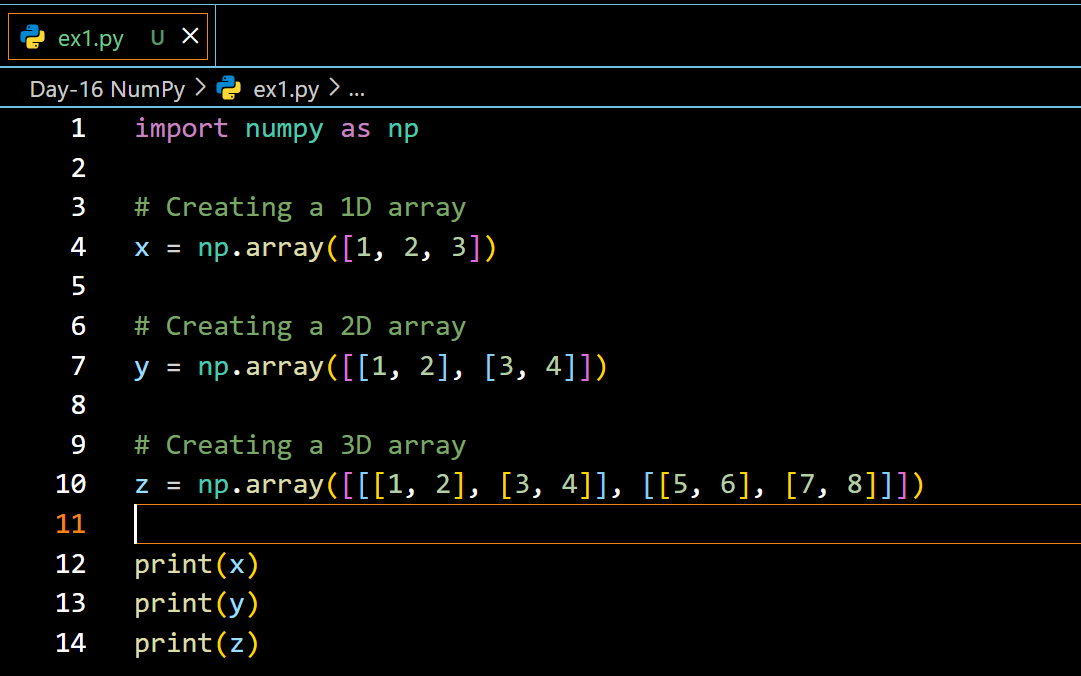
* **N-Dimensional Arrays**: NumPy’s core feature is ndarray, a powerful N-dimensional array object that supports homogeneous data types.
* **Arrays with High Performance**: Arrays are stored in contiguous memory locations, enabling faster computations than Python lists(Please see [Numpy Array vs Python List](https://www.geeksforgeeks.org/python-lists-vs-numpy-arrays/" \t "_blank) for details).
* [**Broadcasting**](https://www.geeksforgeeks.org/numpy-array-broadcasting/)**:** This allows element-wise computations between arrays of different shapes. It simplifies operations on arrays of **various shapes** by automatically aligning their dimensions without creating new data.
* [**Vectorization**](https://www.geeksforgeeks.org/vectorization-in-python/)**:** Eliminates the need for explicit Python loops by applying operations directly on entire arrays.
* **Linear algebra**: NumPy contains routines for linear algebra operations, such as matrix multiplication, decompositions, and determinants.

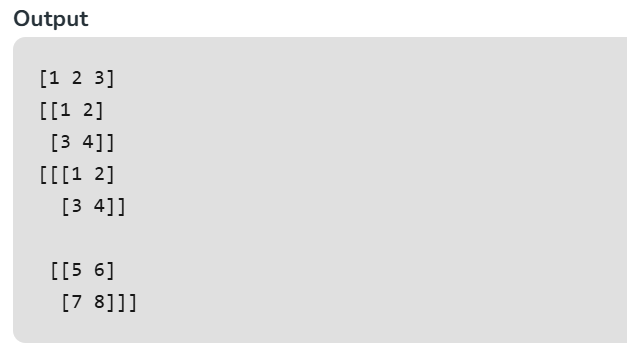


**Create a NumPy ndarray Object**

NumPy is used to work with arrays. The array object in NumPy is called ndarray.

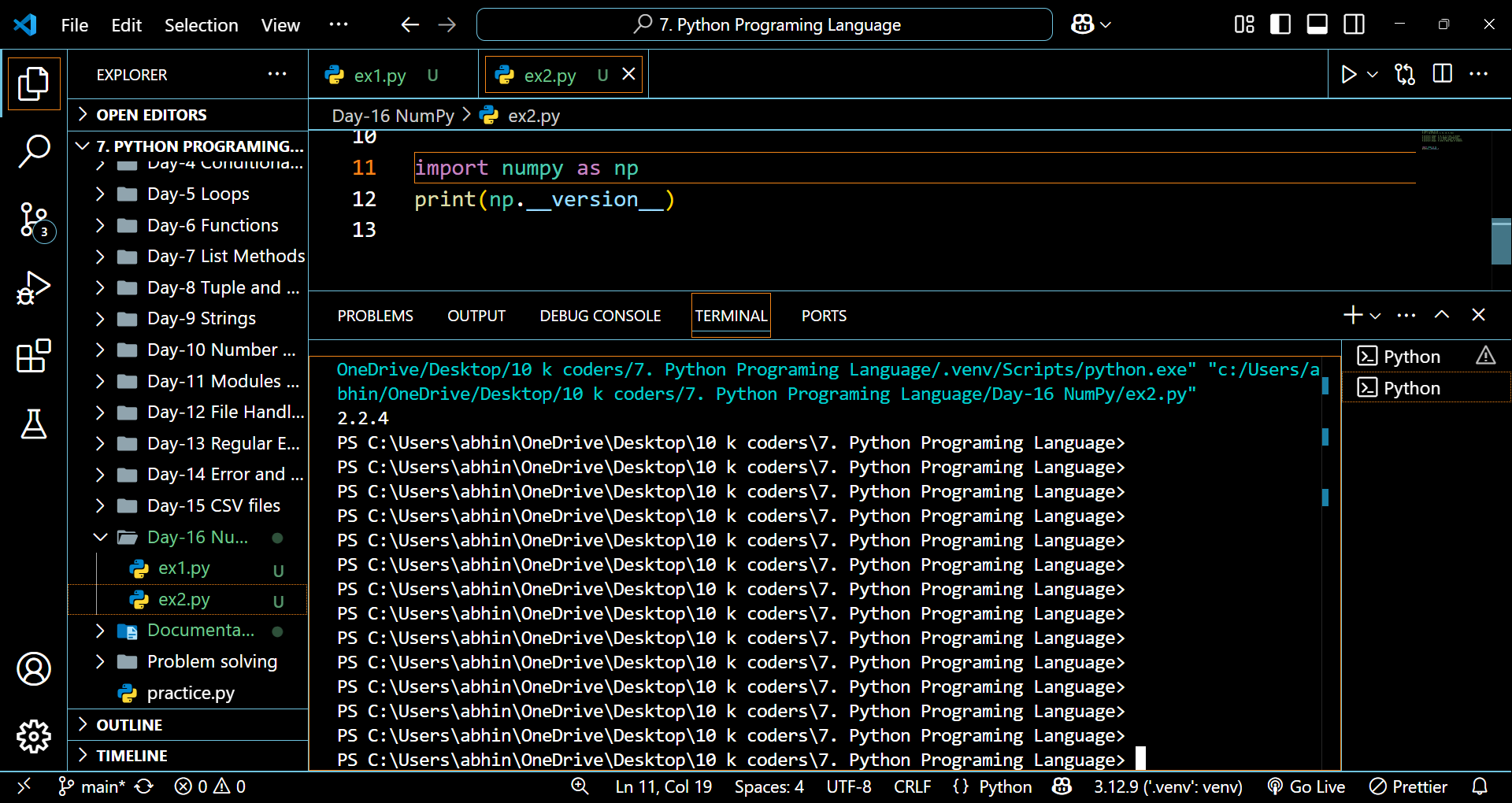
We can create a NumPy ndarray object by using the array() function.





**Checking NumPy Version**

The version string is stored under \_\_version\_\_ attribute.



**Access Array Elements**

Array indexing is the same as accessing an array element.

You can access an array element by referring to its index number.

The indexes in NumPy arrays start with 0, meaning that the first element has index 0, and the second has index 1 etc.

**Example:**

import numpy as np

arr = np.array([1, 2, 3, 4])

print(arr[0]) // 1

**Access 2-D Arrays**

To access elements from 2-D arrays we can use comma separated integers representing the dimension and the index of the element.

Think of 2-D arrays like a table with rows and columns, where the dimension represents the row and the index represents the column.

**Example**

Access the element on the first row, second column:

import numpy as np  
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])  
print('2nd element on 1st row: ', arr[0, 1])

**NumPy Array Slicing**

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

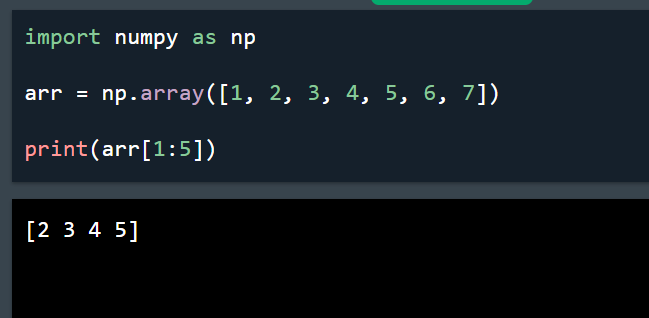
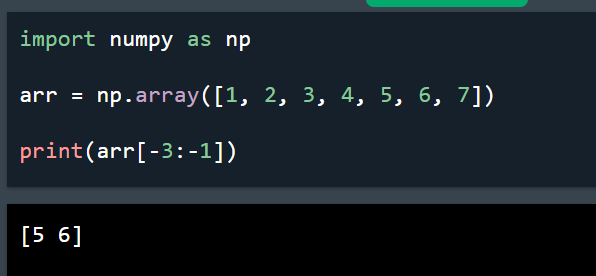
We pass slice instead of index like this: [*start*:*end*].

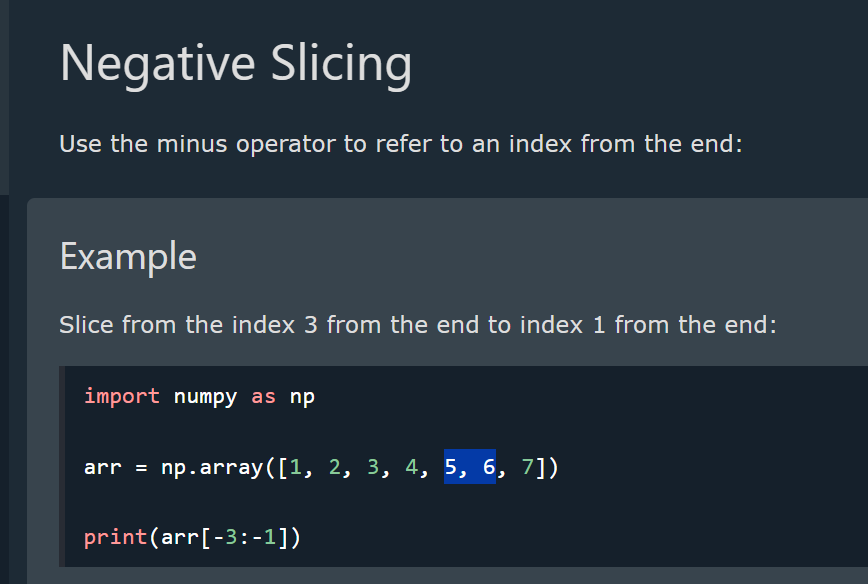
We can also define the step, like this: [*start*:*end*:*step*].

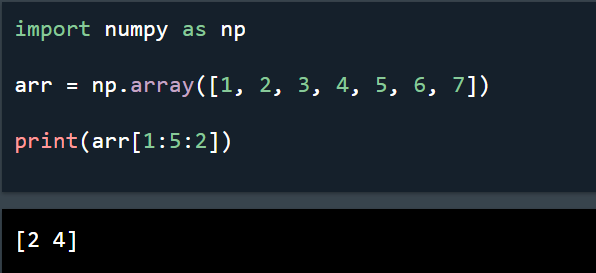
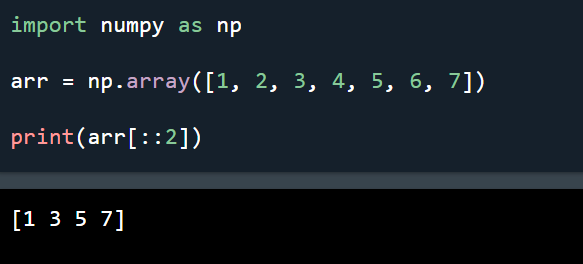
If we don't pass start its considered 0

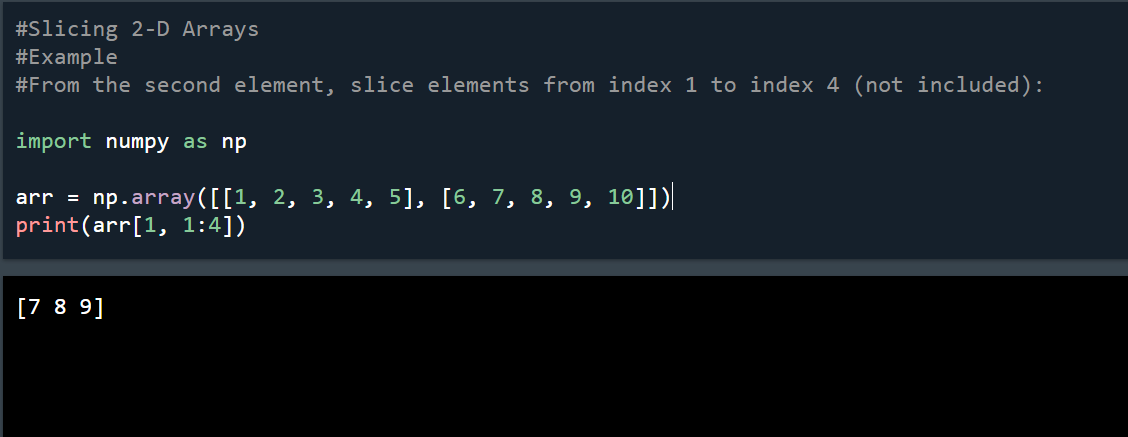
If we don't pass end its considered length of array in that dimension

If we don't pass step its considered 1



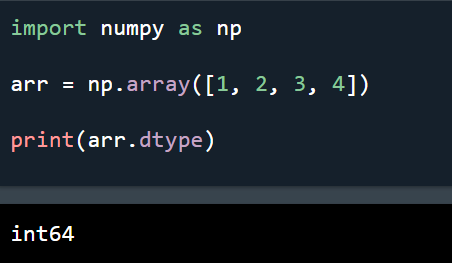


**Data Types in NumPy**

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

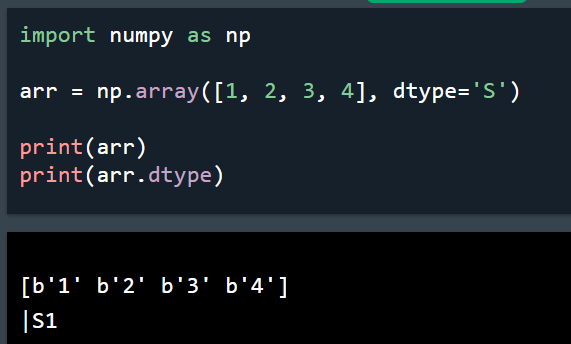
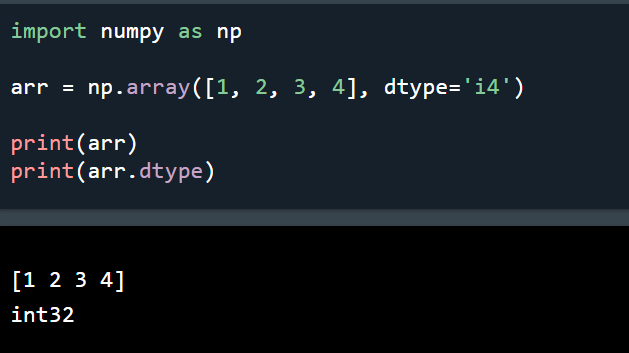
Below is a list of all data types in NumPy and the characters used to represent them.

* 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 )



**Creating Arrays With a Defined Data Type**

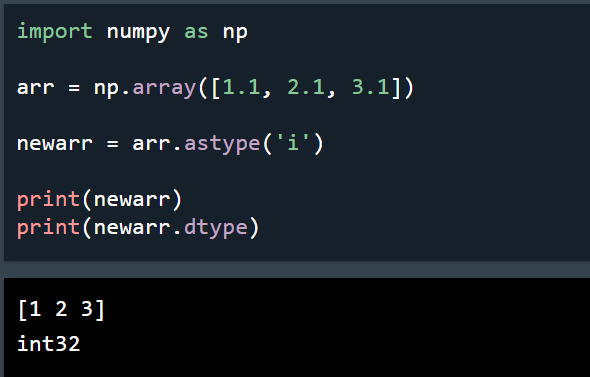
We use the array() function to create arrays, this function can take an optional argument: dtype that allows us to define the expected data type of the array elements:

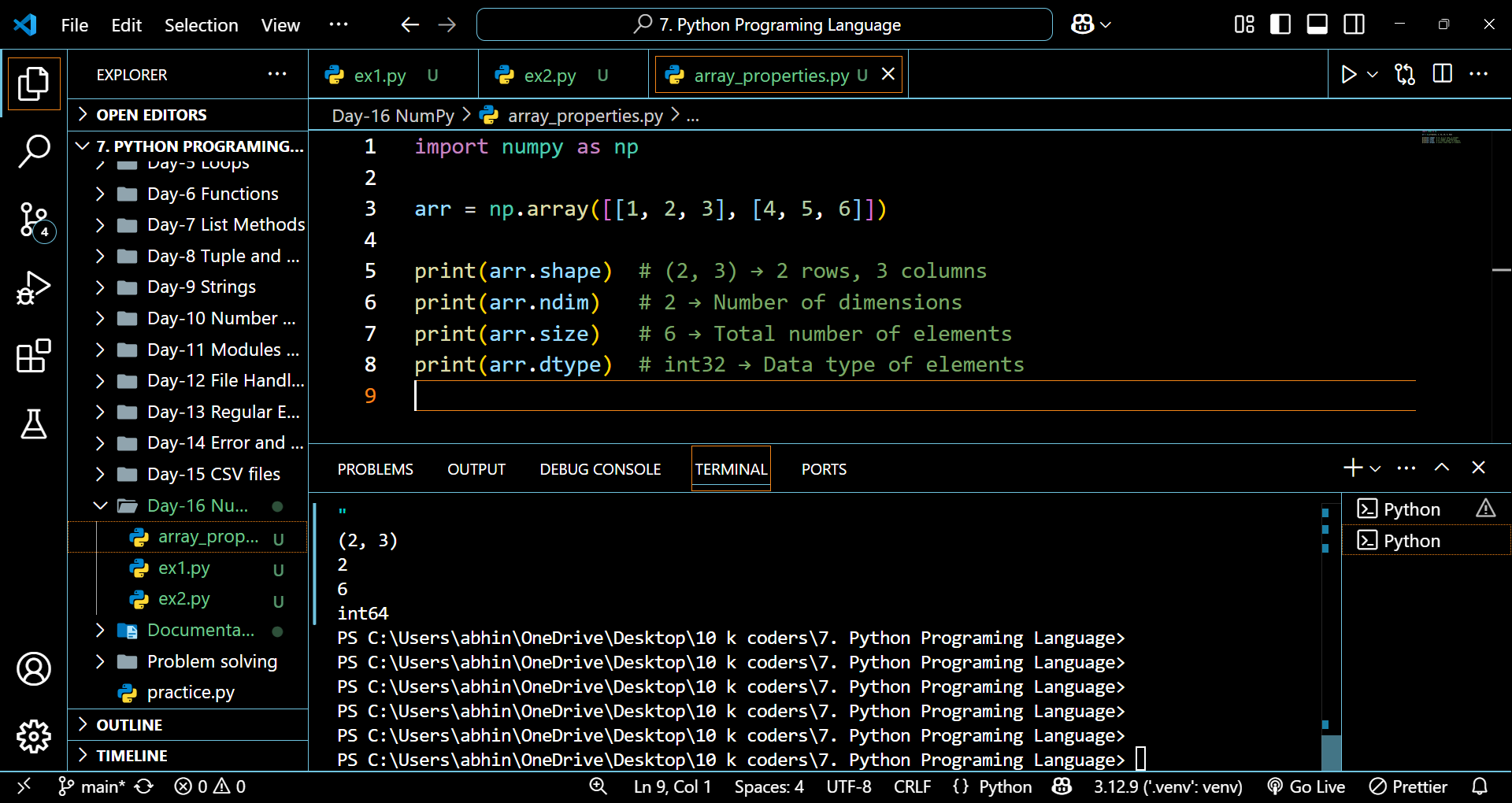
**Converting Data Type on Existing Arrays**

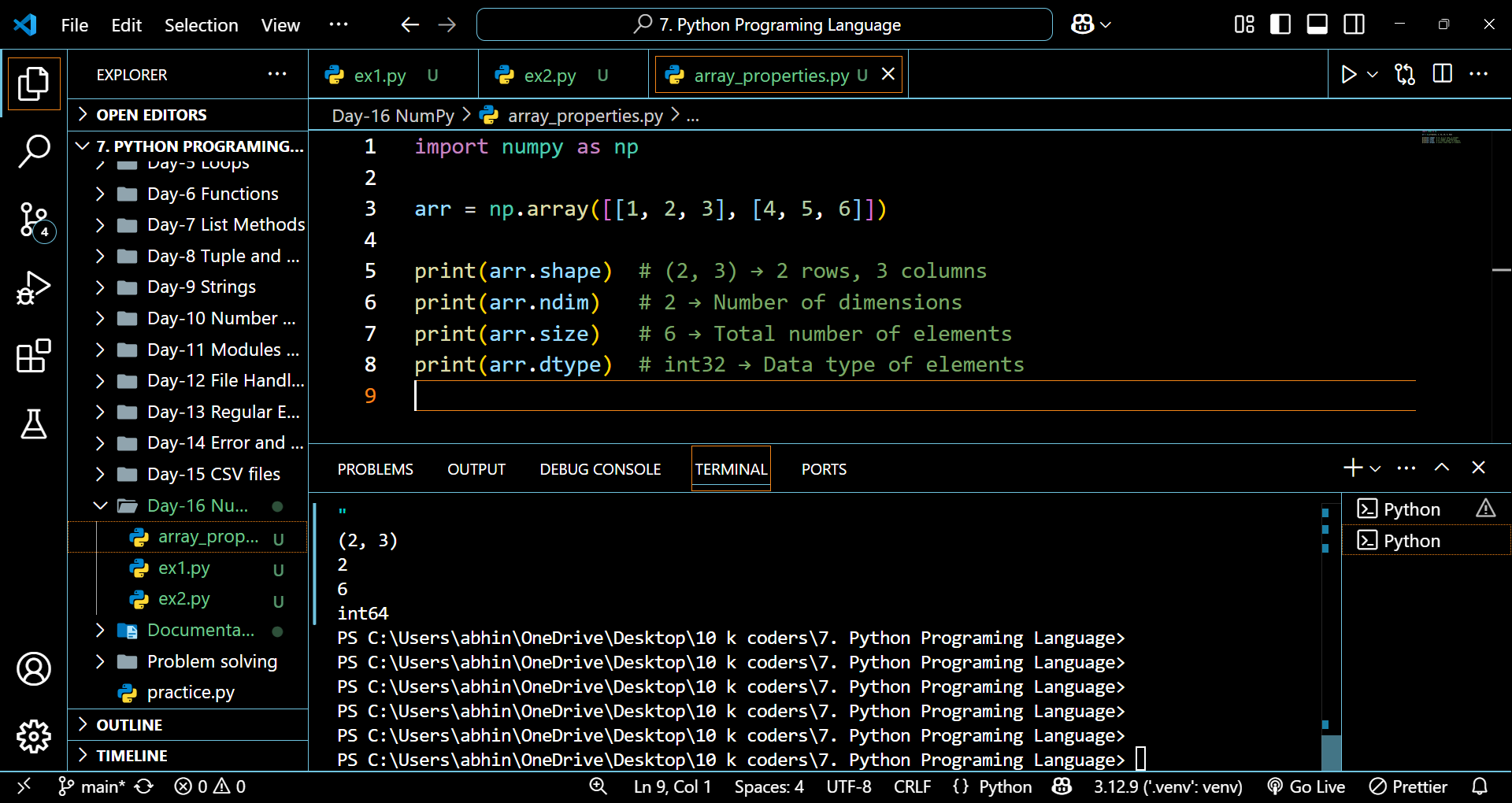
The best way to change the data type of an existing array, is to make a copy of the array with the astype() method.

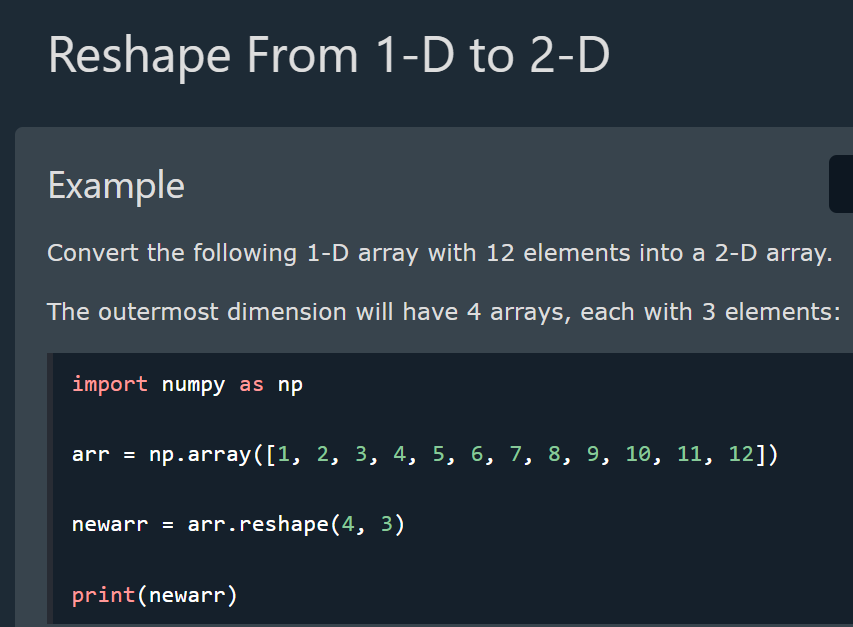
The astype() function creates a copy of the array, and allows you to specify the data type as a parameter.



**NumPy Array Properties**





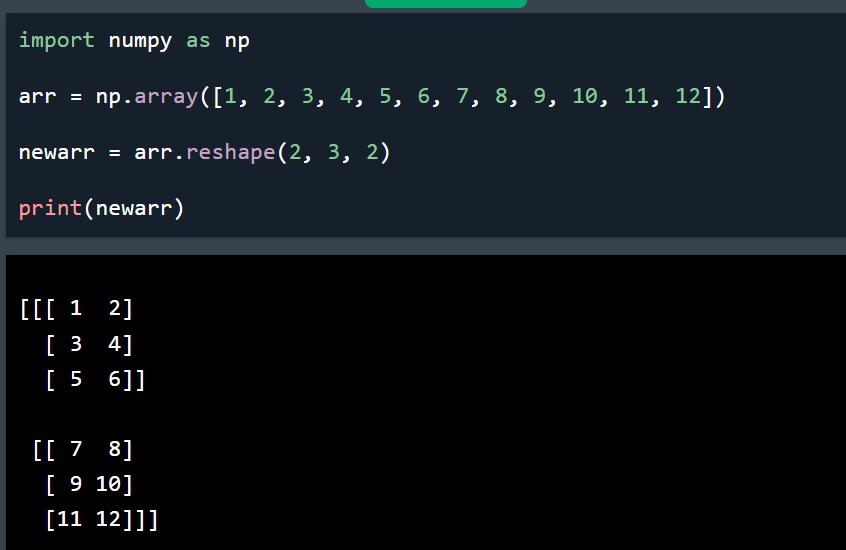


**Reshape From 1-D to 3-D**

**Example**

Convert the following 1-D array with 12 elements into a 3-D array.

The outermost dimension will have 2 arrays that contains 3 arrays, each with 2 elements:



**Generating Special Arrays**

