



PyQB

Monga

NumPy  
ndarr  
ay  
Creation

Indexing  
Vectorization  
Array operations

# Programming in Python<sup>1</sup>

Mattia Monga

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

Academic year 2025/26, I semester

<sup>1</sup>© 2025 M. Monga. Creative Commons Attribution — Condividi allo stesso modo 4.0 Internazionale. <http://creativecommons.org/licenses/by-sa/4.0/deed.it>



PyQB

Monga

NumPy  
ndarr  
ay  
Creation

Indexing  
Vectorization  
Array operations

# Lecture XIII: NumPy arrays



PyQB

Monga

NumPy  
ndarr  
ay  
Creation

Indexing  
Vectorization  
Array operations

# NumPy

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

NumPy  
ndarr  
ay  
Creation

Indexing  
Vectorization  
Array operations

# NumPy arrays

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

NumPy

**ndarray**

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

97

# ndarray



PyQB

Monga

NumPy

**ndarray**

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)

98

# All the elements must have the same size



PyQB

Monga

NumPy

**ndarray**

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', 'cccccccccccccccccccc'])
array(['a', 'bb', 'cccccccccccccccccccc'], dtype='<U21')
```

99

# Usually the length is not changed



PyQB

Monga

NumPy

**ndarray**

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!)

100

# Don't remove, select



PyQB

Monga

NumPy  
ndarray  
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]
>>> x
array([1., 1.])
>>> x[0] = 0
>>> x
array([0., 1.])
>>> a
array([[1., 0., 1.],
       [1., 1., 1.]])

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

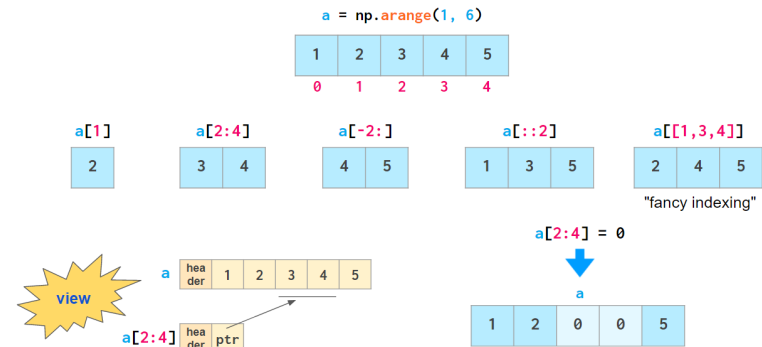
# Indexing is powerful



PyQB

Monga

NumPy  
ndarray  
Creation  
Indexing  
Vectorization  
Array operations



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

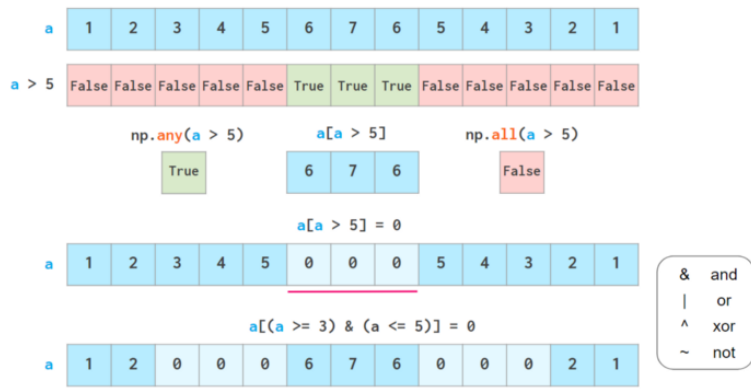
# Indexing is powerful



PyQB

Monga

NumPy  
ndarray  
Creation  
Indexing  
Vectorization  
Array operations



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

# Warning! Assignment works differently from lists



PyQB

Monga

NumPy  
ndarray  
Creation  
Indexing  
Vectorization  
Array operations

```
>>> np = np.array([1,2,3,4,5])
>>> lst = [1,2,3,4,5]
>>> np[2:4] = 0
>>> np
array([1, 2, 0, 0, 5])
>>> lst[2:4] = 0 # Error!
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only assign an iterable
>>> lst[2:4] = [0,0]
>>> lst
[1, 2, 0, 0, 5]
>>> np[2:4] = [0,0]
>>> np[2:4] = [0,0,0] # Error!
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: could not broadcast input array from shape (3,) into
↳ shape (2,)
```

## The highest power: vectorization



PyQB

Monga

NumPy  
ndarr  
ay  
Creation  
Indexing  
Vectorization  
Array operations

Most of the basic mathematical functions 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])
```

105

## Array operations



PyQB

Monga

NumPy  
ndarr  
ay  
Creation  
Indexing  
Vectorization  
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.

106