NumPy - Data Types

NumPy supports a much greater variety of numerical types than Python does. The following table shows different scalar data types defined in NumPy.

Sr.No.	Data Types & Description
1	**bool_** Boolean (True or False) stored as a byte
2	**int_** Default integer type (same as C long; normally either int64 or int32)
3	**intc** Identical to C int (normally int32 or int64)
4	**intp** Integer used for indexing (same as C ssize_t; normally either int32 or int64)
5	**int8** Byte (-128 to 127)
6	**int16** Integer (-32768 to 32767)
7	**int32** Integer (-2147483648 to 2147483647)

8	**int64** Integer (-9223372036854775808 to 9223372036854775807)
9	**uint8** Unsigned integer (0 to 255)
10	**uint16** Unsigned integer (0 to 65535)
11	**uint32** Unsigned integer (0 to 4294967295)
12	**uint64** Unsigned integer (0 to 18446744073709551615)
13	**float_** Shorthand for float64
14	**float16** Half precision float: sign bit, 5 bits exponent, 10 bits mantissa
15	**float32** Single precision float: sign bit, 8 bits exponent, 23 bits mantissa
16	**float64** Double precision float: sign bit, 11 bits exponent, 52 bits mantissa
17	**complex_** Shorthand for complex128

18	**complex64** Complex number, represented by two 32-bit floats (real and imaginary components)
19	**complex128** Complex number, represented by two 64-bit floats (real and imaginary components)

NumPy numerical types are instances of dtype (data-type) objects, each having unique characteristics. The dtypes are available as np.bool_, np.float32, etc.

Data Type Objects (dtype)

A data type object describes interpretation of fixed block of memory corresponding to an array, depending on the following aspects –

- Type of data (integer, float or Python object)
- Size of data
- Byte order (little-endian or big-endian)
- In case of structured type, the names of fields, data type of each field and part of the memory block taken by each field.
- If data type is a subarray, its shape and data type

The byte order is decided by prefixing '<' or '>' to data type. '<' means that encoding is little-endian (least significant is stored in smallest address). '>' means that encoding is big-endian (most significant byte is stored in smallest address).

A dtype object is constructed using the following syntax –

```
numpy.dtype(object, align, copy)
```

The parameters are –

- **Object** – To be converted to data type object
- **Align** – If true, adds padding to the field to make it similar to C-struct

- **Copy** – Makes a new copy of dtype object. If false, the result is reference to builtin data type object

Example 1

Live Demo

```
# using array-scalar type

import numpy as np

dt = np.dtype(np.int32)

print dt
```

The output is as follows –

```
int32
```

Example 2

Live Demo

```
#int8, int16, int32, int64 can be replaced by equivalent string 'i1', 'i2','i4', etc.

import numpy as np


dt = np.dtype('i4')

print dt
```

The output is as follows –

```
int32
```

Example 3

Live Demo

```
# using endian notation

import numpy as np

dt = np.dtype('>i4')

print dt
```

The output is as follows –

```
>i4
```

The following examples show the use of structured data type. Here, the field name and the corresponding scalar data type is to be declared.

Example 4

Live Demo

```
# first create structured data type

import numpy as np

dt = np.dtype([('age',np.int8)])

print dt
```

The output is as follows –

```
[('age', 'i1')]
```

Example 5

Live Demo

```
# now apply it to ndarray object

import numpy as np


dt = np.dtype([('age',np.int8)])

a = np.array([(10,),(20,),(30,)], dtype = dt)

print a
```

The output is as follows –

```
[(10,) (20,) (30,)]
```

Example 6

Live Demo

```
# file name can be used to access content of age column

import numpy as np


dt = np.dtype([('age',np.int8)])
```

```
a = np.array([(10,),(20,),(30,)], dtype = dt)

print a['age']
```

The output is as follows –

```
[10 20 30]
```

Example 7

The following examples define a structured data type called **student** with a string field 'name', an **integer field** 'age' and a **float field** 'marks'. This dtype is applied to ndarray object.

Live Demo

```
import numpy as np

student = np.dtype([('name','S20'), ('age', 'i1'), ('marks', 'f4')])

print student
```

The output is as follows –

```
[('name', 'S20'), ('age', 'i1'), ('marks', '<f4')])
```

Example 8

Live Demo

```
import numpy as np


student = np.dtype([('name','S20'), ('age', 'i1'), ('marks', 'f4')])

a = np.array([('abc', 21, 50),('xyz', 18, 75)], dtype = student)

print a
```

The output is as follows –

```
[('abc', 21, 50.0), ('xyz', 18, 75.0)]
```

Each built-in data type has a character code that uniquely identifies it.

- **'b'** – boolean
- **'i'** – (signed) integer
- **'u'** – unsigned integer

- **'f'** – floating-point
- **'c'** – complex-floating point
- **'m'** – timedelta
- **'M'** – datetime
- **'O'** – (Python) objects
- **'S', 'a'** – (byte-)string
- **'U'** – Unicode
- **'V'** – raw data (void)