

NumPy Standard Data Types

NumPy arrays contain values of a single type, so it is important to have detailed knowledge of those types and their limitations. Because NumPy is built in C, the types will be familiar to users of C, Fortran, and other related languages.

The standard NumPy data types are listed in [Table 2-1](#). Note that when constructing an array, you can specify them using a string:

```
np.zeros(10, dtype='int16')
```

Or using the associated NumPy object:

```
np.zeros(10, dtype=np.int16)
```

Table 2-1. Standard NumPy data types

Data type	Description
bool_	Boolean (True or False) stored as a byte
int_	Default integer type (same as C long; normally either int64 or int32)
intc	Identical to C int (normally int32 or int64)
intp	Integer used for indexing (same as C ssize_t; normally either int32 or int64)
int8	Byte (–128 to 127)
int16	Integer (–32768 to 32767)
int32	Integer (–2147483648 to 2147483647)
int64	Integer (–9223372036854775808 to 9223372036854775807)
uint8	Unsigned integer (0 to 255)
uint16	Unsigned integer (0 to 65535)
uint32	Unsigned integer (0 to 4294967295)
uint64	Unsigned integer (0 to 18446744073709551615)
float_	Shorthand for float64
float16	Half-precision float: sign bit, 5 bits exponent, 10 bits mantissa
float32	Single-precision float: sign bit, 8 bits exponent, 23 bits mantissa
float64	Double-precision float: sign bit, 11 bits exponent, 52 bits mantissa
complex_	Shorthand for complex128
complex64	Complex number, represented by two 32-bit floats
complex128	Complex number, represented by two 64-bit floats

More advanced type specification is possible, such as specifying big or little endian numbers; for more information, refer to the [NumPy documentation](#). NumPy also supports compound data types, which will be covered in [“Structured Data: NumPy’s Structured Arrays” on page 92](#).

The Basics of NumPy Arrays

Data manipulation in Python is nearly synonymous with NumPy array manipulation: even newer tools like Pandas ([Chapter 3](#)) are built around the NumPy array. This section will present several examples using NumPy array manipulation to access data and subarrays, and to split, reshape, and join the arrays. While the types of operations shown here may seem a bit dry and pedantic, they comprise the building blocks of many other examples used throughout the book. Get to know them well!

We'll cover a few categories of basic array manipulations here:

Attributes of arrays

Determining the size, shape, memory consumption, and data types of arrays

Indexing of arrays

Getting and setting the value of individual array elements

Slicing of arrays

Getting and setting smaller subarrays within a larger array

Reshaping of arrays

Changing the shape of a given array

Joining and splitting of arrays

Combining multiple arrays into one, and splitting one array into many

NumPy Array Attributes

First let's discuss some useful array attributes. We'll start by defining three random arrays: a one-dimensional, two-dimensional, and three-dimensional array. We'll use NumPy's random number generator, which we will *seed* with a set value in order to ensure that the same random arrays are generated each time this code is run:

```
In[1]: import numpy as np
       np.random.seed(0) # seed for reproducibility

       x1 = np.random.randint(10, size=6) # One-dimensional array
       x2 = np.random.randint(10, size=(3, 4)) # Two-dimensional array
       x3 = np.random.randint(10, size=(3, 4, 5)) # Three-dimensional array
```

Each array has attributes `ndim` (the number of dimensions), `shape` (the size of each dimension), and `size` (the total size of the array):

```
In[2]: print("x3 ndim: ", x3.ndim)
       print("x3 shape:", x3.shape)
       print("x3 size: ", x3.size)

x3 ndim: 3
x3 shape: (3, 4, 5)
x3 size: 60
```