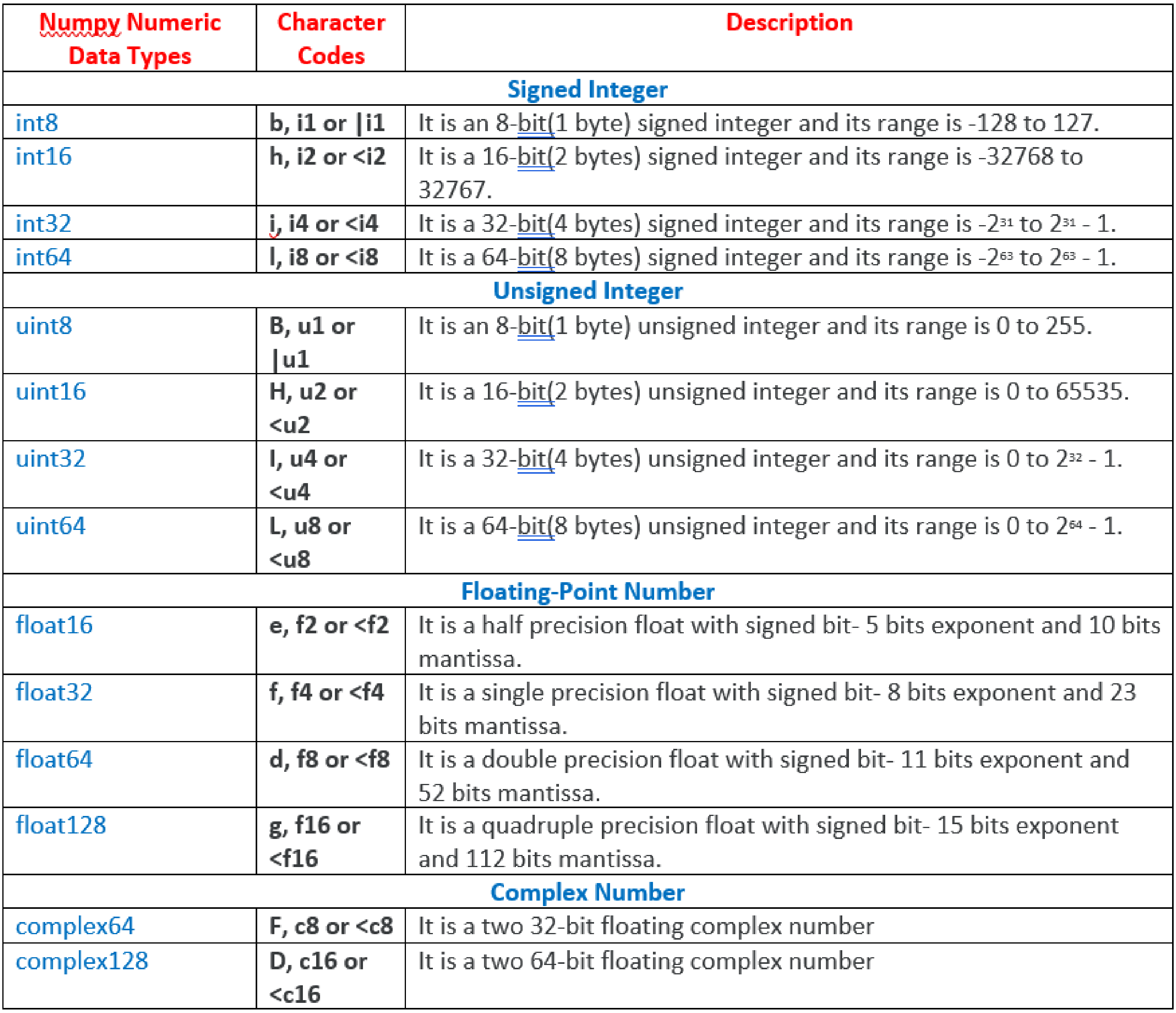
Data Types in NumPy

- The ndarray needs to interpret a chunk of memory as a particular type of data.
- The data type or dtype is a special object containing the information (or metadata, data about data).
- There are three broad categories under which NumPy data types are categorized:

> 1. Numpy Numeric Types
> 2. Numpy String Types
> 3. Numpy Boolean Types

1. Numpy Numeric Data Types

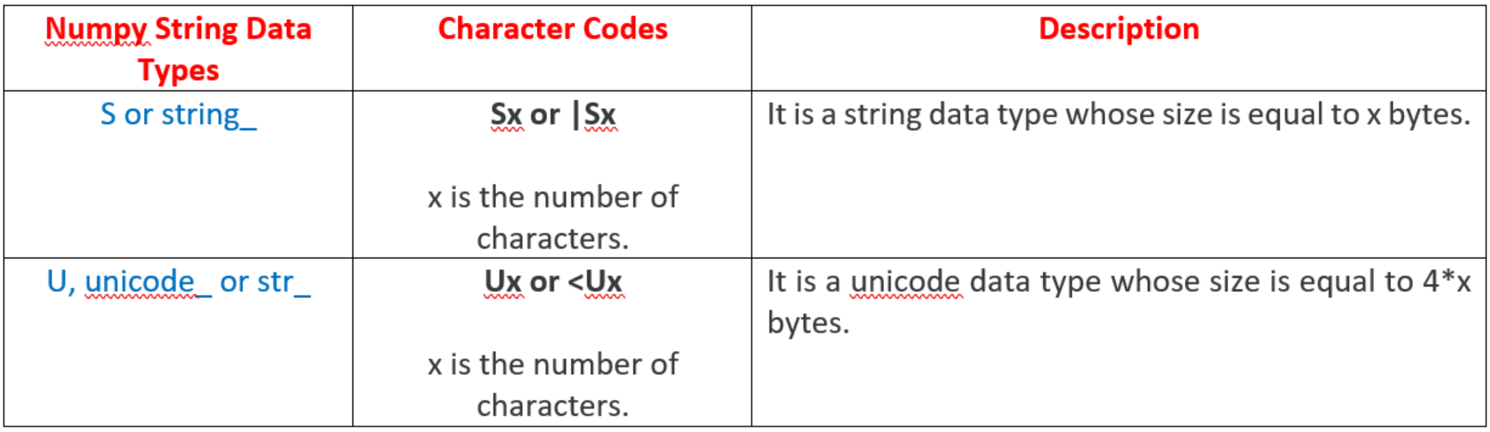
Numeric types include signed and unsigned integer, floating-point numbers, and complex numbers.



Numpy Numeric Data Types	Character Codes	Description
Signed Integer		
int8	**b, i1 or |i1**	It is an 8-bit(1 byte) signed integer and its range is -128 to 127.
int16	**h, i2 or <i2**	It is a 16-bit(2 bytes) signed integer and its range is -32768 to 32767.
int32	**i, i4 or <i4**	It is a 32-bit(4 bytes) signed integer and its range is -2^{31} to 2^{31} - 1.
int64	**l, i8 or <i8**	It is a 64-bit(8 bytes) signed integer and its range is -2^{63} to 2^{63} - 1.
Unsigned Integer		
uint8	**B, u1 or |u1**	It is an 8-bit(1 byte) unsigned integer and its range is 0 to 255.
uint16	**H, u2 or <u2**	It is a 16-bit(2 bytes) unsigned integer and its range is 0 to 65535.
uint32	**I, u4 or <u4**	It is a 32-bit(4 bytes) unsigned integer and its range is 0 to 2^{32} - 1.
uint64	**L, u8 or <u8**	It is a 64-bit(8 bytes) unsigned integer and its range is 0 to 2^{64} - 1.
Floating-Point Number		
float16	**e, f2 or <f2**	It is a half precision float with signed bit- 5 bits exponent and 10 bits mantissa.
float32	**f, f4 or <f4**	It is a single precision float with signed bit- 8 bits exponent and 23 bits mantissa.
float64	**d, f8 or <f8**	It is a double precision float with signed bit- 11 bits exponent and 52 bits mantissa.
float128	**g, f16 or <f16**	It is a quadruple precision float with signed bit- 15 bits exponent and 112 bits mantissa.
Complex Number		
complex64	**F, c8 or <c8**	It is a two 32-bit floating complex number
complex128	**D, c16 or <c16**	It is a two 64-bit floating complex number

2. Numpy String Data Types

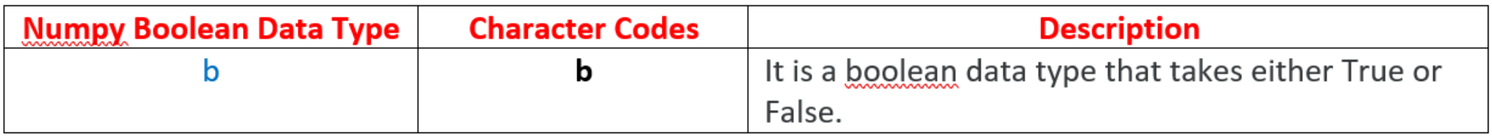
String types include String and Unicode.



Numpy String Data Types	Character Codes	Description
S or string_	**Sx or |Sx** x is the number of characters.	It is a string data type whose size is equal to x bytes.
U, unicode_ or str_	**Ux or <Ux** x is the number of characters.	It is a unicode data type whose size is equal to 4*x bytes.

3. Numpy Boolean Data Types

It represents True or False.



Numpy Boolean Data Type	Character Codes	Description
b	**b**	It is a boolean data type that takes either True or False.

Range of NumPy Data Types

1. bool_ – It is used to return Boolean true or false values.
2. int_ – It is the default integer type (int64 or int32)
3. int8 – It is for assigning 8-bit integer value (-128 to 127)
4. int16 – It is for assigning 16-bit integer value (-32768 to 32767)
5. int32 – It is for assigning 32-bit integer value (-2147483648 to 2147483647)
6. int64 – It is for assigning 64-bit integer value (-9223372036854775808 to 9223372036854775807)
7. uint8 – It is for assigning unsigned 8-bit integer value (0 to 255)
8. uint16 – It is for assigning unsigned 16-bit integer value (0 to 65535)
9. uint32 – It is for assigning unsigned 32-bit integer value (0 to 4294967295)
10. uint64 – It is for assigning unsigned 64-bit integer value (0 to 18446744073709551615)
11. float_ – It is to assign float values.
12. float16 – It is for half precision float values.
13. float32 – It is for single-precision float values.
14. float64 – It is for double-precision float values.
15. complex_ – It is to assign complex values.
16. complex64 – It is to represent two 32-bit float complex values (real and imaginary)

17. complex128 – It is to represent two 64-bit float complex values (real and imaginary)

Checking the Data Type of an Array

- The NumPy array object has a property called **dtype** that returns the data type of the array.
- 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

Syntax:

numpy.dtype(obj, align=False, copy=False)

Parameters:

> 1.**obj-** It represents the object which is to be converted to the data type.

> 2.**align bool (optional)-** It can be set to any boolean value. If true, then it adds extra padding.

> 3.**copy bool (optional)-** It creates another copy of the dtype object.

- We use the array() function to create arrays, the function can take dtype as an argument that helps define the define the data type of the array elements.

```
#Get the data type of an array object:
import numpy as np

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

print(arr.dtype)
```

```
int64
```

```
#Get the data type of an array containing strings:
import numpy as np

arr = np.array(['apple', 'banana', 'cherry'])

print(arr.dtype)
```

```
<U6
```

```
import numpy as np
```

```
new_array = np.array(['Micheal', 'John', 'George'])
print(new_array.dtype)
```

```
<U7
```

```
import numpy as np

new_array = np.array([89.9, 17.9, 45.1])
print(new_array.dtype)
```

```
float64
```

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.

For i, u, f, S and U we can define size as well.

```
#Create an array with data type 4 bytes integer:
import numpy as np

arr = np.array([1, 2, 3, 4], dtype='i4')

print(arr)
print(arr.dtype)
```

```
[1 2 3 4]
int32
```

What if a Value Can Not Be Converted?

> If a type is given in which elements can't be casted then NumPy will raise a ValueError.

> **ValueError:** In Python ValueError is raised when the type of passed argument to a function is unexpected/incorrect.

```
#A non integer string like 'a' can not be converted to integer (will raise an error):
import numpy as np

arr = np.array(['a', '2', '3'], dtype='i')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
```

Changing Data Type of an Existing Array

astype()

- 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.

- Change the data type of array from int64 to float64.

```
import numpy as np

x = np.array([12, 24, 36, 48])
print('Array a:', x)
print('Data type of array a:', x.dtype)

x = x.astype('float64')
print('Data type of array a after calling astype():', x.dtype)
print('Array a:', x)
```

```
import numpy as np

new_arr = np.array([78.3, 17.5, 15.2, 17.1, 19.2])

new_result = new_arr.astype('i')

print(new_result)
```

```
#Change data type from float to integer by using character code 'i' as parameter value:

import numpy as np

a = np.array([1.1, 2.1, 3.1])

newarr = a.astype('i')

print(newarr)
print(newarr.dtype)
```

```
#Change data type from float to integer by using 'int' as parameter value:

import numpy as np

arr = np.array([1.1, 2.1, 3.1])
```

```
newarr = arr.astype(int)

print(newarr)
print(newarr.dtype)
```

```
#Change data type from integer to boolean:
import numpy as np

arr = np.array([1, 0, 3])

newarr = arr.astype(bool)

print(newarr)
print(newarr.dtype)
```

```
arr = np.array([1, 2, 3, 4, 5])

print(arr.dtype)

float_arr = arr.astype(np.float64)

print(float_arr.dtype)
```

NumPy array with mixed data types

```
import numpy as np

res = np.array([("john", 15, 14), ("George", 13, 21)], dtype='|S4, i4, i4')

print(res)
```

```
import numpy as np

new_arr= np.array([('pos', (78, 15, 28)), ('sop', (10, 26, 87))], dtype='3a, 3i')

print(new_arr)
```

```
import numpy as np

new_array1 = np.array([89, 45, 21, 98], dtype='S')
new_array2 = np.array([91, 22, 87, 65], dtype='i4')

print(new_array1.dtype)
print(new_array2.dtype)
```

```
import numpy as np

new_arr = np.dtype('uint8')
print (new_arr)
new_val = np.array([87, 45, 12, 98], dtype='uint8')
new_val = np.array([87, 45, 12, 98], dtype='uint8')

print(new_val.dtype)
```

```
uint8
uint8
```

✓ 0s completed at 13:39 ● ✕