Programming in Python[1]

Mattia Monga

Dip. di Informatica
Università degli Studi di Milano, Italia
`mattia.monga@unimi.it`

Academic year 2021/22, II semester

PyQB

Monga

Third-party
libraries

NumPy

ndarr
ay

Creation
Indexing
Vectorization
Array operations

Lecture XIII: Using Third-party libraries

Third-party libraries

Python is "sold" *batteries included* (with many useful built-in libraries). Moreover, like many modern programming environments, it has standard **online package directories** that list libraries produced by independent developers.
`https://pypi.org/`
The Python package index currently lists almost 300K libraries!

Installing a library

The details are explained here: `https://packaging.python.org/tutorials/installing-packages/`

- In most cases it is very easy, the `pip` program does all the magic
- It is **very** important to understand the difference between a system-wide and a project-specific installation.

System-wide vs. Project-specific

PyQB

Monga

Third-party libraries

NumPy

ndarray

Creation
Indexing
Vectorization
Array operations

If you don't take special precautions, a package is installed in a way that makes it available to your Python system: every Python interpreter you launch sees them.

- In many cases, this is **not** what you want
- **Different projects/programs might depend on different versions of the libraries**
- Libraries themselves depend on other libraries, you want to understand exactly which packages your program is using in order to **reproduce** the settings on other machines

77

Virtual environments

PyQB

Monga

Third-party libraries

NumPy

ndarray

Creation
Indexing
Vectorization
Array operations

Python provides the idea of virtual development environments (`venv`)

- You can create one with: `python -m venv CHOOSE_A_NAME`
- You must activate it (syntax depends on your OS): `CHOOSE_A_NAME\Scripts \activate`
- In an active virtual environment all the installation are **confined** to it
- You can get the list of installed packages with `pip freeze`

78

Simplified venv administration

PyQB

Monga

Third-party libraries

NumPy

ndarray

Creation
Indexing
Vectorization
Array operations

Virtual environments are key to avoid messing up your system. Many tools simplify their administration.

- `pipenv` (my preferred one, we will use this)
- `poetry` (similar to `pipenv`, currently less popular, but it has a better dependency control, a bit more complex)
- `conda` (uses its own package index, great flexibility and complexity, manage different python versions)

79

Virtual environments caveats

PyQB

Monga

Third-party libraries

NumPy

ndarray

Creation
Indexing
Vectorization
Array operations

When you are working in a Python virtual environment, remember to launch **all** your development tools "inside" the virtual space.
For example, to use IDLE don't click on the main application launcher, instead: `python -m idlelib`.

80

NumPy

PyQB

Monga

Third-party
libraries

NumPy
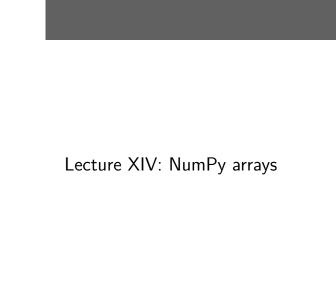
ndarr⌋
ay

Creation
Indexing
Vectorization
Array operations

`NumPy` is a third-party library very popular for
scientific/numerical programming (https://numpy.org/).

- Features familiar to `matlab`, R, Julia programmers
- The key data structure is the array
  - 1-dimension arrays: vectors
  - 2-dimension arrays: matrices
  - n-dimension arrays

In some languages array is more or less synonym of list: Python
distinguishes: lists (mutable, arbitrary elements), arrays
(mutable, all elements have the same type), tuples (immutable,
fixed length, arbitrary elements).

PyQB

Monga

Third-party
libraries

NumPy

ndarr⌋
ay

Creation
Indexing
Vectorization
Array operations

Lecture XIV: NumPy arrays

NumPy arrays

PyQB

Monga

Third-party
libraries

NumPy

ndarr⌋
ay

Creation
Indexing
Vectorization
Array operations

The most important data structure in `NumPy` is `ndarray`: a
(usually fixed-size) sequence of same type elements, organized
in one or more dimensions.
https://numpy.org/doc/stable/reference/arrays.
ndarray.html
Implementation is based on byte arrays: accessing an element
(all of the same byte-size) is virtually just the computation of
an 'address'.

Why?

PyQB

Monga

Third-party
libraries

NumPy

ndarr⌋
ay

Creation
Indexing
Vectorization
Array operations

- using NumPy arrays is often more compact, especially
  when there's more than one dimension
- faster than lists when the operation can be vectorized
- (slower than lists when you append elements to the end)
- can be used with element of different types but this is less
  efficient

PyQB

Monga

Third-party
libraries

NumPy

**ndarr
ay**

Creation
Indexing
Vectorization
Array operations

A ndarray has a dtype (the type of elements) and a shape
(the length of the array on each dimensional axis). (Note the
jargon: slightly different from linear algebra)

- Since appending is costly, normally they are pre-allocated
  (zeros, ones, arange, linspace, ...)
- vectorized operations can simplify code (no need for loops)
  and they are faster with big arrays
- vector indexing syntax (similar to R): very convenient (but
  you need to learn something new)

All the elements must have the same size

PyQB

Monga

Third-party
libraries

NumPy

ndarr
ay

Creation
Indexing
Vectorization
Array operations

This is actually a big limitation: the faster access comes with a
price in flexibility.

```
>>> np.array(['','',''])
array(['', '', ''], dtype='<U1')
>>> np.array(['a','bb','ccc'])
array(['a', 'bb', 'ccc'], dtype='<U3')
>>> np.array(['a','bb','cccxxxxxxxxxxxxxxxxxxx'])
array(['a', 'bb', 'cccxxxxxxxxxxxxxxxxxxx'], dtype='<U21')
```

Usually the length is not changed

PyQB

Monga

Third-party
libraries

NumPy

ndarr
ay

Creation
Indexing
Vectorization
Array operations

The best use of arrays is to avoid a change in their length, that
can be costly. Thus, they are normally **preallocated** at
creation:

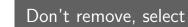- np.array([1,2,3])
- np.zeros(2), np.zeros(2, float), np.ones(2)
- np.empty((2,3)) six not meaningful float values
- np.arange(1, 5) be careful with floats:
  ```
  >>> np.arange(0.4, 0.8, 0.1)
  array([0.4, 0.5, 0.6, 0.7])
  >>> np.arange(0.5, 0.8, 0.1)
  array([0.5, 0.6, 0.7, 0.8])
  ```
- np.linspace(0.5, 0.8, 3) with this the length is
  easier to predict

You can concatenate arrays with np.concatenate (be careful
with the shapes!)

Don't remove, select

PyQB

Monga

Third-party
libraries

NumPy

ndarr
ay

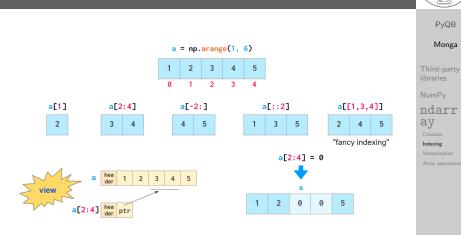Creation
Indexing
Vectorization
Array operations

In general you don't remove elements but select them. Be
careful: if you don't make an explicit **copy** you get a "view"
and possibly side-effects.

```
>>> a = np.ones((2,3))
>>> a
array([[1., 1., 1.],
       [1., 1., 1.]])          >>> x = a[:, 1].copy()
>>> x = a[:, 1]                >>> x[1] = 100
>>> x                         >>> x
array([1., 1.])               array([  0., 100.])
>>> x[0] = 0                  >>> a
>>> x                         array([[1., 0., 1.],
array([0., 1.])                      [1., 1., 1.]])
>>> a
array([[1., 0., 1.],
       [1., 1., 1.]])
```

Indexing is powerful

```
a = np.arange(1, 6)
```



```
a[1]   a[2:4]   a[-2:]   a[::2]   a[[1,3,4]]
```

"fancy indexing"

```
a[2:4] = 0
```

view

Picture from "NumPy Illustrated: The Visual Guide to NumPy", highly
recommended

Indexing is powerful

```
np.any(a > 5)     a[a > 5]     np.all(a > 5)
a[a > 5] = 0
a[(a >= 3) & (a <= 5)] = 0
```

```
&   and
|   or
^   xor
~   not
```

Picture from "NumPy Illustrated: The Visual Guide to NumPy", highly
recommended
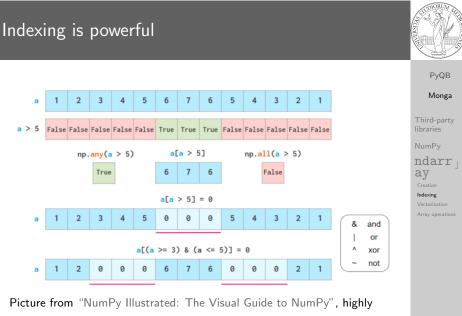
The highest power: vectorization

Most of the basic mathematical function are vectorized: no
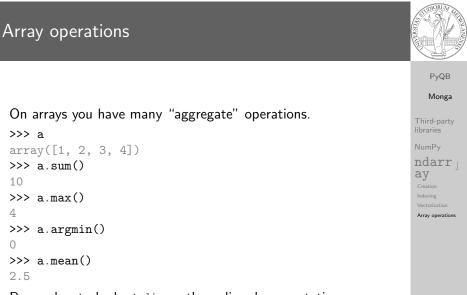need for loops! This is both convenient and faster!

```
>>> a = np.array([1,2,3,4])
>>> a + 1
array([2, 3, 4, 5])
>>> a ** 2
array([ 1,  4,  9, 16])
>>> np.exp(a)
array([ 2.71828183,  7.3890561 , 20.08553692,
↪   54.59815003])
```

Array operations

On arrays you have many "aggregate" operations.

```
>>> a
array([1, 2, 3, 4])
>>> a.sum()
10
>>> a.max()
4
>>> a.argmin()
0
>>> a.mean()
2.5
```

Remember to look at `dir` or the online documentation.