Data types

What is a data type

The data type of an array gives `numpy` some information on how to deal with this array. Most common data types are:

- `int_`
- `float_`
- `str_`
- `bool_`

These types are a bit different from the ones of Python. They can be accessed using the `dtype` attribute (whereas Python type is given by `type(...)`)

A numpy array is always of type `numpy.ndarray`, but the dtype depends on its content:

In [1]:

```python
import numpy as np
arr = np.array([1, 2, 3])
print(type(arr))
print(arr.dtype)
```

```
<class 'numpy.ndarray'>
int64
```

Use data types

Automatically assigned data type

In most cases, numpy will choose a dtype automatically. The chosen dtype is the one compatible with all the elements of the array.

```
In [2]: np.array([1, 2, 3]).dtype
```

```
Out[2]:   dtype('int64')
```

If one of the integer has a '.', Python thinks it's a float (even though decimal part is 0):

```
In [3]: np.array([1, 2, 3.]).dtype
```

```
Out[3]:   dtype('float64')
```

If some non-numeric values exist, the dtype is non-numeric and mathematical operations are impossible:

In [4]:

```
arr = np.array(['azerty', 45, 98])
print(arr.dtype)
arr.sum()
```

```
<U21
```

```
---------------------------------------------------------------------------
UFuncTypeError                            Traceback (most recent call last)
Cell In[4], line 3
      1 arr = np.array(['azerty', 45, 98])
      2 print(arr.dtype)
----> 3 arr.sum()

File ~/Python/3.12/lib/python3.12/site-packages/numpy/core/_methods.py:49, i
n _sum(a, axis, dtype, out, keepdims, initial, where)
     47 def _sum(a, axis=None, dtype=None, out=None, keepdims=False,
     48          initial=_NoValue, where=True):
---> 49     return umr_sum(a, axis, dtype, out, keepdims, initial, where)

UFuncTypeError: ufunc 'add' did not contain a loop with signature matching t
ypes (dtype('<U21'), dtype('<U21')) -> None
```

Change data type

One can change the data type using `astype`, by specifying one of these:

- a numpy dtype: object or string
- a Python type for which equivalent dtype exists in `numpy`

In [5]:

```python
arr = np.array([1, 2, 3])
print(arr.dtype)
arr = arr.astype(np.float_)   # numpy dtype, specified as an object
print(arr.dtype)
```

```
int64
float64
```

In [6]:

```python
arr = arr.astype(int)         # python type
print(arr.dtype)
```

```
int64
```

In [7]:

```python
arr = arr.astype('complex')   # numpy dtype, specified as a string
print(arr.dtype)
```

```
complex128
```

Modifying the *dtype* can change the data:

```
In [8]: np.array([1, 2, 3.65]).astype(int)
```

```
Out[8]: array([1, 2, 3])
```

casting is sometimes possible, for instance regarding boolean values:

```
In [9]: np.array([1, 2, 0]).astype(bool)
```

```
Out[9]:   array([ True,  True, False])
```

Working with `nan`

Definition

`nan` means 'not a number'. A `nan` value (`np.nan`) is used to describe:

- a missing or unknown value
- the result of an impossible mathematical operation

You must never deal with `np.nan` using equality tests (`==`): the preferred way is to use dedicated functions of `numpy`.

`nan` propagation

As `np.inf` (infinite), `nan` values propagate in mathematical operations:

In [10]:

```
arr = np.arange(16).reshape((4,4)).astype(float)
arr[1, 2] = np.nan
arr
```

Out[10]:

```
array([[ 0.,  1.,  2.,  3.],
       [ 4.,  5., nan,  7.],
       [ 8.,  9., 10., 11.],
       [12., 13., 14., 15.]])
```

In [11]:

```
arr.sum(axis=1)
```

Out[11]:

```
array([ 6., nan, 38., 54.])
```

`numpy.isnan()` returns a boolean describing which value is a nan. With `numpy.where` replacement is possible:

```
In [12]:
cond = np.isnan(arr)
arr[cond] = 0
arr.sum(axis=1)
```

```
Out[12]:
array([ 6., 16., 38., 54.])
```