

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Formation python Orienté Objet et interfaces graphiques

Matthieu Falce

Avril 2022

# Au programme I

Matthieu Falce

Vue d'ensemble

Vue d'ensemble

Langage Python

Langage Python

Programmation Orientée objet (POO)

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bonnes pratiques

Bibliothèque standard

Bibliothèque  
standard

Création de modules

Création de  
modules

Interface graphiques

Interface  
graphiques

Code natif

Python scientifique

# Au programme II

## Code natif

## Python scientifique

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# A propos de moi – Qui suis-je ?

Matthieu Falce

## ► Qui suis-je ?

- Matthieu Falce
- habite à Lille
- ingénieur en bioinformatique (INSA Lyon)

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# A propos de moi – Qui suis-je ?

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

## ► Qu'est ce que j'ai fait ?

- ▶ ingénieur R&D en Interaction Homme-Machine (IHM),  
Inria Lille, équipe Mint puis Mjolnir
- ▶ développeur *fullstack / backend* à FUN-MOOC (France  
Université Numérique)

# A propos de moi – Actuellement

Matthieu Falce

- ▶ entrepreneur salarié dans une SCOP (Société COOPérative) : MFconsulting
  - ▶ conseil en python
  - ▶ rédaction de dossier de financement de l'innovation
  - ▶ formations

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# A propos de moi – Actuellement

Matthieu Falce

- ▶ entrepreneur salarié dans une SCOP (Société COOPérative) : MFconsulting
  - ▶ conseil en python
  - ▶ rédaction de dossier de financement de l'innovation
  - ▶ formations
- ▶ créateur de *Oh Ce Cours Formation*

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# A propos de moi – Actuellement

Matthieu Falce

- ▶ entrepreneur salarié dans une SCOP (Société COOPérative) : MFconsulting
  - ▶ conseil en python
  - ▶ rédaction de dossier de financement de l'innovation
  - ▶ formations
- ▶ créateur de *Oh Ce Cours Formation*

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# A propos de moi – Actuellement

Matthieu Falce

- ▶ entrepreneur salarié dans une SCOP (Société COOPérative) : MFconsulting
  - ▶ conseil en python
  - ▶ rédaction de dossier de financement de l'innovation
  - ▶ formations
- ▶ créateur de *Oh Ce Cours Formation*
- ▶ cofondateur / CTO de ExcellencePriority (site de partage exclusif de petites annonces orienté luxe)

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# A propos de moi – Actuellement

Matthieu Falce

- ▶ entrepreneur salarié dans une SCOP (Société COOPérative) : MFconsulting
  - ▶ conseil en python
  - ▶ rédaction de dossier de financement de l'innovation
  - ▶ formations
- ▶ créateur de *Oh Ce Cours Formation*
- ▶ cofondateur / CTO de ExcellencePriority (site de partage exclusif de petites annonces orienté luxe)
- ▶ coorganisateur de meetups à Lille
  - ▶ python
  - ▶ big data et machine learning

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Où me trouver ?

Matthieu Falce

- ▶ mail: matthieu@falce.net
- ▶ github : ice3
- ▶ twitter : @matthieufalce
- ▶ site: falce.net

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Vue d'ensemble

## Vue d'ensemble

Historique

Philosophie

Python, CPython, ...

Cas d'utilisations de python

Installation

Environnement de développement

## Langage Python

### Programmation Orientée objet (POO)

### Bonnes pratiques

### Bibliothèque standard

### Création de modules

### Interface graphiques

### Code natif

### Python scientifique

# 1- Vue d'ensemble

## 1.1. Historique

Vue d'ensemble

**Historique**

Philosophie

Python, CPython, ...

Cas d'utilisations de python

Installation

Environnement de développement

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Un vieux langage ?

Matthieu Falce

Vue d'ensemble

Historique

Philosophie

Python, CPython, ...

Cas d'utilisations de python

Installation

Environnement de développement

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

- ▶ Créateur (et bdf) : Guido van Rossum
- ▶ 1ère version : 20 février 1991
- ▶ dernière version stable sortie : 3.10.0 (4 octobre 2021)

# Un vieux langage ?

Matthieu Falce

- ▶ Créateur (et bdfl) : Guido van Rossum
- ▶ 1ère version : 20 février 1991
- ▶ dernière version stable sortie : 3.10.0 (4 octobre 2021)



Source: <http://pypl.github.io/PYPL.html>

Vue d'ensemble

Historique

Philosophie

Python, CPython, ...

Cas d'utilisations de python

Installation

Environnement de développement

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Création de modules

Interface graphiques

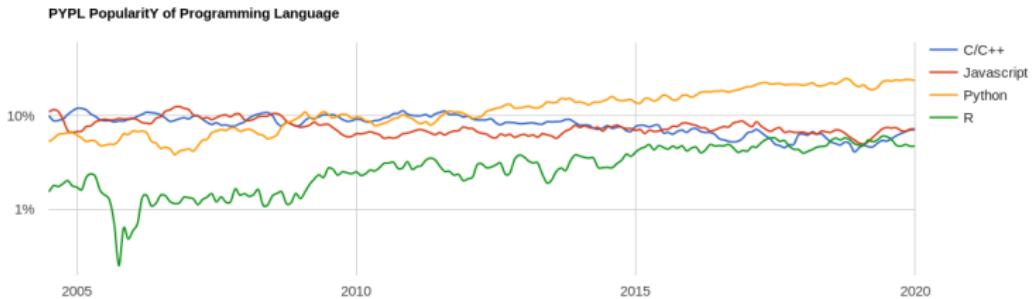
Code natif

Python scientifique

# Un vieux langage ?

Matthieu Falce

- ▶ Créateur (et bdf) : Guido van Rossum
- ▶ 1ère version : 20 février 1991
- ▶ dernière version stable sortie : 3.10.0 (4 octobre 2021)



Source: <http://pypl.github.io/PYPL.html>

Vue d'ensemble

Historique

Philosophie

Python, CPython, ...

Cas d'utilisations de python

Installation

Environnement de développement

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Création de modules

Interface graphiques

Code natif

Python scientifique

# Origine du nom

Matthieu Falce

Le nom n'est pas inspiré du serpent...

Over six years ago, in December 1989, I was looking for a 'hobby' programming project that would keep me occupied during the week around Christmas. My office ... would be closed, but I had a home computer, and not much else on my hands. I decided to write an interpreter for the new scripting language I had been thinking about lately: a descendant of ABC that would appeal to Unix/C hackers. I chose Python as a working title for the project, being in a slightly irreverent mood (and a big fan of Monty Python's Flying Circus)."

---

Guido Van Rossum

[Vue d'ensemble](#)

[Historique](#)

[Philosophie](#)

[Python, CPython, ...](#)

[Cas d'utilisations de python](#)

[Installation](#)

[Environnement de développement](#)

[Langage Python](#)

[Programmation Orientée objet \(POO\)](#)

[Bonne pratiques](#)

[Bibliothèque standard](#)

[Création de modules](#)

[Interface graphiques](#)

[Code natif](#)

[Python scientifique](#)

# Origine du nom

Matthieu Falce

- ▶ Il y a de nombreuses références aux Monty Python dans la communauté, la documentation officielle.
- ▶ Listing d'autres exemples sur Quora
- ▶ Le plus connu est l'utilisation de spam et egg au lieu de foo et bar.

```
def spam():
    eggs = 12
    return eggs

print(spam())
```

[Vue d'ensemble](#)

[Historique](#)

[Philosophie](#)

[Python, CPython, ...](#)

[Cas d'utilisations de python](#)

[Installation](#)

[Environnement de développement](#)

[Langage Python](#)

[Programmation Orientée objet \(POO\)](#)

[Bonne pratiques](#)

[Bibliothèque standard](#)

[Création de modules](#)

[Interface graphiques](#)

[Code natif](#)

[Python scientifique](#)

# Rétrocompatibilité

Matthieu Falce

- ▶ Python est un langage plutôt stable.
- ▶ La syntaxe a globalement peu changé depuis le début.

## Un exemple de code de démo de la version 1.0.0



Vue d'ensemble

Historique

Philosophie

Python, CPython, ...

Cas d'utilisations de python

Installation

Environnement de développement

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

- ▶ Python est un langage plutôt stable.
- ▶ La syntaxe a globalement peu changé depuis le début.

## Un exemple de code de démo de la version 1.0.0

```
from math import sqrt

class complex:

    def __init__(self, re, im):
        self.re = float(re)
        self.im = float(im)

    def __repr__(self):
        return 'complex' + [self.re, self.im]

    def __cmp__(a, b):
        a = a.__abs__()
        b = b.__abs__()
        return (a > b) - (a < b)

    def __float__(self):
        if self.im:
            raise ValueError, 'cannot convert complex to float'
        return float(self.re)

    ...


```

Vue d'ensemble

Historique

Philosophie

Python, CPython, ...

Cas d'utilisations de python

Installation

Environnement de développement

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Python 2 vs Python 3

Matthieu Falce

[Vue d'ensemble](#)

[Historique](#)

[Philosophie](#)

[Python, CPython, ...](#)

[Cas d'utilisations de python](#)

[Installation](#)

[Environnement de développement](#)

[Langage Python](#)

[Programmation Orientée objet \(POO\)](#)

[Bonne pratiques](#)

[Bibliothèque standard](#)

[Création de modules](#)

[Interface graphiques](#)

[Code natif](#)

[Python scientifique](#)

Cependant la compatibilité ascendante a été cassée en passant de python 2 à python 3.

- ▶ réduire les redondances dans le fonctionnement de Python
- ▶ suppression des méthodes obsolètes
- ▶ modification de la grammaire
- ▶ modification des opérations mathématiques
- ▶ beaucoup d'opérations deviennent paresseuses
- ▶ ...

Transition plutôt compliquée :

- ▶ certains développements continuent en python 2
- ▶ nouvelles habitudes
- ▶ grosses bases de code à modifier
- ▶ manque de certaines bibliothèques "essentielles" (non portées)

De nos jours, python 3 est complètement utilisable pour un nouveau projet.

Vue d'ensemble

Historique

Philosophie

Python, CPython, ...

Cas d'utilisations de python

Installation

Environnement de développement

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque standard

Création de modules

Interface graphiques

Code natif

Python scientifique

# Python 2 End Of Life

Matthieu Falce

Fin du support de Python le 1er janvier 2020



[Vue d'ensemble](#)

[Historique](#)

[Philosophie](#)

[Python, CPython, ...](#)

[Cas d'utilisations de python](#)

[Installation](#)

[Environnement de développement](#)

[Langage Python](#)

[Programmation Orientée objet \(POO\)](#)

[Bonne pratiques](#)

[Bibliothèque standard](#)

[Création de modules](#)

[Interface graphiques](#)

[Code natif](#)

[Python scientifique](#)

# Python 2 End Of Life

Matthieu Falce

## Fin du support de Python le 1er janvier 2020

If people find catastrophic security problems in Python 2, or in software written in Python 2, then most volunteers will not help fix them. If you need help with Python 2 software, then many volunteers will not help you, and over time fewer and fewer volunteers will be able to help you. You will lose chances to use good tools because they will only run on Python 3, and you will slow down people who depend on you and work with you. Some of these problems will start on January 1. Other problems will grow over time.

---

<https://www.python.org/doc/sunset-python-2/>

Vue d'ensemble

Historique

Philosophie

Python, CPython, ...

Cas d'utilisations de python

Installation

Environnement de développement

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# 1- Vue d'ensemble

## 1.2. Philosophie

Vue d'ensemble

Historique

**Philosophie**

Python, CPython, ...

Cas d'utilisations de python

Installation

Environnement de développement

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Zen of Python

Matthieu Falce

## Vue d'ensemble

[Historique](#)

**Philosophie**

[Python, CPython, ...](#)

[Cas d'utilisations de python](#)

[Installation](#)

[Environnement de développement](#)

## Langage Python

### Programmation

Orientée objet  
(POO)

### Bonnes pratiques

### Bibliothèque standard

### Création de modules

### Interface graphiques

### Code natif

### Python scientifique

Le langage (et ses utilisateurs) ont des idées plutôt précises de ce qui fait un "bon code".

# Zen of Python (PEP 20<sup>1</sup>)<sup>2</sup>

Matthieu Falce

## import this

The Zen of Python, by Tim Peters

Beautiful is better than ugly.  
Explicit is better than implicit.  
Simple is better than complex.  
Complex is better than complicated.  
Flat is better than nested.  
Sparse is better than dense.  
Readability counts.  
Special cases aren't special enough to break the rules.  
Although practicality beats purity.  
Errors should never pass silently.  
Unless explicitly silenced.  
In the face of ambiguity, refuse the temptation to guess.  
There should be one-- and preferably only one --obvious way to do it.  
Although that way may not be obvious at first unless you're Dutch.  
Now is better than never.  
Although never is often better than \*right\* now.  
If the implementation is hard to explain, it's a bad idea.  
If the implementation is easy to explain, it may be a good idea.  
Namespaces are one honking great idea -- let's do more of those!

[Vue d'ensemble](#)

[Historique](#)

[Philosophie](#)

[Python, CPython, ...](#)

[Cas d'utilisations de python](#)

[Installation](#)

[Environnement de développement](#)

[Langage Python](#)

[Programmation Orientée objet \(POO\)](#)

[Bonnes pratiques](#)

[Bibliothèque standard](#)

[Création de modules](#)

[Interface graphiques](#)

[Code natif](#)

[Python scientifique](#)

---

1.<https://www.python.org/dev/peps/pep-0020/>

2.<https://inventwithpython.com/blog/2018/08/17/the-zend-of-python-explained/>

# 1- Vue d'ensemble

## 1.3. Python, CPython, ...

Vue d'ensemble

Historique

Philosophie

Python, CPython, ...

Cas d'utilisations de python

Installation

Environnement de développement

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# C'est quoi python au final ?

Matthieu Falce

Python peut désigner plusieurs choses quand on n'est pas précis.

- ▶ un langage (la syntaxe et des règles de grammaire)
- ▶ un interpréteur officiel (CPython)
- ▶ des interpréteurs tiers (Jython, IronPython, PyPy, ...)
- ▶ des compilateurs (Cython, Nuitka, ...)

La plupart des gens parlent de CPython avec la grammaire standard quand ils parlent de python.

[Vue d'ensemble](#)

[Historique](#)

[Philosophie](#)

[Python, CPython, ...](#)

[Cas d'utilisations de python](#)

[Installation](#)

[Environnement de développement](#)

[Langage Python](#)

[Programmation Orientée objet \(POO\)](#)

[Bonne pratiques](#)

[Bibliothèque standard](#)

[Création de modules](#)

[Interface graphiques](#)

[Code natif](#)

[Python scientifique](#)

# 1- Vue d'ensemble

## 1.4. Cas d'utilisations de python

Vue d'ensemble

Historique

Philosophie

Python, CPython, ...

**Cas d'utilisations de python**

Scripting

Exemples personnels

Installation

Environnement de développement

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Interpréteur embarqué dans des logiciels

Matthieu Falce

Python sert de langage de script dans de nombreux logiciels :

- ▶ blender <sup>3</sup>
- ▶ qgis <sup>4</sup>
- ▶ autodesk <sup>5</sup>
- ▶ Vim <sup>6</sup>
- ▶ Minecraft <sup>7</sup>
- ▶ ...

---

3.<https://blender.org>

4.<https://qgis.org/en/site/>

5.<https://autodesk.com/>

6.<https://www.vim.org/>

7.<https://minecraft.net/fr-ca/>

Vue d'ensemble

Historique

Philosophie

Python, CPython, ...

Cas d'utilisations de python

Scripting

Exemples personnels

Installation

Environnement de développement

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Exemples personnels

Matthieu Falce

- ▶ électronique / projets *makers*
  - ▶ Artefact (un jeu d'énigmes tangible)<sup>8 9</sup>
  - ▶ *Real Full Stack Python* (du microcontrôleur à la page web en python)<sup>10</sup>
  - ▶ Réalisation de souris / claviers / joysticks / touchpad USB HID
- ▶ Web
  - ▶ EdX<sup>11</sup> / OpenFUN<sup>12</sup>
- ▶ Analyse de données
  - ▶ analyse de séries temporelles
  - ▶ analyse géospatiale

Vue d'ensemble

Historique

Philosophie

Python, CPython, ...

Cas d'utilisations de python

Scripting

Exemples personnels

Installation

Environnement de développement

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

8.<https://bidouilleurslibristes.github.io/Artefact/>

9.[http://falce.net/presentation/Artefact-LillePy/prez\\_artefact.slides.html](http://falce.net/presentation/Artefact-LillePy/prez_artefact.slides.html)

10.[http://falce.net/presentation/IoT\\_Dashboard/index.html](http://falce.net/presentation/IoT_Dashboard/index.html)

11.<https://github.com/edx>

12.<https://github.com/openfun>

# 1- Vue d'ensemble

## 1.5. Installation

Vue d'ensemble

Historique

Philosophie

Python, CPython, ...

Cas d'utilisations de python

**Installation**

Environnement de développement

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Il existe plusieurs distributions python.  
Les plus connues :

- ▶ l'officielle
- ▶ anaconda
- ▶ compilation par Intel
- ▶ ...

Pour commencer et sous Windows, je conseille anaconda

[Vue d'ensemble](#)

[Historique](#)

[Philosophie](#)

[Python, CPython, ...](#)

[Cas d'utilisations de python](#)

[Installation](#)

[Environnement de développement](#)

[Langage Python](#)

[Programmation Orientée objet \(POO\)](#)

[Bonnes pratiques](#)

[Bibliothèque standard](#)

[Création de modules](#)

[Interface graphiques](#)

[Code natif](#)

[Python scientifique](#)

# 1- Vue d'ensemble

## 1.6. Environnement de développement

Vue d'ensemble

Historique

Philosophie

Python, CPython, ...

Cas d'utilisations de python

Installation

Environnement de  
développement

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Pas forcément besoin d'outils spécifiques pour développer (à part un éditeur de texte)...

- ▶ éditeurs de texte + extensions
  - ▶ Microsoft Studio Code
  - ▶ sublime text 3 + anaconda (plugin ST, pas la distribution du dessus...)
  - ▶ ViM / Emacs + plugins
- ▶ IDE
  - ▶ eclipse + mode python
  - ▶ PyCharm

Toujours une faiblesse des outils par rapport à Java par exemple... Mais ça s'améliore.

[Vue d'ensemble](#)

[Historique](#)

[Philosophie](#)

[Python, CPython, ...](#)

[Cas d'utilisations de python](#)

[Installation](#)

[Environnement de développement](#)

[Langage Python](#)

[Programmation Orientée objet \(POO\)](#)

[Bonne pratiques](#)

[Bibliothèque standard](#)

[Création de modules](#)

[Interface graphiques](#)

[Code natif](#)

[Python scientifique](#)

Vue d'ensemble

## Langage Python

- Syntaxe
- Types standards
- Gestion des variables
- Duck typing
- Slicing
- Gestion des fichiers
- Encodage des caractères
- Contrôle de flux
- Fonctions
- Exceptions
- Bibliographie

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Langage Python

## 2- Langage Python

### 2.1. Syntaxe

Vue d'ensemble

Langage Python

#### Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation

Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Votre premier programme Python

Matthieu Falce



A partir de maintenant, toutes les commandes se tapent dans un terminal.

## Comment lancer un programme python ?

```
## En codant directement
## depuis l'interpréteur

python
# Python 3.6.3 (default,
# Oct 3 2017, 21:45:48)
# [GCC 7.2.0] on linux
# Type "help", "copyright",
# "credits" or "license"
# for more information.

print("Bonjour le monde")
```

```
## En lançant un fichier.py

# on écrit dans un fichier
echo \
    "print('Bonjour le monde')" \
> hello.py

# on lance le fichier
python hello.py
```

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodeage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Votre premier programme Python

Matthieu Falce



A partir de maintenant, toutes les commandes se tapent  
dans un terminal.

Comment lancer un programme python ?

Essayez aussi : jupyter notebook et ouvrez votre navigateur sur le lien marqué dans la console

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodeage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation

Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Analyse de la syntaxe

Matthieu Falce

```
# Factorielle en Python

def factorielle(n):
    if n < 2:
        return 1
    else:
        return n * factorielle(n - 1)
```

```
// factorielle en C

int factorielle(int n) {
    if (n < 2) {
        return 1;
    } else {
        return n * factorielle(n - 1);
    }
}
```

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation

Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Analyse de la syntaxe

Matthieu Falce

```
# Factorielle en Python

def factorielle(n):
    """Doc de la fonction.
    Prend un nombre et renvoie n!

    Args:
        n (int): le nombre
        dont on veut la factorielle.

    Returns:
        int. la factorielle
    """
    if n < 2:
        # condition d'arrêt
        return 1
    else:
        return n * factorielle(n - 1)
```

```
// factorielle en C

int factorielle(int n) {
    /* doc de la fonction :
    Prend un nombre et renvoie n!

    Args:
        n (int): le nombre
        dont on veut la factorielle.

    Returns:
        int. la factorielle
    */
    if (n < 2) {
        // condition d'arrêt
        return 1;
    } else {
        return n * factorielle(n - 1);
    }
}
```

Vue d'ensemble

Langage Python

## Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodeage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation

Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Analyse de la syntaxe

Matthieu Falce

- ▶ séparation par l'indentation
- ▶ pas de séparateur de fin de ligne (juste retour chariot)
- ▶ typage dynamique (pas de déclaration des types)
- ▶ mots clefs réservés par le langage

Vue d'ensemble

Langage Python

## Syntaxe

Types standards  
Gestion des variables  
Duck typing  
Slicing  
Gestion des fichiers  
Encodeage des caractères  
Contrôle de flux  
Fonctions  
Exceptions  
Bibliographie

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Analyse de la syntaxe

Matthieu Falce

- ▶ séparation par l'indentation
- ▶ pas de séparateur de fin de ligne (juste retour chariot)
- ▶ typage dynamique (pas de déclaration des types)
- ▶ mots clefs réservés par le langage



Ne jamais mélanger espaces et tabulation dans un fichier.

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation

Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

## 2- Langage Python

### 2.2. Types standards

Vue d'ensemble

Langage Python

Syntaxe

**Types standards**

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodeage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation

Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Types numériques

Matthieu Falce

- ▶ entier (aussi grand que la RAM le peut)
- ▶ flottants
- ▶ type décimal
- ▶ type complexe

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation

Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Types numériques

Matthieu Falce

- ▶ entier (aussi grand que la RAM le peut)
- ▶ flottants
- ▶ type décimal
- ▶ type complexe

```
a = 2 * 2 + 3
print(a)

# http://mortada.net/can-integer-operations-overflow-in-python.html
# https://stackoverflow.com/questions/4581842/python-integer-ranges
a = 2 ** 32 ** 2
print(a) # pas d'overflow sur les grands ints

a = 23134/2
print(a, type(a))

a = 2**3 + 1
print(bin(a)) # avoir la représentation sous forme binaire

c = complex(0, -1)
print(c)
```

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation

Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque standard

Création de modules

Interface graphiques

Code natif

Python scientifique

# Calculs

Matthieu Falce

- ▶ divisions flottantes par défaut
- ▶ ordre des opérateurs mathématiques

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation

Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

- ▶ divisions flottantes par défaut
- ▶ ordre des opérateurs mathématiques

```
import math
import cmath

print("Priorité des opérations")
un = (2 * (3 + 1) - 1) / 7
print(un)

print("calcul sur les nombres réels")
pi_sur_deux = math.pi / 2
print(math.cos(pi_sur_deux))

print("calcul sur les complexes")
c = complex(0, -1)
print(cmath.exp(c * math.pi))
```

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

## On peut manipuler facilement les chaînes :

```
print("Concaténation : ")
debut = "il était"
fin = "une fois"
print(debut + fin)

try:
    print("Attention au typage : ")
    print(debut + 1)
except Exception as e:
    print(e)

print("Fonctions de formatage")
i = 10
print("il y a {} éléments".format(i))
print(f"il y a {i} éléments") # fstring ; python >= 3.6
```

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation

Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Chaînes – performances

Matthieu Falce

Concaténation des chaînes  $\neq$  rapide :

```
print("Attention pour les performances")
print("Les chaines sont immutables")
a = ""
print(id(a))
a += "Autre chose"
print(id(a))
a += "Encore autre chose"
print(id(a))
```

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodeage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation

Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Chaînes – contenu spécial

Matthieu Falce

# Problème d'échappement

```
## "\\" pour échapper un caractère spécial  
## Chemin de fichier windows => C:\Foo\Bar\Baz
```

```
print("C:\\\\F00\\\\Bar\\\\Baz")
```

# raw strings (un seul \)

```
print(r"C:\Foo\Bar\Baz\ ")
```

# Les chaînes en Python 3 sont unicodes

```
print("éàùµ")
```

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation

Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

## Les conteneurs permettent de regrouper plusieurs valeurs

```
# différents types de conteneurs

# ajout d'un élément
liste = [1, 2, 3]
liste.append(4)

humanize = {
    0: "zero",
    1: "un",
}
humanize[2] = "deux"

# un tuple bloque la modification
# du conteneur après sa création
immutable = tuple(liste)

# il ne peut pas y avoir de
# duplication dans les set
pas_elements_double = set([1, 2, 3])
pas_elements_double.add(1)
```

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

## Les conteneurs permettent de regrouper plusieurs valeurs

- ▶ les conteneurs n'ont pas de contraintes de type des objets contenus
- ▶ les conteneurs peuvent avoir une taille infinie
- ▶ chaque type a des propriétés et des complexités (algorithmique) spécifiques
- ▶ les conteneurs sont itérables

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation

Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque standard

Création de modules

Interface graphiques

Code natif

Python scientifique

# Conteneurs

Matthieu Falce

## Les conteneurs permettent de regrouper plusieurs valeurs

```
# récupération d'un élément
liste = [1, 2, 3]
print(liste[0])
print(len(liste))

try:
    print(liste[10])
except Exception as e:
    # les conteneurs sont protégés contre
    # les débordements mémoire
    print(e)

humanize = {
    0: "zero",
    1: "un",
}
print(humanize[0])

# il ne peut pas y avoir de duplication dans les set
ensemble = set([1, 2, 3])
print(1 in ensemble)
print("non" in ensemble)
print(len(ensemble))
```

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

## Les conteneurs permettent de regrouper plusieurs valeurs

```
ensemble = set([1, 2, 3])

try:
    ensemble[2]
except Exception as e:
    # les ensembles ne sont pas ordonnés
    print(e)

humanize = {
    0: "zero",
    1: "un",
}

# on peut récupérer les éléments d'un dictionnaire
print(list(humanize.items()))
print(list(humanize.keys()))
print(len(humanize))

# on peut avoir des valeurs par défaut pour les dico
print(humanize.get("absent", "valeur par default"))

# les tests d'inclusions sont rapides
print(0 in humanize)
print("absent" in humanize)
```

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation

Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Chaînes comme conteneurs

Matthieu Falce

```
print("Transformer un iterable en chaîne :")
elements = (1, 2, 3)
print("-".join([str(i) for i in elements]))\n\nprint("Transformer une chaîne en itérable :")
chaîne = "Il était \n une fois"
print(chaîne.split("\n"))\n\nprint("Les chaînes sont des conteneurs que l'on peut slicer :")
ma_chaîne = "Il était une fois"
print(ma_chaîne[5:10])
```

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation

Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Trouver le type d'une variable

Matthieu Falce

```
a = "une variable"
print(a, type(a))
# une variable <class 'str'>

a = 1
print(a, type(a))
# 1 <class 'int'>

b = 1.1
print(b, type(b))
# 1.1 <class 'float'>

print(a == b, type(a == b))
# False <class 'bool'>

c = complex(1, i)
print(c)
# (1+4j)
```

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodeage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation

Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

## 2- Langage Python

### 2.3. Gestion des variables

Vue d'ensemble

Langage Python

Syntaxe

Types standards

**Gestion des variables**

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation

Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Passage par référence

Matthieu Falce



Python fait le maximum pour abstraire la gestion de mémoire.

Tous les passages se font par référence. Mais certains types sont mutables et pas d'autres.

Vue d'ensemble

Langage Python

Syntaxe

Types standards

**Gestion des variables**

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation

Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque standard

Création de modules

Interface graphiques

Code natif

Python scientifique

# Mutabilité

Matthieu Falce

```
# un entier est un type primitif  
# on a le vrai objet
```

```
a = 2  
b = a  
print(a, b)  
# 2, 2  
  
a = 3  
print(a, b)  
# 3, 2
```

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Mutabilité

Matthieu Falce

*# Quand on utilise des conteneurs, on manipule  
# une référence vers l'objet (+/- un pointeur)*

```
a = [1]
b = a
print(a, b)
#[1] [1]

a[0] = 3
print(a, b)
#[3] [3]
```

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Mutabilité

Matthieu Falce

- ▶ types immutables
  - ▶ tuple
  - ▶ string
  - ▶ int / float
  - ▶ None
- ▶ types mutables
  - ▶ list
  - ▶ dict
  - ▶ set
  - ▶ types personnels
  - ▶ ...

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation

Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Construction des conteneurs

Matthieu Falce

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodeage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

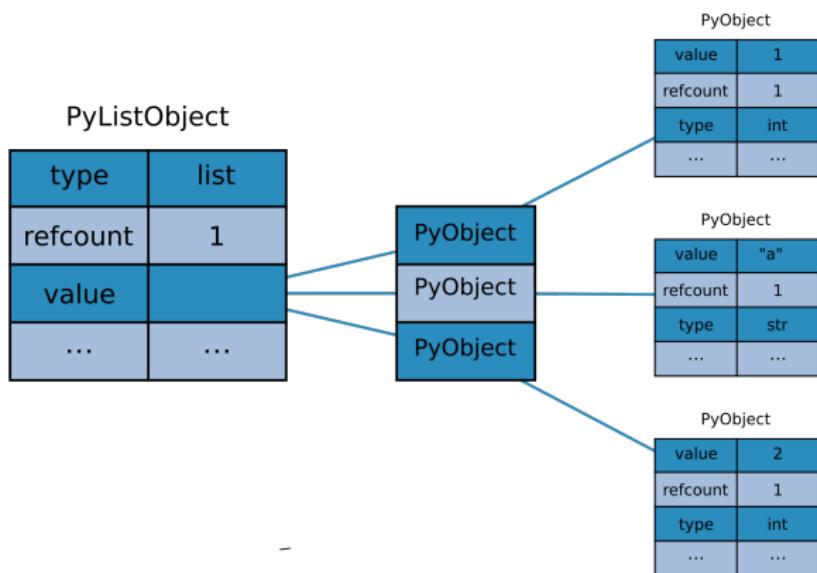
Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

## Structure mémoire d'une liste



# Pour les classes

Matthieu Falce

```
class Exemple():
    a = [1, 2]

exemple1 = Exemple()
exemple2 = Exemple()

print(exemple1.a, exemple2.a) # [1, 2] [1, 2]
print(exemple1.a is exemple2.a) # True

exemple1.a.pop()
print(exemple1.a, exemple2.a) # [1] [1]
print(exemple1.a is exemple2.a) # True

exemple1.a = [10]
print(exemple1.a, exemple2.a) # [10] [1]
print(exemple1.a is exemple2.a) # False
# a est devenu un attribut et non plus une variable de classe
```

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation

Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

## Copier une variable

```
import copy

a = [1, 2]
b = a[:]
print(a is b) # False

a = [1, 2]
b = copy.copy(a)
print(a is b) # False

a = [[1, 2], [3, 4]]
b = copy.copy(a)
print(a[0] is b[0]) # True

c = copy.deepcopy(a)
print(a[0] is c[0]) # False
```

# Cycle de vie

Matthieu Falce

Il y a un *garbage collector* qui s'occupe de supprimer les variables inutilisées.

Il compte les références vers une variable.

Quand il n'y en a plus, il la supprime.

Voilà comment supprimer une référence.

```
a = [1, 2]
```

```
b = a
```

```
del a
```

```
print(b) # [1, 2]
```

```
del b # plus de références
```

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation

Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

## 2- Langage Python

### 2.4. Duck typing

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

**Duck typing**

Slicing

Gestion des fichiers

Encodeage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation

Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Le *duck typing* ?

Matthieu Falce

Si ça ressemble à un canard, si ça nage comme un canard et si ça cancane comme un canard, c'est qu'il s'agit sans doute d'un canard.

---

## Le test du canard

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

**Duck typing**

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation

Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Le *duck typing* ?

Matthieu Falce

A pythonic programming style which determines an object's type by inspection of its method or attribute signature rather than by explicit relationship to some type object ("If it looks like a duck and quacks like a duck, it must be a duck.").

By textualizing interfaces rather than specific types, well-designed code improves its flexibility by allowing polymorphic substitution. Duck-typing avoids tests using `type()` or `isinstance()`. (Note, however, that duck-typing can be complemented with abstract base classes.) Instead, it typically employs `hasattr()` tests or EAFP programming.

---

<https://docs.python.org/3.0/glossary.html>

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation

Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Le *duck typing* ?

Matthieu Falce

Les objets sont contraints selon leur comportement et pas leur type.

- ▶ déterminé à l'exécution plutôt qu'à la compilation
- ▶ l'objet doit posséder une certaine méthode
- ▶ cela rend les paramètres plus génériques
- ▶ on s'intéresse à ce que l'objet peut faire plutôt qu'à ce qu'il est

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation

Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Exemple

Matthieu Falce

```
def prend_premier(conteneur):
    return conteneur[0]

def prend_premier_2(iterable):
    for element in iterable:
        return element

print(prend_premier([1, 2]))
print(prend_premier((1, 2)))
print(prend_premier(open("/etc/hosts"))) # TypeError

print(prend_premier_2([1, 2]))
print(prend_premier_2((1, 2)))
print(prend_premier_2(open("/etc/hosts")))
```

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation

Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque standard

Création de modules

Interface graphiques

Code natif

Python scientifique

Il est classique en python d'utiliser le *duck typing* pour définir des paramètres.

- ▶ **iterable** : on peut appliquer une boucle `for`
- ▶ **callable** : on peut utiliser `x()` dessus
- ▶ **hashable** : peut être passé à la fonction `hash`
- ▶ **indexable** : on peut récupérer un élément précis
- ▶ **slicable** : on peut appliquer une slice
- ▶ ...

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation

Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

## 2- Langage Python

### 2.5. Slicing

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

**Slicing**

Gestion des fichiers

Encodeage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation

Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Slicing

Matthieu Falce

```
a = [x for x in range(100)]
print(a[30:50])
print(a[30:])
print(a[:30])
print(a[1000:2200])

# extended slices
print(a[30:50:10])
print(a[30:50:-1])
print(a[:50:-1])
print(a[30::-1])
print(a[::-1])

# replacement
a[2:5] = [0, 0, 0, 0]
a[::-10] = [0, 0, 0, 0, 0, 0, 0, 0] # ValueError
```

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation

Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

**Slicing**

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

## Explication des slices

### Slicing avec un seul bord

index	0	1	2	3	4	5	6	7
valeur	12	18	15	20	28	10	6	5
liste[:5]						liste[5:]		

### Slicing avec index négatif

index	0	1	2	3	4	5	6	7
valeur	12	18	15	20	28	10	6	5
liste[2:-3]								

### Slicing avec pas

index	0	1	2	3	4	5	6	7
valeur	12	18	15	20	28	10	6	5
liste[::2]				liste[1::2]				

## 2- Langage Python

### 2.6. Gestion des fichiers

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

**Gestion des fichiers**

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation

Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Lecture de fichiers

Matthieu Falce

```
# lecture fichier texte
# par défaut "lecture en mode texte"

## chemin absolu
f_text = open("/tmp/text.txt")

## chemin relatif
f_text = open("../text.txt")

## qu'est-ce que c'est que f_text
# f_text
# <_io.TextIOWrapper name='/tmp/text.txt' mode='r' encoding='UTF-8'>
# c'est une sorte de générateur

text = f_text.read()
text = f_text.read() # texte est vide

# pour lire ligne par ligne
lines = f_text.readlines()
## ou bien
for line in f_text: # équivalent à "in f_text.readline()"
    print(line)
```

Vue d'ensemble

Langage Python

- Syntaxe
- Types standards
- Gestion des variables
- Duck typing
- Slicing

## Gestion des fichiers

- Encodeage des caractères
- Contrôle de flux
- Fonctions
- Exceptions
- Bibliographie

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Lecture de fichiers

Matthieu Falce

```
# lecture binaire

f_data = open("/tmp/image.png", "rb")

## si on lit en mode texte
# f_data = open("/tmp/image.png")
# f_data.read()
# UnicodeDecodeError: 'utf-8' codec can't decode byte 0x89
# in position 0: invalid start byte

# en binaire les fichiers contiennent des bytes strings
magic_number = b'\x89\x50\x4E\x47\x0D\x0A'
(magic_number in f_data) is True
```

Vue d'ensemble

Langage Python

- Syntaxe
- Types standards
- Gestion des variables
- Duck typing
- Slicing

Gestion des fichiers

- Encodeage des caractères
- Contrôle de flux
- Fonctions
- Exceptions
- Bibliographie

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Ecriture de fichiers

Matthieu Falce

```
# ATTENTION : l'écriture d'un fichier l'efface

# on peut écrire toute une chaîne de caractères
f = open("/tmp/text.txt", "w")
f.write("Oh le joli\nmoustique")
f.close()

# ou donner une liste de lignes
f = open("/tmp/text2.txt", "w")
f.writelines(["Oh le joli\n", "moustique.\n\n"])
f.close()

# on peut rajouter des éléments à la suite d'un
# fichier en l'ouvrant différemment
f = open("/tmp/text2.txt", "a")
f.writelines(["Il fait du bruit près de mon oreille\n"])
f.close()

# attention le fichier n'est écrit qu'après l'appel de "flush" ou "close"
```

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation

Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Context Manager – gestionnaire de contexte

Matthieu Falce

```
# plutot que de fermer explicitement les fichiers,  
# on peut dire qu'ils appartiennent à une partie du code particulière
```

```
with open("/tmp/texte.txt") as f_text:  
    for line in f_text:  
        print(line)  
assert f_text.closed is True
```

```
# on peut aussi ouvrir plusieurs fichiers
```

```
with open("./text.txt") as f_rel, open("/tmp/texte.txt") as f_abs:  
    print(f_rel.readlines())  
    print(f_abs.readlines())
```

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation

Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Context Manager – gestionnaire de contexte

Matthieu Falce

```
# plutot que de fermer explicitement les fichiers,  
# on peut dire qu'ils appartiennent à une partie du code particulière  
  
with open("/tmp/texte.txt") as f_text:  
    for line in f_text:  
        print(line)  
assert f_text.closed is True  
  
# on peut aussi ouvrir plusieurs fichiers  
with open("./text.txt") as f_rel, open("/tmp/texte.txt") as f_abs:  
    print(f_rel.readlines())  
    print(f_abs.readlines())
```

Les gestionnaires de contexte sont bien plus génériques que ça. Ils facilitent la gestion de ressources et plus encore.

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation

Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque standard

Création de modules

Interface graphiques

Code natif

Python scientifique

## 2- Langage Python

### 2.7. Encodage des caractères

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

**Encodage des caractères**

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation

Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Encodage des caractères

Matthieu Falce

Vérifiez toujours l'encodage de vos entrées / sorties.  
Spécifiez les si besoin.

```
import sys, locale

# essai réalisé sous windows
print(locale.getpreferredencoding(), sys.getdefaultencoding())
# cp1252, utf-8
print(sys.stdout.encoding, sys.stdin.encoding)
# utf-8, utf-8

# phrases_magic_8_ball est un fichier texte, encodé en UTF8
# il contient des guillements anglais « » qui ne sont pas
# ascii

# on lit le fichier en mode binaire, nous renvoie un bytestring
a = open("./phrases_magic_8_ball.txt", "rb").read()
print(a.decode("utf8"))
# « Essaye plus tard »
# « Pas d'avis »
# ...

# on lit le fichier en précisant l'encoding, nous renvoie de l'unicode
print(open("phrases_magic_8_ball.txt", encoding="utf8").read())
# ...
# « C'est non »
# « Peu probable »
# ...
```

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation

Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

## 2- Langage Python

### 2.8. Contrôle de flux

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

**Contrôle de flux**

Fonctions

Exceptions

Bibliographie

Programmation

Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Boucles

Matthieu Falce

```
# on peut itérer sur un conteneur
ages = [5, 19, 30]
for age in ages:
    print(age)

noms = {"tuple": (), "liste": []}
for nom in noms:
    print(nom, noms[nom])

# on peut créer des "listes" de nombre
for i in range(10):
    print(i)

# il y a aussi while
i = 0
while i != 10:
    i += 1
```

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

- ▶ **break** : sortir de la boucle
  - ▶ **continue** : passer à l'élément suivant
- Vue d'ensemble
- Langage Python
- Syntaxe
  - Types standards
  - Gestion des variables
  - Duck typing
  - Slicing
  - Gestion des fichiers
  - Encodage des caractères
- Contrôle de flux**
- Fonctions
  - Exceptions
  - Bibliographie
- Programmation
- Orientée objet
- (POO)
- Bonnes pratiques
- Bibliothèque standard
- Création de modules
- Interface graphiques
- Code natif
- Python scientifique

# Boucles – “pythonique et non pythonique”

Matthieu Falce



Python a une approche particulière des itérations.  
Il *faut* itérer sur les conteneurs et pas les index.

```
# OUI :o)
elements = [3, 2, 40, 10]
for element in elements:
    print(element)
```

```
# NON :(
elements = [3, 2, 40, 10]
for index in range(len(elements)):
    print(elements[index])
```

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Tuple unpacking

Matthieu Falce

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation

Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

On peut déconstruire des tuples à la volée.

```
premier, deuxieme, *autres, avant_dernier, dernier = range(10)
print("premier", premier)
print("deuxieme", deuxieme)
print("autres", autres)
print("avant_dernier", avant_dernier)
print("dernier", dernier)
```

# \* en compréhension

Matthieu Falce

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodeage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation

Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

On peut construire / manipuler des itérables à la volée

On appelle ça les listes en compréhension ('list comprehension') ou 'dictionnaire comprehension' selon ce que l'on fait.

```
pts = [1, 2, 10, 103]
carres = [p**2 for p in pts]

nbs = range(100)
somme_des_carres_pairs = sum(nb**2 for nb in nbs if nb % 2 == 0)

# marche aussi avec les dictionnaires
noms = ["un", "deux", "trois"]
elements = [1, 2, 3]
humanize = {e: n for e, n in zip(elements, noms)}
```

# Tests et conditions – syntaxe

Matthieu Falce

On utilise `if`, `elif`, `else` pour tester une variable

```
a = 3
```

```
if a == 1:  
    print("ah")  
elif a == 2:  
    print("je le savais")  
else:  
    print(":('")
```

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation

Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Tests et conditions – booléens

Matthieu Falce

On peut convertir (*caster*) quasiment tous les types en booléens :

```
# les variables ont des évaluations booléennes logiques
a_evaluer = ["salut", [], {}, (), "", 0, (), [[], None, 50]
bools = [bool(element) for element in a_evaluer]

# les évaluations booléennes (et, ou...) sont paresseuses
et = False and 1 / 0
ou = True or 1 / 0
```

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodeage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation

Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Paresse et générateurs

Matthieu Falce

```
# instantannée (évaluation paresseuses)
gen = (i for i in range(100000) if i % 2 == 0)

# plus "long" + utilisation mémoire car provoque l'évaluation
b = list(gen)
b = list(gen) # vide car le générateur est déjà parcouru
print(b)

# on peut chainer les générateurs :
elements = range(100000)
divisible_par_1000 = (e for e in elements if e % 1000 == 0)
multiple_de_43 = (e for e in divisible_par_1000 if e % 43 == 0)
carre = (x ** 2 for x in multiple_de_43)
somme = sum(carre)

# range ne crée pas de liste
# et est plus malin que ce que l'on croit
gros_range = range(20000, int(2e100), 10)
23000 in gros_range
```

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation

Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

## 2- Langage Python

### 2.9. Fonctions

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Gestion des arguments

Gotchas

Higher order functions

Closures

Décorateurs

Exceptions

Bibliographie

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

# Déclaration d'une fonction

Matthieu Falce

```
def ma_fonction(param1):
    param1 * 2

def autre_fonction(param1):
    return param1 * 2

# Les fonctions renvoient toujours quelque chose.
# Si pas de return, elles renvoient "None"
a = ma_fonction(1)
print(a)

b = autre_fonction(2)
print(b)

# Une fonction peut renvoyer plusieurs valeurs,
# de plusieurs types différents
def exemple_return():
    return None, [1, 2, 3]

a = exemple_return()
print(a)
```

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Gestion des arguments

Gotchas

Higher order functions

Closures

Décorateurs

Exceptions

Bibliographie

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

# Déclaration d'une fonction

Matthieu Falce

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Gestion des arguments

Gotchas

Higher order functions

Closures

Décorateurs

Exceptions

Bibliographie

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

```
def exemple_defauts(param1, param2=None):
    """Une fonction peut accepter des paramètres
    nommés et des paramètres par défaut"""
    print(param1, param2)

exemple_defauts() # 1, None
exemple_defauts(1, 2) # 1, 2
exemple_defauts(1, param2=32) # 1, 32

def example_arg_kwargs(param1, *args, **kwargs):
    """Une fonction peut accepter un nombre dynamique
    de paramètres anonymes et nommés.
    Souvent utilisés par les API de bibliothèques.
    Ou quand on ne connaît pas le nombre d'éléments à priori
    """
    print("obligatoire", param1)
    print("liste d'autres arguments anonymes", args)
    print("dict des autres arguments nommés", kwargs)

example_arg_kwargs()
example_arg_kwargs(1)
example_arg_kwargs(1, 2)
example_arg_kwargs(1, 2, param3=3)
```

# Déclaration d'une fonction

Matthieu Falce

Ces trois codes sont globalement équivalents

```
# fonction classique
def addition(x, y):
    return x+y
addition(2, 3)
```

```
# lambda
addition = lambda x, y: x+y
addition(2, 3)
```

```
# fonction anonyme
(lambda x, y: x+y)(2, 3)
```

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Gestion des arguments

Gotchas

Higher order functions

Closures

Décorateurs

Exceptions

Bibliographie

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

# Arguments des fonctions

Matthieu Falce

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

**Gestion des arguments**

Gotchas

Higher order functions

Closures

Décorateurs

Exceptions

Bibliographie

Programmation

Orientée objet

(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

A votre avis, que donnent les fonctions suivantes ?

```
def f(a, b="default"):  
    print("a", a)  
    print("b", b)  
    print("-----")
```

f(1)

f(1, 2)

f(1, 2, 3)

f([1, 2], (3, 4))

# Arguments des fonctions

Matthieu Falce

A votre avis, que donnent les fonctions suivantes ?

```
def g(a, b, *args):
    print("a", a)
    print("b", b)
    print("args", args)
    print("-----")
```

```
g(1, 2)
```

```
g(1, 2, 3)
```

```
## opérateur splat
```

```
liste_example = [1, 2, 3, 4, 5]
g(liste_example)
g(*liste_example)
```

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Gestion des arguments

Gotchas

Higher order functions

Closures

Décorateurs

Exceptions

Bibliographie

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

# Arguments des fonctions

Matthieu Falce

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

**Gestion des arguments**

Gotchas

Higher order functions

Closures

Décorateurs

Exceptions

Bibliographie

Programmation

Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

A votre avis, que donnent les fonctions suivantes ?

```
def f(a, b="default"):  
    print("a", a)  
    print("b", b)  
    print("-----")
```

f(1)

f(1, 2)

f(1, b=2)

f(1, c=2)

# Arguments des fonctions

Matthieu Falce

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

**Gestion des arguments**

Gotchas

Higher order functions

Closures

Décorateurs

Exceptions

Bibliographie

Programmation

Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

A votre avis, que donnent les fonctions suivantes ?

```
def g(a, b, **kwargs):
    print("a", a)
    print("b", b)
    print("kwargs", kwargs)
    print("-----")

g(1, 2)
g(1, 2, c=(3, 4))
g(1, c=3)

## opérateur double splat
dico_example = {"a": 1, "b": 2, "c": 3, "d": 4}
g(dico_example)
g(**dico_example)
```

# Arguments des fonctions

Matthieu Falce

A votre avis, que donnent les fonctions suivantes ?

```
def f(a, b="default", *args, **kwargs):
    print("a", a)
    print("b", b)
    print("args", args)
    print("kwargs", kwargs)
    print("-----")
```

f(1)

f(1, 2)

f(1, b=2)

f(1, 2, 3, b=4, c=5)

f(1, \*[ "c", 3, 4], \*\*{ "d": 5, "e": 6})

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Gestion des arguments

Gotchas

Higher order functions

Closures

Décorateurs

Exceptions

Bibliographie

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

# Liens avec le unpacking

Matthieu Falce

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

**Gestion des arguments**

Gotchas

Higher order functions

Closures

Décorateurs

Exceptions

Bibliographie

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

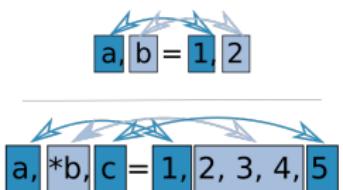
Bibliothèque  
standard

Création de  
modules

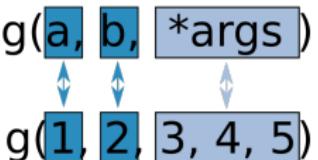
Interface  
graphiques

## Unpacking

Pour les variables



Pour les arguments



# Arguments des fonctions – résumé

Matthieu Falce

- ▶ args et kwargs sont des conventions
- ▶ \* permet de *pack* / *unpack* les listes
- ▶ \*\* permet de *pack* / *unpack* les dictionnaires
- ▶ \* / \*\* sont appelés opérateurs splat

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

**Gestion des arguments**

Gotchas

Higher order functions

Closures

Décorateurs

Exceptions

Bibliographie

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

# Intérêts / limites

Matthieu Falce

## Intérêts :

- ▶ `kwargs.pop` permet de gérer les valeurs de paramètres par défaut
- ▶ intérêt pour les API
  - ▶ manipulation de fonction sans connaître ses paramètres (décorateurs)
  - ▶ fonctions plus ou moins spécialisées (`matplotlib`)
  - ▶ faible couplage entre les fonctions

## Limites :

- ▶ complexifie la documentation / utilisation

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Gestion des arguments

Gotchas

Higher order functions

Closures

Décorateurs

Exceptions

Bibliographie

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

# Problèmes classiques – éléments mutables

14 15

Matthieu Falce

```
# Attention voilà ce qu'il ne faut pas faire.  
# Ne pas mettre d'éléments mutables dans les  
# arguments par défaut  
  
def append_wrong(value, li=[]):  
    """On s'attend à toujours avoir une liste d'un élément."""  
    li.append(value)  
    return li  
  
a = append_wrong(1)  
b = append_wrong(2)  
print(a, b)  
# [1, 2], [1, 2]  
  
# on peut également tester en mettant arg=time.time() pour comprendre  
# le moment de l'évaluation des paramètres  
  
def append_correct(value, li=None):  
    """On met une valeur nulle par défaut et on regarde  
    si elle est renseignée ou pas."""  
    if li is None:  
        li = []  
    li.append(value)  
    return li  
  
a = append_correct(1)  
b = append_correct(2)
```

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodeage des caractères

Contrôle de flux

Fonctions

Gestion des arguments

**Gotchas**

Higher order functions

Closures

Décorateurs

Exceptions

Bibliographie

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

14.<http://docs.python-guide.org/en/latest/writing/gotchas/>

15.<http://blog.notdot.net/2009/11/Python-Gotchas>

# Problèmes classiques – portée des variables

Matthieu Falce

```
variable = 1

def print_variable():
    print(variable)

def modifie_variable():
    variable += 1

def local_variable():
    variable = 2
    return variable

def modifie_variable_ok():
    global variable
    variable += 1

def outer():
    variable = 1
    def inner():
        nonlocal variable
        variable = 2

        print("avant appel inner", variable)
        inner()
        print("apres appel inner", variable)

#### late binding des variables dans les fonctions
variable = 10
print_variable()
variable = 11
print_variable()
```

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodeage des caractères

Contrôle de flux

Fonctions

Gestion des arguments

**Gotchas**

Higher order functions

Closures

Décorateurs

Exceptions

Bibliographie

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodeage des caractères

Contrôle de flux

Fonctions

Gestion des arguments

**Gotchas**

Higher order functions

Closures

Décorateurs

Exceptions

Bibliographie

Programmation

Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

## Espaces de noms

### Espace global

Espace local  
(fonction 1)

a = 1  
b = 2

Espace local  
(fonction 2)

a = 2  
b = 3

a = 4  
b = 5

# Fonctions d'ordre supérieur

Matthieu Falce

- ▶ les fonctions sont des variables comme les autres
- ▶ on peut les passer comme argument à d'autres fonctions
- ▶ on dit que les fonctions sont des *first class citizen*

Les fonctions d'ordre supérieur manipulent d'autres fonctions

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Gestion des arguments

Gotchas

Higher order functions

Closures

Décorateurs

Exceptions

Bibliographie

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

# Fonctions d'ordre supérieur

Matthieu Falce

- ▶ les fonctions sont des variables comme les autres
- ▶ on peut les passer comme argument à d'autres fonctions
- ▶ on dit que les fonctions sont des *first class citizen*

Les fonctions d'ordre supérieur manipulent d'autres fonctions

```
# on veut trier selon la lettre
a = [(1, "d"), (2, "c"), (3, "b"), (4, "a")]
b = sorted(a, key=lambda x: x[1])
```

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Gestion des arguments

Gotchas

Higher order functions

Closures

Décorateurs

Exceptions

Bibliographie

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

# Fonctions comme variables

Matthieu Falce

```
def plus(a, b):
    return a + b

print(ma_fonction, type(ma_fonction))
# <function ma_fonction at 0x7f97716e5620> <class 'function'>

calcul = {
    "plus": plus,
    "moins": lambda x, y: x - y,
    "fois": lambda x, y: x * y,
    "divide": lambda x, y: x / y,
}

calcul["moins"](2, 1)
```

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Gestion des arguments

Gotchas

Higher order functions

Closures

Décorateurs

Exceptions

Bibliographie

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

# Fonctions comme variables

Matthieu Falce

```
def plus(a, b):
    return a + b

print(ma_fonction, type(ma_fonction))
# <function ma_fonction at 0x7f97716e5620> <class 'function'>

calcul = {
    "plus": plus,
    "moins": lambda x, y: x - y,
    "fois": lambda x, y: x * y,
    "divide": lambda x, y: x / y,
}

calcul["moins"](2, 1)
```



Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Gestion des arguments

Gotchas

Higher order functions

Closures

Décorateurs

Exceptions

Bibliographie

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

# Closures / Fermeture

Matthieu Falce

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Gestion des arguments

Gotchas

Higher order functions

Closures

Décorateurs

Exceptions

Bibliographie

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Dans un langage de programmation, une fermeture ou clôture (en anglais : closure) est une fonction accompagnée de l'ensemble des variables non locales qu'elle a capturé.

---

[https://fr.wikipedia.org/wiki/Fermeture\\_\(informatique\)](https://fr.wikipedia.org/wiki/Fermeture_(informatique))

# Closures – Exemples

Matthieu Falce

# on peut déclarer des fonctions locales à d'autres fonctions.

```
def parler():
    # On peut définir une fonction à la volée dans "parler" ...
    def chuchoter(mot="yes"):
        return mot.lower() + "..."

    # ... et l'utiliser immédiatement !
    print(chuchoter())

parler()
# chuchoter n'existe pas dans l'espace global
try:
    print(chuchoter())
except NameError as e:
    print(e)
# output : "name 'chuchoter' is not defined"
```

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Gestion des arguments

Gotchas

Higher order functions

Closures

Décorateurs

Exceptions

Bibliographie

Programmation

Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

# Closures – Exemples

Matthieu Falce

```
def ajoute_avec(nombre):
    def ajouter(autre_nombre):
        return nombre + autre_nombre
    return ajouter
```

```
ajoute_avec_10 = ajoute_avec(10)
print(ajoute_avec_10(5)) # 15
```

```
ajoute_avec_20 = ajoute_avec(20)
print(ajoute_avec_20(2)) # 22
```

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodeage des caractères

Contrôle de flux

Fonctions

Gestion des arguments

Gotchas

Higher order functions

Closures

Décorateurs

Exceptions

Bibliographie

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

# Décorateurs – Syntaxe

Matthieu Falce

Les décorateurs permettent de modifier ou d'injecter un comportement à des fonctions.

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Gestion des arguments

*Gotchas*

Higher order functions

Closures

Décorateurs

Exceptions

Bibliographie

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

# Décorateurs – Syntaxe

Matthieu Falce

La syntaxe avec le @ est un raccourci syntaxique.  
Ces deux façons de faire sont identiques.

```
@decorateur  
def fonction():  
    pass  
  
fonction = decorateur(fonction)
```

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodeage des caractères

Contrôle de flux

Fonctions

Gestion des arguments

Gotchas

Higher order functions

Closures

Décorateurs

Exceptions

Bibliographie

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

# Décorateurs – Syntaxe

Matthieu Falce

```
def ecrit_avant_apres(fonction_a_decorer):
    """Cette fonction prend une fonction qu'elle va
decorer.
"""

def wrapper():
    """Cette fonction entoure l'appel de la fonction
d'origine."""
    print("avant")
    res = fonction_a_decorer()
    print("apres")
    return res

# on retourne la **fonction** wrapper
return wrapper

@ecrit_avant_apres
def test_deco_syntaxe():
    print("dans test deco syntaxe")

print(test_deco_syntaxe())
```

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodeage des caractères

Contrôle de flux

Fonctions

Gestion des arguments

Gotchas

Higher order functions

Closures

Décorateurs

Exceptions

Bibliographie

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

# Décorateurs – Syntaxe

Matthieu Falce

```
# comment accepter des paramètres

def ecrit_avant_apres(fonction_a_decorer):
    """Cette fonction prend une fonction qu'elle va
    decorer.
    """

    def wrapper(*args, **kwargs):
        """Cette fonction entoure l'appel de la fonction
        d'origine."""
        print("avant")
        res = fonction_a_decorer(*args, **kwargs)
        print("pendant", res)
        print("apres")
        return res

    # on retourne la **fonction** wrapper
    return wrapper

@ecrit_avant_apres
def test_deco_syntaxe(a, b, c=0):
    return "resultat test 2", a, b, c

print(test_deco_syntaxe(1, b=2, c=3))
```

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodeage des caractères

Contrôle de flux

Fonctions

Gestion des arguments

Gotchas

Higher order functions

Closures

Décorateurs

Exceptions

Bibliographie

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

# Cas d'usage

Matthieu Falce

- ▶ étendre une fonction qu'on ne peut pas modifier
- ▶ gérer des permissions
- ▶ analyse de performances (mesure du temps passé / mémoire utilisée)
- ▶ mise en cache
- ▶ casting du résultat d'une fonction dans un type
- ▶ ...

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Gestion des arguments

Gotchas

Higher order functions

Closures

Décorateurs

Exceptions

Bibliographie

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

## 2- Langage Python

### 2.10. Exceptions

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation

Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Exceptions

Matthieu Falce

Toujours utiliser une exception précise et bien logguer les erreurs.

Sinon des erreurs peuvent en cacher d'autres.

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation

Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Exceptions

Matthieu Falce

```
try:  
    print("peut lever une exception")  
    raise AssertionError()  
except AssertionError as e:  
    print("    gère l'exception AssertionError")  
except (IndexError, ArithmeticError) as e:  
    print("    gère d'autres exceptions")  
except Exception as e:  
    print("    gère le reste des exceptions")  
else:  
    print("suite logique du code qui peut lever une exception")  
    print("mais qui n'en lève pas lui-même")  
finally:  
    print("appelé quel que soit le parcours d'exception")
```

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation

Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Exceptions

Matthieu Falce

```
# Philosophie en python
# Mieux vaut demander pardon que la permission

def utile(tableau):
    try :
        clef, valeur = tableau[0]
    except IndexError as e:
        clef, valeur = None, None
    else:
        valeur *= 3
    finally:
        return clef, valeur

print(utile([]))
print(utile([1, 2]))
print(utile([(3, 4)]))
```

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation

Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Exceptions

Matthieu Falce

```
def test1():
    try:
        return 1 + "1"
    except TypeError:
        return "exception"

def test2():
    try:
        return 1 + "1"
    except TypeError:
        return "exception"
    finally:
        return "finally"

print(test1())
print(test2())
```

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation

Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Demander pardon plutôt que la permission

Matthieu Falce

Point pythonique : capturer l'exception plutôt que tester si l'action est possible

Easier to ask for forgiveness than permission. This common Python coding style assumes the existence of valid keys or attributes and catches exceptions if the assumption proves false. This clean and fast style is characterized by the presence of many try and except statements. The technique contrasts with the **LBYL** style common to many other languages such as C.

---

Documentation Python

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation

Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque standard

Création de modules

Interface graphiques

Code natif

Python scientifique

## 2- Langage Python

### 2.11. Bibliographie

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation

Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Bibliographie I

Matthieu Falce

## ► Décorateurs

- ▶ <http://sametmax.com/comprendre-les-decorateur-python-pas-a-pas-partie-2/>
- ▶ <http://sametmax.com/le-pattern-observer-en-utilisant-des-decorateurs/>
- ▶ <https://python-3-patterns-idioms-test.readthedocs.io/en/latest/PythonDecorators.html>

## ► Utilisation des astérisques

- ▶ <http://treyhunner.com/2018/10/asterisks-in-python-what-they-are-and-how-to-use-them/>

## ► Variables :

- ▶ <http://sametmax.com/valeurs-et-references-en-python/>
- ▶ <http://sametmax.com/id-none-et-bidouilleries-memoire-en-python/>

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation

Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque standard

Création de modules

Interface graphiques

Code natif

Python scientifique

# Bibliographie II

Matthieu Falce

- ▶ <https://medium.com/@tyastropheus/tricky-python-memory-management-for-mutable-immutable-objects-21507d1e5b95>
- ▶ Exceptions :
  - ▶ <http://sametmax.com/gestion-des-erreurs-en-python/>
  - ▶ <http://sametmax.com/comment-recruter-un-developpeur-python/>
  - ▶ <http://sametmax.com/pourquoi-utiliser-un-mechanisme-d-exceptions/>
- ▶ *Context managers*
  - ▶ <http://sametmax.com/les-context-managers-et-le-mot-cle-with-en-python/>
  - ▶ <https://alysivji.github.io/managing-resources-with-context-managers-pythonic.html>
  - ▶ <http://eigenhombre.com/introduction-to-context-managers-in-python.html>

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodeage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation

Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque standard

Création de modules

Interface graphiques

Code natif

Python scientifique

# Bibliographie III

Matthieu Falce

## ► *Duck Typing*

- ▶ <https://stackoverflow.com/questions/4205130/what-is-duck-typing>
- ▶ <https://hackernoon.com/python-duck-typing-or-automatic-interfaces-73988ec9037f>
- ▶ [https://en.wikipedia.org/wiki/Duck\\_typing](https://en.wikipedia.org/wiki/Duck_typing)
- ▶ <http://sametmax.com/quest-ce-que-le-duck-typing-et-a-quoi-ca-sert/>
- ▶ <http://sametmax.com/les-trucmuchables-en-python/>
- ▶ <https://stackoverflow.com/questions/1952464/in-python-how-do-i-determine-if-an-object-is-iterable>
- ▶ <https://stackoverflow.com/questions/6589967/how-to-handle-duck-typing-in-python>

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodeage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation

Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque standard

Création de modules

Interface graphiques

Code natif

Python scientifique

Vue d'ensemble

Langage Python

## Programmation Orientée objet (POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Bibliographie

Bonnes pratiques

Bibliothèque standard

Création de modules

Interface graphiques

Code natif

Python scientifique

# Programmation Orientée objet (POO)

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

**Concepts**

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Bibliographie

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

## 3- Programmation Orientée objet (POO)

### 3.1. Concepts

# Programmation orientée objet (POO)

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Bibliographie

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

La POO consiste en la définition et l'interaction de briques logicielles appelées objets ; un objet représente un concept, une idée ou toute entité du monde physique, comme une voiture, une personne ou encore une page d'un livre. Il possède une structure interne et un comportement, et il sait interagir avec ses pairs.

---

[https://fr.wikipedia.org/wiki/Programmation\\_orient%C3%A9e\\_objet](https://fr.wikipedia.org/wiki/Programmation_orient%C3%A9e_objet)

# Programmation orientée objet (POO)

Matthieu Falce

Il s'agit donc de représenter ces objets et leurs relations ; l'interaction entre les objets via leurs relations permet de concevoir et réaliser les fonctionnalités attendues, de mieux résoudre le ou les problèmes. Dès lors, l'étape de modélisation revêt une importance majeure et nécessaire pour la POO. C'est elle qui permet de transcrire les éléments du réel sous forme virtuelle.

---

[https://fr.wikipedia.org/wiki/Programmation\\_orient%C3%A9e\\_objet](https://fr.wikipedia.org/wiki/Programmation_orient%C3%A9e_objet)

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Bibliographie

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Constitution d'une classe

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Bibliographie

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Une classe est constituée de 2 entités (en gros) :

- ▶ les méthodes : des "fonctions" qui s'appliquent sur un objet
- ▶ les attributs : des "variables" qui s'appliquent sur un objet

# Constitution d'une classe

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Bibliographie

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Une classe est constituée de 2 entités (en gros) :

- ▶ les méthodes : des "fonctions" qui s'appliquent sur un objet
- ▶ les attributs : des "variables" qui s'appliquent sur un objet

Cela permet de conserver le *comportement* et *l'état* à l'intérieur de l'instance.

Des appels à des méthodes vont modifier l'état interne en changeant les attributs.

# Constitution d'une classe

Matthieu Falce

Une classe est constituée de 2 entités (en gros) :

- ▶ les méthodes : des "fonctions" qui s'appliquent sur un objet
- ▶ les attributs : des "variables" qui s'appliquent sur un objet

Cela permet de conserver le *comportement* et *l'état* à l'intérieur de l'instance.

Des appels à des méthodes vont modifier l'état interne en changeant les attributs.

Une classe est une *boîte noire*. On interagit avec elle à l'aide de quelques leviers et boutons sans savoir ce qui se passe à l'intérieur.

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Bibliographie

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Vocabulaire

Matthieu Falce

- ▶ une classe défini un nouveau *type* (comme `int`)
- ▶ un *objet* est une *instance* d'une classe (comme `2` est une instance de `int`)

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Bibliographie

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Bibliographie

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

## 3- Programmation Orientée objet (POO)

### 3.2. Association

# Association entre classes

Matthieu Falce

2 grandes techniques pour associer des classes entre elles :

- ▶ *héritage* (*inheritence* en anglais): on étend une classe mère en faisant un nouveau type qui le restreint
  - ▶ modélise la relation "*est un*"
  - ▶ le type fille peut être utilisé à la place du type mère (*polymorphisme*)
  - ▶ on peut redéfinir ou *surcharger* certains comportements (méthodes, attributs)
  - ▶ les relations classe mère / classe fille définissent un *arbre d'héritage*

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Bibliographie

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Association entre classes

Matthieu Falce

2 grandes techniques pour associer des classes entre elles :

- ▶ *composition* : on étend une classe en l'utilisant comme attribut d'une classe
  - ▶ modélise la relation "*possède un*"
  - ▶ assouplit la relation de dépendance

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Bibliographie

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Concepts

Association

**Modélisation**

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Bibliographie

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

## 3- Programmation Orientée objet (POO)

### 3.3. Modélisation

Le Langage de Modélisation Unifié, de l'anglais Unified Modeling Language (UML), est un langage de modélisation graphique à base de pictogrammes conçu pour fournir une méthode normalisée pour visualiser la conception d'un système. Il est couramment utilisé en développement logiciel et en conception orientée objet.

---

[\*\*https:\*\*  
//fr.wikipedia.org/wiki/UML\\_\(informatique\)](https://fr.wikipedia.org/wiki/UML_(informatique))

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Bibliographie

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

## Différents types de diagrammes

- ▶ *diagramme de classes* : représente les classes intervenant dans le système
- ▶ *diagramme d'objets* : représente les instances de classes
- ▶ *diagramme d'activité* : représente la suite des actions à effectuer dans le programme
- ▶ ...

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Bibliographie

Bonnes pratiques

Bibliothèque  
standard

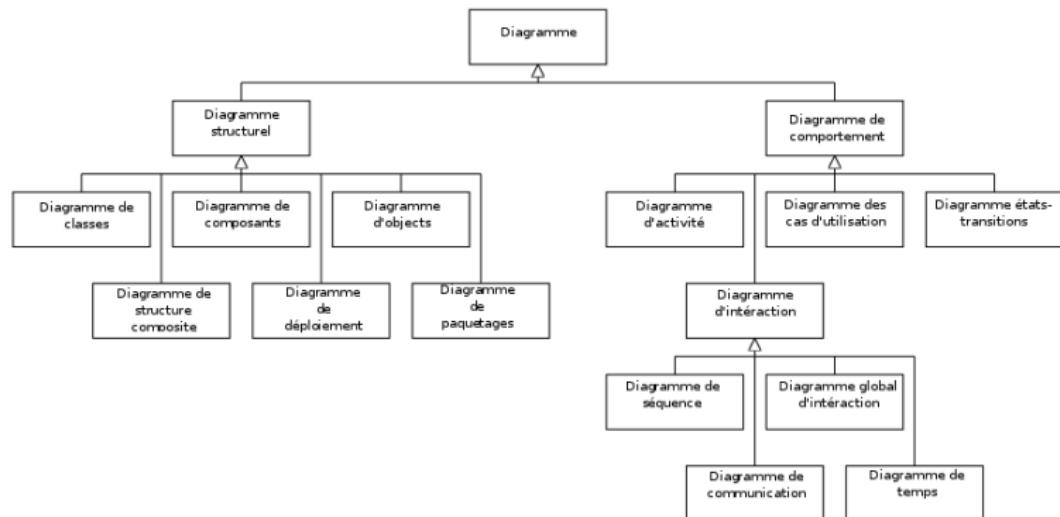
Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

## Diagramme montrant la hiérarchie de types de diagrammes UML



source: [https://fr.wikipedia.org/wiki/UML\\_\(informatique\)#/media/File:Uml\\_diagram-fr.png](https://fr.wikipedia.org/wiki/UML_(informatique)#/media/File:Uml_diagram-fr.png)

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Bibliographie

Bonnes pratiques

Bibliothèque standard

Création de modules

Interface graphiques

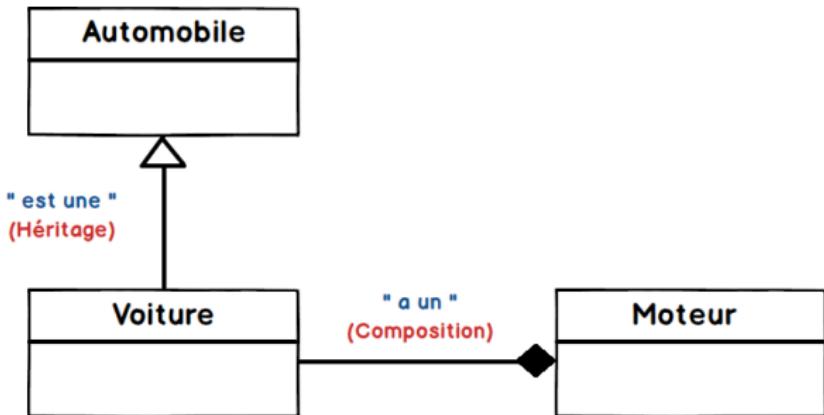
Code natif

Python scientifique

# Diagrammes de classe

Matthieu Falce

## Diagramme de classes montrant composition et héritage



source: <https://waytolearnx.com/2018/08/difference-entre-heritage-et-composition.html>

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Bibliographie

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Bibliographie

Bonnes pratiques

Bibliothèque  
standard

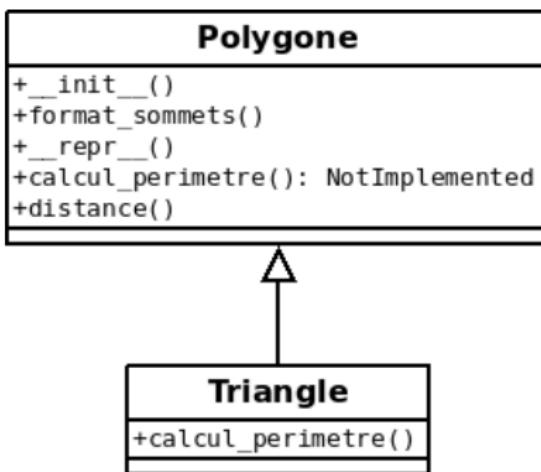
Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

## Diagramme de classes montrant un exemple d'héritage



Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Bibliographie

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

## 3- Programmation Orientée objet (POO)

### 3.4. POO en python

# Créer une classe

Matthieu Falce

```
class MonObjet():
    pass
```

```
o = MonObjet()
print(o)
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Bibliographie

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Créer une classe

Matthieu Falce

```
# Constructeur, méthodes et attributs

class MonAutreObjet:
    def __init__(self, nom):
        self.nom = nom

    def dis_ton_nom(self):
        print("Bonjour, je suis {}".format(self.nom))

o1 = MonAutreObjet(1)
o2 = MonAutreObjet(2)

print(o1.nom)
print(o2.nom)

o1.dis_ton_nom()
o2.dis_ton_nom()
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Bibliographie

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Créer une classe

Matthieu Falce

```
# Les attributs sont dynamiques et ajoutable
# TOUT EST PUBLIC (en première approximation)

class DisBonjour():
    def dis_bonjour(self):
        print("Bonjour : {}".format(self.nom))

d = DisBonjour()
try:
    # ne fonctionne pas ici, self.nom n'est pas défini
    d.dis_bonjour()
except NameError:
    pass

d.nom = "Toto" # on définit un nom à qui dire bonjour
d.dis_bonjour()
d.nom = "Tata"
d.dis_bonjour()
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Bibliographie

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Méthodes magiques

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Bibliographie

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Certaines méthodes (les `__*__`) sont utilisées par l'interpréteur pour modifier le comportement des objets.

La plus connue est `__init__` qui permet d'initialiser l'objet.

Mais il y en a d'autres.

# Méthodes magiques

Matthieu Falce

```
class Point():
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __repr__(self):
        """Appelée lors de print(Point(1,1))."""
        return "({}, {})".format(self.x, self.y)

    def __eq__(self, other):
        return self.x == other.x and self.y == other.y

    def __lt__(self, other):
        print(self, other)
        return self.x < other.x # bah

    def __gt__(self, other):
        return not self.__lt__(other)

p1 = Point(1, 1)
p2 = Point(2, 1)
assert (p1 < p2) is True
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Bibliographie

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Héritage

Matthieu Falce

```
class Bonjour():
    """Classe "abstraite"""
    """
    def __init__(self, nom):
        self.nom = nom

    def dis_ton_nom(self):
        # Méthode "abstraite"
        raise NotImplementedError

class BonjourFrancais(Bonjour):
    def dis_ton_nom(self):
        print("Bonjour, je suis {}".format(self.nom))

class BonjourItalien(Bonjour):
    def dis_ton_nom(self):
        print("Ciao, sono {}".format(self.nom))

# le __init__ et le nom sont gérés dans la classe mère
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Bibliographie

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Héritage

Matthieu Falce

```
import math

class Polygone():
    def __init__(self, sommets):
        self.sommets = [tuple(p) for p in sommets]
        self.name = "Polygone"

    def format_sommets(self):
        return " - ".join([str(point) for point in self.sommets])

    def __repr__(self):
        return "{}: {}".format(self.name, self.format_sommets())

    def calcule_perimetre(self):
        raise NotImplementedError

    def distance(self, a, b):
        return math.sqrt((a[0]-b[0]) ** 2 + (a[1] - b[1]) ** 2)

class Triangle(Polygone):
    def __init__(self, sommets):
        super().__init__(sommets) # !!
        self.name = "triangle"

    def calcule_perimetre(self):
        cotes = [
            (self.sommets[0], self.sommets[1]),
            (self.sommets[1], self.sommets[2]),
            (self.sommets[2], self.sommets[0])
        ]
        ds = [self.distance(p1, p2) for p1, p2 in cotes]
        return sum(ds)
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Bibliographie

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Accès aux attributs

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Bibliographie

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Les attributs sont publics par défaut. Comment protéger certaines contraintes dans ce cas ?

- ▶ contrat avec les autres développeurs : variables "privées", préfixées par \_ : (\_temperature)
- ▶ on peut préfixer avec un double underscore (\_\_temperature) pour les rendre inaccessible hors de l'instance (l'attribut est renommé automatiquement par l'interpréteur)
- ▶ getters / setters : utiliser les properties

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Bibliographie

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

## 3- Programmation Orientée objet (POO)

### 3.5. Gestion des exceptions

# Capturer une exception

Matthieu Falce

```
# on peut capturer une exception
try:
    a = 1 / 0
except Exception as e:
    print(e)
else:
    print("Si pas d'exception")
finally:
    print("Dans tous les cas")

# il faut essayer d'être plus précis dans son exception
try:
    a = 1 / 0
    print(a)
except ZeroDivisionError as e:
    print(e)

# on peut capturer plusieurs exceptions
li = [0]
try:
    calcul = 1 / li[0]
    print(a)
except (IndexError, ZeroDivisionError) as e:
    print(e)
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Bibliographie

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Lever une exception – Personnalisation

Matthieu Falce

```
# On peut lever des exceptions dans certains cas
def notation(note):
    if 0 < note < 20:
        raise ValueError(
            "une note est entre 0 et 20, pas {}".format(note)
        )
    # faire des choses avec la note correcte
```

```
# =====
```

```
# On peut créer ses propres exceptions
# Les exceptions héritent toutes de Exception,
# c'est pour ça que 'except Exception' fonctionne
```

```
class MaBelleException(Exception):
    pass
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Bibliographie

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Taxonomie d'exceptions de la DB API

Matthieu Falce

## Taxonomie des exceptions d'après la PEP 249

`StandardError`

  |\_\_`Warning`

  |\_\_`Error`

    |\_\_`InterfaceError`

    |\_\_`DatabaseError`

      |\_\_`DataError`

      |\_\_`OperationalError`

      |\_\_`IntegrityError`

      |\_\_`InternalError`

      |\_\_`ProgrammingError`

      |\_\_`NotSupportedError`

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Bibliographie

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Bibliographie

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

## 3- Programmation Orientée objet (POO)

### 3.6. Classe ou pas ?

# Quand utiliser une classe ?

Matthieu Falce

```
class Bonjour():
    def __init__(self, nom):
        self.nom = nom

    def parle(self):
        return "Bonjour {}".format(self.nom)

bonjour = Bonjour("Matthieu")
print(bonjour.parle())
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Bibliographie

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Quand utiliser une classe ?

Matthieu Falce

```
class Bonjour():
    def __init__(self, nom):
        self.nom = nom

    def parle(self):
        return "Bonjour {}".format(self.nom)
```

```
bonjour = Bonjour("Matthieu")
print(bonjour.parle())
```

---

```
def bonjour(nom):
    return "Bonjour {}".format(nom)

print(bonjour("Matthieu"))
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Bibliographie

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Quand utiliser une classe ?

Matthieu Falce

## ► Ne pas utiliser

- ▶ quand moins de 2 méthodes...
- ▶ seulement conteneurs, pas de méthodes (utiliser plutôt `dict`, `namedtuple`, ...)
- ▶ gestion des ressources (plutôt `context manager`)

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Bibliographie

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Quand utiliser une classe ?

Matthieu Falce

## ► Ne pas utiliser

- ▶ quand moins de 2 méthodes...
- ▶ seulement conteneurs, pas de méthodes (utiliser plutôt `dict`, `namedtuple`, ...)
- ▶ gestion des ressources (plutôt `context manager`)

## ► Utiliser une classe

- ▶ organisation (boîte noire)
- ▶ conserver un état
- ▶ profiter de l'OOP (héritage, ...)
- ▶ surcharge d'opérateurs / méthodes magiques
- ▶ produire une API définie

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Bibliographie

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Conteneurs

Matthieu Falce

```
Point2d = collections.namedtuple('Point2d', ['x', 'y'])
p1 = Point2d(3, 2)
p2 = Point2d(10, 1)

dist = math.sqrt(
    (p2.x - p1.x)**2 + (p2.y - p1.y)**2
)
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Bibliographie

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Dataclasses

Matthieu Falce



Version python  $\geqslant 3.7$

```
@dataclass
class InventoryItem:
    '''Class for keeping track of an item in inventory.'''
    name: str
    unit_price: float
    quantity_on_hand: int = 0

    def total_cost(self) -> float:
        return self.unit_price * self.quantity_on_hand
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Bibliographie

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

**Méthodes**

Bibliographie

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

## 3- Programmation Orientée objet (POO)

### 3.7. Méthodes

# Différents types de méthodes

Matthieu Falce

```
class Exemple():
    def __init__(self, attribut):
        self.attribut = attribut

    def methode(self, param):
        print(self, type(self))
        return self.attribut + param

e = Exemple(10)
print(e.methode(2))
```

## method

- ▶ classique
- ▶ s'applique à une instance
- ▶ accès aux variables de classe et d'instance
- ▶ **self** est injecté automatiquement (bound method)

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Bibliographie

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Différents types de méthodes

Matthieu Falce

```
class Exemple():
    variable_de_classe = 1

    @classmethod
    def methode_de_classe(cls, param):
        print(cls, type(cls))
        return cls.variable_de_classe + param
```

```
print(Exemple.methode_de_classe(5))
```

## classmethod

- ▶ s'applique sur une classe et pas une instance
- ▶ accès aux variables de classe
- ▶ `cls` est injecté automatiquement

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Bibliographie

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Différents types de méthodes

Matthieu Falce

```
class Galette():
    def __init__(self, ingredients):
        self.ingredients = ingredients

    @classmethod
    def complete(cls):
        return cls(["jambon", "fromage", "oeuf"])

    @classmethod
    def nature(cls):
        return cls(["beurre salé"])

print(Galette.complete().ingredients)
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Bibliographie

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Différents types de méthodes

Matthieu Falce

```
class Exemple():
    @staticmethod
    def methode_statique(param):
        return param
```

```
print(Exemple.methode_statique(5))
```

## staticmethod

- ▶ permet de regrouper des fonctions dans l'objet
- ▶ n'a accès à aucune information classe ou instance
- ▶ ne va pas modifier l'état de la classe ou de l'instance

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Bibliographie

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

## Quel est le résultat ?

```
class MyClass:  
    def method(self):  
        return "méthode d'instance", self  
  
    @classmethod  
    def _classmethod(cls):  
        return 'méthode de classe', cls  
  
    @staticmethod  
    def _staticmethod():  
        return 'méthode statique'  
  
print(MyClass._staticmethod())  
print(MyClass._classmethod())  
print(MyClass.method())  
  
m = MyClass()  
print(m._staticmethod())  
print(m._classmethod())  
print(m.method())
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Bibliographie

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

**Bibliographie**

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

## 3- Programmation Orientée objet (POO)

### 3.8. Bibliographie

# Bibliographie I

Matthieu Falce

- ▶ Tous les sujets :
  - ▶ <http://www.dabeaz.com/py3meta/Py3Meta.pdf>
- ▶ Classe ou pas
  - ▶ <https://eev.ee/blog/2013/03/03/the-controller-pattern-is-awful-and-other-oo-heresy/>
  - ▶ <https://www.youtube.com/watch?v=o9pEzgHorH0>
  - ▶ <http://lucumr.pocoo.org/2013/2/13/moar-classes/>
- ▶ Méthodes de classe / statiques / méthode :
  - ▶ <https://realpython.com/instance-class-and-static-methods-demystified/>
  - ▶ commentaire de l'article  
<http://sametmax.com/comprendre-les-decorateur-python-pas-a-pas-partie-2/>
  - ▶ <https://rushter.com/blog/python-class-internals/>

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Bibliographie

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Bibliographie II

Matthieu Falce

- ▶ Loi de Demeter :
- ▶ <https://www2.ccs.neu.edu/research/demeter/demeter-method/LawOfDemeter/paper-boy/demeter.pdf>

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

**Bibliographie**

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

## Bonnes pratiques

Pourquoi ?

Installation de paquets

Débug

Linter

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Bonnes pratiques

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Débug

Linter

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

## 4- Bonnes pratiques

### 4.1. Pourquoi ?

# Qu'est-ce que c'est ?

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Débug

Linter

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

## La QA (*Quality Assurance*)

- ▶ monitore le développement logiciel et les méthodes utilisées
- ▶ doit être suivie et contrôlée
- ▶ doit s'adapter aux nécessités métier (ne pas être trop contraignante)

# Pourquoi ?

Matthieu Falce

- ▶ le code est plus souvent lu que écrit
  - ▶ règle de nommage des fichiers / modules / fonctions / variables
  - ▶ *linter*
  - ▶ documentation (qui évolue avec le code)

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Débug

Linter

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Pourquoi ?

Matthieu Falce

- ▶ le code est plus souvent lu que écrit
  - ▶ règle de nommage des fichiers / modules / fonctions / variables
  - ▶ *linter*
  - ▶ documentation (qui évolue avec le code)
- ▶ le code doit fonctionner
  - ▶ vérifier le code avec des tests unitaires
  - ▶ utiliser des vérificateurs de typage statique

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Débug

Linter

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Pourquoi ?

Matthieu Falce

- ▶ le code est plus souvent lu que écrit
  - ▶ règle de nommage des fichiers / modules / fonctions / variables
  - ▶ *linter*
  - ▶ documentation (qui évolue avec le code)
- ▶ le code doit fonctionner
  - ▶ vérifier le code avec des tests unitaires
  - ▶ utiliser des vérificateurs de typage statique
- ▶ le code doit pouvoir être déployé facilement
  - ▶ utiliser des système de build automatiques (qui évoluent avec le code)
  - ▶ utiliser un système d'intégration continue (*CI*)

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Débug

Linter

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Pourquoi ?

Matthieu Falce

- ▶ le code est plus souvent lu que écrit
  - ▶ règle de nommage des fichiers / modules / fonctions / variables
  - ▶ *linter*
  - ▶ documentation (qui évolue avec le code)
- ▶ le code doit fonctionner
  - ▶ vérifier le code avec des tests unitaires
  - ▶ utiliser des vérificateurs de typage statique
- ▶ le code doit pouvoir être déployé facilement
  - ▶ utiliser des systèmes de build automatiques (qui évoluent avec le code)
  - ▶ utiliser un système d'intégration continue (*CI*)
- ▶ on peut revenir à une version antérieure du projet / savoir qui a fait quoi / quand
  - ▶ utiliser un système de contrôle de version (Git, ...)

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Débug

Linter

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

**Installation de paquets**

Avant propos

Écosystème

pip

Environnements virtuels

Débug

*Linter*

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

## 4- Bonnes pratiques

### 4.2. Installation de paquets

# Avant Propos

Matthieu Falce

Le packaging en python est relativement mal connu et compris.

- ▶ plusieurs outils concurrents (`distutils`, `setuptools`, `pip`, `pipenv`, `virtuelenv`...)
- ▶ difficulté à installer des packages (compilation à l'installation)
- ▶ peu de considération des “core dev”

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Avant propos

Écosystème

pip

Environnements virtuels

Débug

Linter

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Avant Propos

Matthieu Falce

Le packaging en python est relativement mal connu et compris.

- ▶ plusieurs outils concurrents (`distutils`, `setuptools`, `pip`, `pipenv`, `virtuelenv`...)
- ▶ difficulté à installer des packages (compilation à l'installation)
- ▶ peu de considération des “core dev”

Ce n'est plus trop le cas aujourd'hui.

A présent : outils matures, inclus par défaut et utilisés.

Merci au PyPA <3

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Avant propos

Écosystème

pip

Environnements virtuels

Débug

Linter

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Écosystème

Matthieu Falce

## ► Gestion environnement :

- ▶ `virtualenv` (+ wrappers)
- ▶ `pip`
- ▶ `easy_install`
- ▶ `pipenv`
- ▶ `(conda)`

## ▶ PyPI

## ▶ wheels

## ▶ eggs

## ▶ ...

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Avant propos

Écosystème

pip

Environnements virtuels

Débug

Linter

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Avant propos

Écosystème

pip

Environnements virtuels

Débug

Linter

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Si on lui donne un chemin, pip cherche un setup.py

Si on lui donne un nom, il va chercher sur pypi.

On peut aussi lui donner un chemin distant en http / git / hg / ...

```
# installation depuis Pypi  
pip install numpy
```

# Commandes classiques

Matthieu Falce

## Installation

```
# installer depuis PyPi
pip install unModule

# installer depuis un wheel local
pip install unModule-1.0-py2.py3-none-any.whl

# installer une version "précise"
pip install unModule==0.10.1
pip install unModule>=0.9,<0.11

# installation depuis un chemin
pip install .

# installation depuis git
## url d'un dépôt git
## git@github.com:pypa/sampleproject.git
## on doit rajouter git+ssh:// et changer le :pypa en /pypa
pip install git+ssh://git@github.com/pypa/sampleproject.git

# installer des paquets avec des options
pip install "project[extra]"
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Avant propos

Écosystème

pip

Environnements virtuels

Débug

Linter

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Commandes classiques

Matthieu Falce

## Installation (cas particuliers)

```
# installation depuis un chemin
pip install .

# installation depuis git
## url d'un dépôt git
## git@github.com:pypa/sampleproject.git
## on doit rajouter git+ssh:// et changer le :pypa en /pypa
pip install git+ssh://git@github.com/pypa/sampleproject.git

# installer des paquets avec des options
pip install "project[extra]"
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Avant propos

Écosystème

pip

Environnements virtuels

Débug

Linter

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Commandes classiques

Matthieu Falce

## Cycle de vie des paquets installés

```
# lister les modules non à jour  
pip list --outdated
```

```
# mettre à jour un module  
pip install --upgrade unModule  
pip install -U unModule
```

```
# supprimer un module  
pip uninstall SomePackage
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Avant propos

Écosystème

pip

Environnements virtuels

Débug

Linter

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Commandes classiques

Matthieu Falce

## Fichier requirements.txt

```
# freeze des dépendances
pip freeze > requirements.txt
```

```
# installer depuis un fichier de requirements
pip install -r requirements.txt
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Avant propos

Écosystème

pip

Environnements virtuels

Débug

Linter

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Autres commandes

Matthieu Falce

- ▶ pip download (télécharge sans installer)
- ▶ pip list (liste les paquets installés)
- ▶ pip show (liste les informations sur les paquets installés)
- ▶ pip search (cherche les paquets avec un nom compatible)
- ▶ pip check (vérifie si les dépendances sont compatibles)
- ▶ pip wheel (construit un wheel)
- ▶ pip hash (calcule le *hash* d'un module)

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Avant propos

Écosystème

pip

Environnements virtuels

Débug

Linter

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

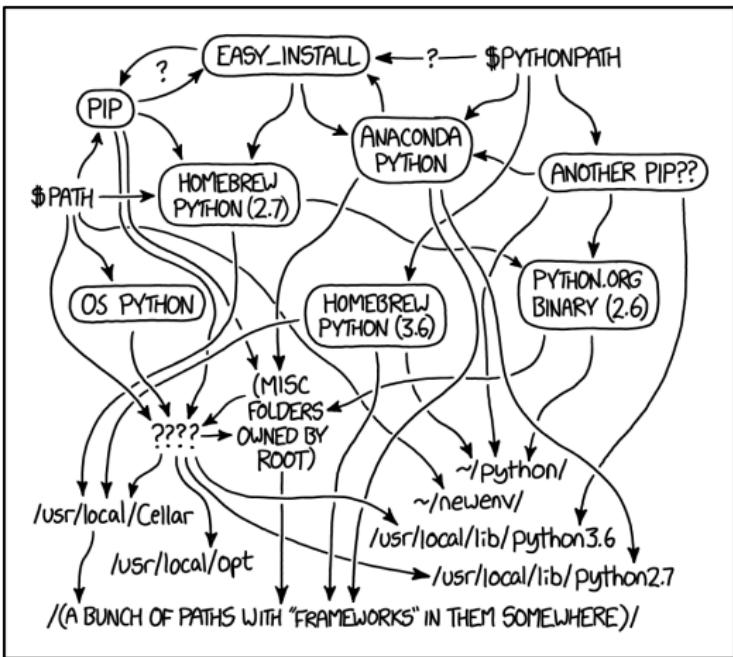
Interface  
graphiques

Code natif

Python scientifique

# Environnement d'installation sain

Matthieu Falce



MY PYTHON ENVIRONMENT HAS BECOME SO DEGRADED  
THAT MY LAPTOP HAS BEEN DECLARED A SUPERFUND SITE.

<https://xkcd.com/1987/>

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Avant propos

Écosystème

pip

Environnements virtuels

Débug

Linter

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Environnement d'installation sain

Matthieu Falce

- ▶ savoir ce que l'on installe ;
- ▶ savoir comment on l'installe ;
- ▶ savoir où on l'installe ;

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Avant propos

Écosystème

pip

Environnements virtuels

Débug

Linter

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Installer des modules externes

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Avant propos

Écosystème

pip

Environnements virtuels

Débug

Linter

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

On ne veut pas forcément installer des dépendances de façon globale :

- ▶ `virtualenv` (solution standard)
- ▶ `conda env` (développé par Continuum Analytics, ceux qui font Anaconda, utilisé en calcul scientifique, gère les bibliothèques C...)

# virtualenv

Matthieu Falce

- ▶ s'abstraire du python système
- ▶ changer de projet facilement
- ▶ avoir des versions différentes de bibliothèques installées en parallèle
- ▶ être "iso" avec l'environnement de production (plus subtil que ça)

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Avant propos

Écosystème

pip

Environnements virtuels

Débug

Linter

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# virtualenv

Matthieu Falce

```
#installation (avec le Python système)
pip install virtualenv

# aller dans le dossier où l'on veut créer le venv
# dossier du projet ou dossier commun à tous les venvs
cd my_project_folder

# on crée le venv
virtualenv venv

# on l'active (modifie les variables d'environnement pour Python)
source venv/bin/activate

# on vérifie que ça a marché
which python

### c'est ici qu'on travaille...

# on désactive pour quitter (restore les variables d'environnement)
deactivate
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Avant propos

Écosystème

pip

Environnements virtuels

Débug

Linter

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Avant propos

Écosystème

pip

Environnements virtuels

Débug

Linter

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

## Python système

### Projet data 1

python 2.7  
seaborn 0.9.0  
numpy 1.16.1  
pandas 0.24.1  
...

### Projet web 1

Python 3.6  
Django 1.11  
request 2.21.0  
psycopg2.7  
...

### Projet data 2

python 3.6  
scipy 0.19.0  
numpy 1.13.1  
pandas 0.20.1  
...

...

Coexistence de plusieurs versions de Python

# virtualenv

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Avant propos

Écosystème

pip

**Environnements virtuels**

Débug

Linter

Analyse des performances

Tests

Documentation

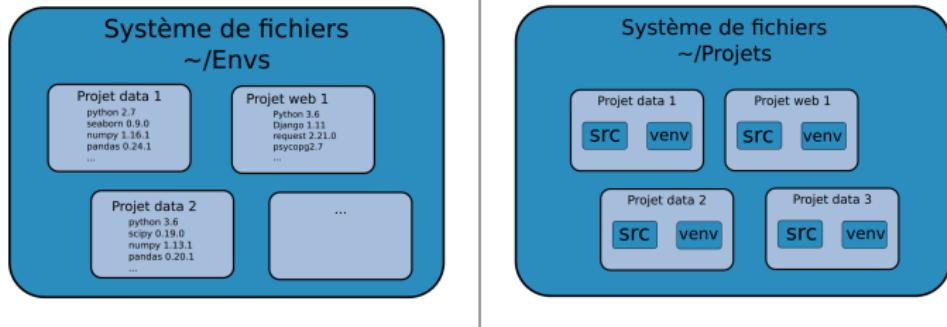
Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique



## Organisation des environnements virtuels

# virtualenv

Matthieu Falce

- ▶ on peut préciser la version de python (`virtualenv -p /usr/bin/python2.7 venv`)
- ▶ s'utilise souvent avec des *wrappers*
  - ▶ `pew`
  - ▶ `virtualenvwrapper`
  - ▶ ...
- ▶ ne permet pas l'isolation parfaite, juste Python
  - ▶ les dépendances externes (installer un paquet système) peuvent être gérées (`wheel`)
  - ▶ utiliser Vagrant ou Docker dans les cas complexes

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Avant propos

Écosystème

pip

Environnements virtuels

Débug

Linter

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Débug

Linter

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

## 4- Bonnes pratiques

### 4.3. Débug

# Outils de débogage

Matthieu Falce

Python contient des outils permettant de débuger et d'analyser le bytecode généré pour une fonction

```
import pdb, dis

for i in range(-10, 11):
    try:
        print(100 / i)
    except Exception:
        import pdb; pdb.set_trace()

#####
def rapide():
    return 1

def lente():
    a = 5
    return a

print("décompilation de rapide : ")
dis.dis(rapide)
print("décompilation de lente : ")
dis.dis(lente)
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Débug

Linter

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Débug

*Linter*

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

## 4- Bonnes pratiques

### 4.4. *Linter*

# Qualité du code – pep8 / linters

Matthieu Falce

Python propose sa vision d'un "code propre" : la PEP8

- ▶ indentation avec 4 espaces
- ▶ lignes de 80 caractères
- ▶ respect d'une aération du code
- ▶ espace dans les expressions
- ▶ ...

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Débug

*Linter*

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Qualité du code – pep8 / linters

Matthieu Falce

Il existe des “linters” pour vous assister dans l’écriture.<sup>16</sup> <sup>17</sup>  
Ils peuvent lister les erreurs, variables non déclarées, typos,  
mauvais import...

- ▶ flake8
- ▶ pylint
- ▶ ...

Intégration avec les éditeurs de texte.

Vue d’ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Débug

*Linter*

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

---

16.https:

//stackoverflow.com/questions/5611776/what-are-the-comprehensive-lint-checkers-for-python

17.https:

//stackoverflow.com/questions/1428872/pylint-pychecker-or-pyflakes?noredirect=1&lq=1

# Qualité du code – pep8 / linters

Matthieu Falce

Certains outils reformatent automatiquement le code que vous leur donnez (concentration sur le code plutôt que la présentation).<sup>16</sup> <sup>17</sup>

- ▶ black
- ▶ yapf
- ▶ autopep8
- ▶ ...

Intégration avec les éditeurs de texte.

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Débug

Linter

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

16. <https://medium.com/3yourmind/auto-formatters-for-python-8925065f9505>

17. <https://news.ycombinator.com/item?id=17155048>

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Débug

Linter

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

## 4- Bonnes pratiques

### 4.5. Analyse des performances

# Timing et profilage

Matthieu Falce

```
import time, timeit, cProfile

def fonction_1():
    sum([i for i in range(int(1e5))])

def fonction_2():
    sum(i for i in range(int(1e5)))

tic = time.time()
fonction_1()
print("fonction 1 : {}".format(time.time() - tic))

print("100x fonction2 : {}".format(
    timeit.timeit("fonction_2()", number=100, globals=globals()))
))

cProfile.run('fonction_1()')
cProfile.run('fonction_2()')
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Débug

Linter

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Timing et profilage

Matthieu Falce

## Résultat

```
6 function calls in 0.004 seconds
Ordered by: standard name
ncalls  tottime  percall  cumtime  percall filename:lineno(function)
    1    0.001    0.001    0.004    0.004 <ipython-input-8-ac539deb9692>:4(fonction_1)
    1    0.002    0.002    0.002    0.002 <ipython-input-8-ac539deb9692>:5(<listcomp>)
    1    0.000    0.000    0.004    0.004 <string>:1(<module>)
    1    0.000    0.000    0.004    0.004 {built-in method builtins.exec}
    1    0.001    0.001    0.001    0.001 {built-in method builtins.sum}
    1    0.000    0.000    0.000    0.000 {method 'disable' of '\_lsprof.Profiler' objects}

=====
100006 function calls in 0.012 seconds
Ordered by: standard name
ncalls  tottime  percall  cumtime  percall filename:lineno(function)
    1    0.000    0.000    0.012    0.012 <ipython-input-8-ac539deb9692>:8(fonction_2)
100001    0.006    0.000    0.006    0.000 <ipython-input-8-ac539deb9692>:9(<genexpr>)
    1    0.000    0.000    0.012    0.012 <string>:1(<module>)
    1    0.000    0.000    0.012    0.012 {built-in method builtins.exec}
    1    0.006    0.006    0.012    0.012 {built-in method builtins.sum}
    1    0.000    0.000    0.000    0.000 {method 'disable' of '\_lsprof.Profiler' objects}
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?  
Installation de paquets  
Débug  
Linter  
Analyse des performances  
Tests  
Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Débug

Linter

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

## 4- Bonnes pratiques

### 4.6. Tests

En programmation informatique, le test unitaire ou test de composants est une procédure permettant de vérifier le bon fonctionnement d'une partie précise d'un logiciel ou d'une portion d'un programme (appelée « unité » ou « module »). Dans les applications non critiques, l'écriture des tests unitaires a longtemps été considérée comme une tâche secondaire. Cependant, les méthodes Extreme programming (XP) ou Test Driven Development (TDD) ont remis les tests unitaires, appelés ‘tests du programmeur’, au centre de l’activité de programmation. À noter que le test unitaire peut ne pas être automatique.

---

[https://fr.wikipedia.org/wiki/Test\\_unitaire](https://fr.wikipedia.org/wiki/Test_unitaire)

Nous allons utiliser la bibliothèque unittest<sup>18</sup>

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Débug

Linter

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

---

18.<https://docs.python.org/3/library/unittest.html>

# Tests unitaires – tests verts

Matthieu Falce

```
import unittest

class TestThings(unittest.TestCase):
    def test_upper(self):
        self.assertEqual('foo'.upper(), 'FOO')

    def test_isupper(self):
        self.assertTrue('FOO'.isupper())
        self.assertFalse('Foo'.isupper())

    def test_split(self):
        s = 'hello world'
        self.assertEqual(s.split(), ['hello', 'world'])
        # check that s.split fails when the separator is not a string
        with self.assertRaises(TypeError):
            s.split(2)

    def test_almostEqual(self):
        self.assertAlmostEqual(1/3, 0.333333333333)

if __name__ == '__main__':
    unittest.main()
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Débug

Linter

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Tests unitaires – tests verts

Matthieu Falce

## Résultat :

```
python test_unittest.py
```

```
....
```

```
-----
```

```
Ran 4 tests in 0.001s
```

```
OK
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Débug

Linter

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Tests unitaires – tests rouges

Matthieu Falce

```
import unittest

class TestErrors(unittest.TestCase):
    def test_error(self):
        computation = 2+2
        should_be = 3
        self.assertEqual(computation, should_be)

    def test_exception(self):
        computation = 1/0
        should_not_be = 1
        self.assertNotEqual(computation, should_be)

if __name__ == '__main__':
    unittest.main()
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Débug

Linter

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Tests unitaires – tests rouges

Matthieu Falce

## Résultat :

```
FE.  
=====  
ERROR: test_exception (__main__.TestMath)  
-----  
Traceback (most recent call last):  
  File "../codes/modules/test_unittest2.py", line 13, in test_exception  
    computation = 1/0  
ZeroDivisionError: division by zero  
  
=====  
FAIL: test_error (__main__.TestMath)  
-----  
Traceback (most recent call last):  
  File "../codes/modules/test_unittest2.py", line 10, in test_error  
    self.assertEqual(computation, should_be)  
AssertionError: 4 != 3  
  
-----  
Ran 3 tests in 0.001s  
  
FAILED (failures=1, errors=1)
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Débug

Linter

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Tests unitaires – fixtures

Matthieu Falce

```
import unittest

class FixturesTest(unittest.TestCase):
    @classmethod
    def setUpClass(cls):
        print('In setUpClass()'); cls.set_for_class = 10

    @classmethod
    def tearDownClass(cls):
        print('\nIn tearDownClass()'); print(cls.set_for_class)
        del cls.set_for_class

    def setUp(self):
        super().setUp(); print('\n      In setUp()')
        self.set_for_function = 5

    def tearDown(self):
        print('      In tearDown()', '\n      ', 'set_for_function:', self.set_for_function)
        del self.set_for_function; super().tearDown()

    def test1(self):
        print('          In test1()');
        print('          ', FixturesTest.set_for_class, '\n          ', self.set_for_function);
        FixturesTest.set_for_class = 1; self.set_for_function = 2

    def test2(self):
        print('          In test2()');
        print('          ', FixturesTest.set_for_class, '\n          ', self.set_for_function);
        FixturesTest.set_for_class = 3; self.set_for_function = 4

if __name__ == '__main__':
    unittest.main()
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Débug

Linter

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Tests unitaires – fixtures

Matthieu Falce

Voilà le résultat :

```
In setUpClass()

In setUp()
    In test1()
        10
        5
In tearDown()
    set_for_function: 2
.

In setUp()
    In test2()
        1
        5
In tearDown()
    set_for_function: 4
.

In tearDownClass()
3

-----
Ran 2 tests in 0.000s

OK
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Débug

Linter

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Bonne explication du module unittest :

<https://pymotw.com/3/unittest/>

Pour aller plus loin:

- ▶ découverte automatique de tests
- ▶ tearDown plus fiables
- ▶ code coverage et rapports
- ▶ ...

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Débug

Linter

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Aller plus loin

Matthieu Falce

Bonne explication du module unittest :

<https://pymotw.com/3/unittest/>

Pour aller plus loin:

- ▶ découverte automatique de tests
- ▶ tearDown plus fiables
- ▶ code coverage et rapports
- ▶ ...

## Cycle TDD (*Test Driven Development*)

1. écriture du test
2. erreur
3. écriture du code minimal pour passer le test
4. le test passe
5. retour à 1.

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Débug

Linter

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Il existe d'autres modules pour lancer les tests ('testrunners')

<sup>19</sup>:

- ▶ (doctest<sup>20</sup>)
- ▶ nose<sup>21</sup>
- ▶ pytest (allège la syntaxe des tests)<sup>22</sup>

Les tests sont souvent utilisés avec des 'mocks'<sup>23</sup> pour modifier le comportement des modules externes.

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Débug

Linter

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

19.<https://stackoverflow.com/questions/28408750/unittest-vs-pytest-vs-nose>

20.<https://docs.python.org/3.6/library/doctest.html>

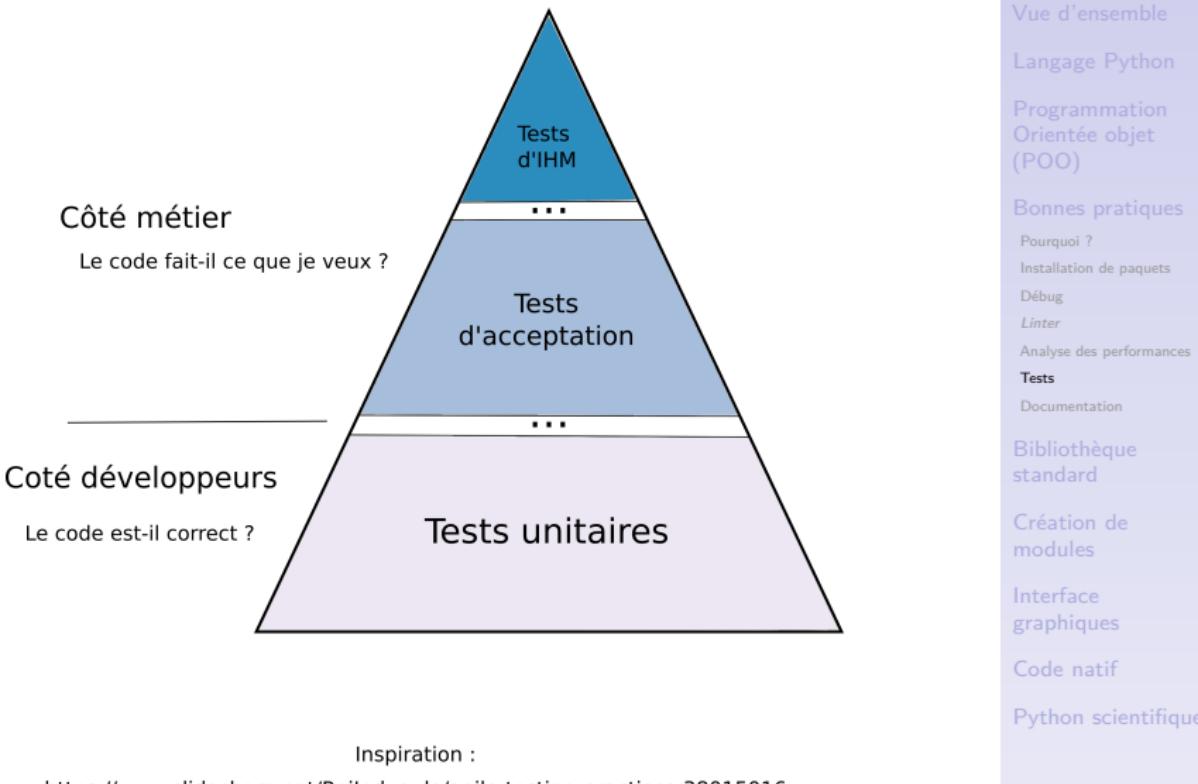
21.<https://nose.readthedocs.io/en/latest/>

22.<https://docs.pytest.org/en/latest/>

23.<https://docs.python.org/3.6/library/unittest.mock.html>

# Aller plus loin – autres types de tests

Matthieu Falce



Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Débug

Linter

Analyse des performances

Tests

**Documentation**

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

## 4- Bonnes pratiques

### 4.7. Documentation

# Documentation ?

Matthieu Falce

- ▶ commentaires : donner des informations aux autres développeurs
- ▶ docstring : pour tout le monde

"""

*Une docstring pour le module / fichier ...*

*Ici on décrit ce que doit faire le module*

"""

```
def spam(arg):  
    """  
        Une docstring pour la fonction  
  
    Params:  
        arg: int  
            Retourné par la fonction  
    """  
    # Attention : magique, ne pas toucher  
    return arg
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Débug

Linter

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Documentation ?

Matthieu Falce

- ▶ commentaires : donner des informations aux autres développeurs
- ▶ docstring : pour tout le monde

```
"""
Une docstring pour le module / fichier ...
Ici on décrit ce que doit faire le module
"""
```

```
def spam(arg):
    """
    Une docstring pour la fonction

    Params:
        arg: int
            Retourné par la fonction
    """
    # Attention : magique, ne pas toucher
    return arg
```

Les docstrings sont traitées comme des objets python par l'interpréteur.

```
""" Show how to display docstrings in python."""
# help(int)
# print(int.__doc__)
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Débug

Linter

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Comment écrire sa documentation ?

Matthieu Falce

## Exemple minimal

```
def add(a, b):
    """Addition for floats."""
    return float(a + b)
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Débug

Linter

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Comment écrire sa documentation ?

Matthieu Falce

## Exemple complet

```
"""  
This module defines some operations on floating point numbers.  
"""
```

```
def add_float(a, b):  
    """
```

*Adds two numbers and casts them to float.*

*Implements the binary function performing internal  
law of composition on floats.*

*See:*

- \* [https://en.wikipedia.org/wiki/Binary\\_function](https://en.wikipedia.org/wiki/Binary_function)
- \* [https://fr.wikipedia.org/wiki/Loi\\_de\\_composition\\_interne](https://fr.wikipedia.org/wiki/Loi_de_composition_interne)

*Args:*

- arg1(float): First number to sum*
- arg2(float): Second number to sum*

*Returns:*

*float: Sum of the 2 arguments*

```
"""
```

```
return float(a + b)
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Débug

Linter

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Outils d'extraction de documentation

Matthieu Falce

- ▶ sphinx (semi automatique) avec :
  - ▶ autosummary <sup>24</sup>
  - ▶ autodoc <sup>25</sup>

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Débug

Linter

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

---

24.<http://www.sphinx-doc.org/en/master/usage/extensions/autosummary.html>

25.<http://www.sphinx-doc.org/en/master/usage/extensions/autodoc.html>

# Outils d'extraction de documentation

Matthieu Falce

- ▶ sphinx (semi automatique) avec :
  - ▶ autosummary <sup>24</sup>
  - ▶ autodoc <sup>25</sup>
- ▶ sphinx (automatique) avec :
  - ▶ autoapi <sup>26</sup>
  - ▶ sphinx-autoapi <sup>27</sup>

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Débug

Linter

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

---

24.<http://www.sphinx-doc.org/en/master/usage/extensions/autosummary.html>

25.<http://www.sphinx-doc.org/en/master/usage/extensions/autodoc.html>

26.<http://autoapi.readthedocs.io/>

27.<http://sphinx-autoapi.readthedocs.io/en/latest/index.html>

# Outils d'extraction de documentation

Matthieu Falce

- ▶ sphinx (semi automatique) avec :
  - ▶ autosummary <sup>24</sup>
  - ▶ autodoc <sup>25</sup>
- ▶ sphinx (automatique) avec :
  - ▶ autoapi <sup>26</sup>
  - ▶ sphinx-autoapi <sup>27</sup>
- ▶ pdoc <sup>28</sup>
- ▶ pydoc <sup>29</sup>
- ▶ doxygen <sup>30</sup>

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Débug

Linter

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

---

24.<http://www.sphinx-doc.org/en/master/usage/extensions/autosummary.html>

25.<http://www.sphinx-doc.org/en/master/usage/extensions/autodoc.html>

26.<http://autoapi.readthedocs.io/>

27.<http://sphinx-autoapi.readthedocs.io/en/latest/index.html>

28.<https://github.com/mitmproxy/pdoc>

29.<https://docs.python.org/3.6/library/pydoc.html>

30.<http://www.stack.nl/~dimitri/doxygen/>

# Syntaxe pour extraction automatique

Matthieu Falce

- ▶ PEP 8 : <https://www.python.org/dev/peps/pep-0008/#documentation-strings>
- ▶ PEP 257 :  
<https://www.python.org/dev/peps/pep-0257/>

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Débug

Linter

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Syntaxe pour extraction automatique

Matthieu Falce

- ▶ PEP 8 : <https://www.python.org/dev/peps/pep-0008/#documentation-strings>
- ▶ PEP 257 :  
<https://www.python.org/dev/peps/pep-0257/>
- ▶ pdoc : markdown <sup>31</sup>
- ▶ doxygen : markdown + syntaxe spécifique <sup>32</sup>
- ▶ sphinx : RestructuredText <sup>33</sup>
- ▶ sphinx avec extension Napoleon <sup>34</sup>
  - ▶ Google <sup>35</sup>
  - ▶ Numpy <sup>36</sup>

---

31.<https://help.github.com/articles/basic-writing-and-formatting-syntax/>

32.<https://www.stack.nl/~dimitri/doxygen/manual/docblocks.html>

33.[https://thomas-cokelaer.info/tutorials/sphinx/rest\\_syntax.html](https://thomas-cokelaer.info/tutorials/sphinx/rest_syntax.html)

34.<http://www.sphinx-doc.org/en/master/usage/extensions/napoleon.html>

35.<https://github.com/google/styleguide/blob/gh-pages/pyguide.md>

36.<https://numpydoc.readthedocs.io/en/latest/format.html>

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Débug

Linter

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Formatage des docstrings – Doxygen

Matthieu Falce

```
## @package pyexample
# Documentation for this module.
#
# More details.

## Documentation for a function.
#
# More details.
def func():
    pass
## Documentation for a class.
#
# More details.
class PyClass:

    ## The constructor.
    def __init__(self):
        self._memVar = 0;

    ## Documentation for a method.
    # @param self The object pointer.
    def PyMethod(self):
        pass

    ## A class variable.
    classVar = 0;
    ## @var _memVar
    # a member variable
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Débug

Linter

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Formatage des docstrings – Doxygen

Matthieu Falce

## Python

Main Page Packages ▾ Classes ▾

### pyexample Namespace Reference

Documentation for this module. [More...](#)

#### Classes

class **PyClass**  
Documentation for a class. [More...](#)

#### Functions

def **func()**  
Documentation for a function. [More...](#)

#### Detailed Description

Documentation for this module.  
More details.

#### Function Documentation

◆ **func()**

```
def pyexample.func( )
```

Documentation for a function.  
More details.

Generated by **doxygen** 1.8.15

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Débug

Linter

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Résultat HTML de l'exemple précédent

# Formatage des docstrings – reST

Matthieu Falce

"""

*This is a reST style.*

```
:param param1: this is a first param
:param param2: this is a second param
:returns: this is a description of what is returned
:raises KeyError: raises an exception
"""
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Débug

Linter

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Formatage des docstrings – Google vs Numpy

Matthieu Falce

```
"""
This is an example of Google style.
```

## Args:

```
    param1 (array): This is the first param.  
    param2: This is a second param.
```

## Returns:

```
    This is a description of what  
    is returned.
```

## Raises:

```
    KeyError: Raises an exception.
```

```
"""
```

```
"""
This is an example of numpydoc style.
```

## Parameters

```
-----  
param1 : array_like  
    This is the first param.  
param2 :  
    This is a second param.
```

## Returns

```
-----  
string  
    This is a description of what  
    is returned.
```

## Raises

```
-----  
KeyError  
    when a key error
"""
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Débug

Linter

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Formatage des docstrings – Google vs Numpy

Matthieu Falce

## `exemple_docstring_simple.top_secret(param1, param2)`

This is an example of Google style.

- Parameters:
- `param1` – This is the first param.
  - `param2` – This is a second param.

Returns:

This is a description of what is returned.

Raises:

`KeyError` – Raises an exception.

Résultat HTML de l'exemple précédent

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Débug

Linter

Analyse des performances

Tests

Documentation

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Bibliographie

Matthieu Falce

## ► documentation

- ▶ <http://queirozf.com/entries/docstrings-by-example-documenting-python-code-the-right-way>
- ▶ <https://stackoverflow.com/questions/3898572/what-is-the-standard-python-docstring-format>
- ▶ <https://docs.python-guide.org/writing/documentation/>
- ▶ <https://fr.slideshare.net/shimizukawa/sphinx-autodoc-automated-api-documentation-europython-2015-in-bilbao>
- ▶ génération / formattage automatique des docstrings :  
<https://github.com/dadadel/pymment>

## ► code formatters

- ▶ <http://sametmax.com/once-you-go-black-you-never-go-back/>

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Pourquoi ?  
Installation de paquets  
Débug  
Linter

Analyse des performances  
Tests

Documentation

Bibliothèque standard

Création de modules

Interface graphiques

Code natif

Python scientifique

# Bibliothèque standard

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

## Bibliothèque standard

Batteries included

Module sys

Module os

Mathématiques

Expressions régulières

Base de données

XML

JSON

Interaction réseau

Archivage des fichiers

Aller plus loin

## Création de modules

Interface  
graphiques

Code natif

Python scientifique

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

**Batteries included**

Module sys

Module os

Mathématiques

Expressions régulières

Base de données

XML

JSON

Interaction réseau

Archivage des fichiers

Aller plus loin

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

## 5- Bibliothèque standard

### 5.1. Batteries included

# “Batteries included”

Python est un langage avec beaucoup de fonctionnalités incluses par défaut

- ▶ gestion de fichiers et des OS (lecture / écriture, compression, diff...)
- ▶ programmation réseau / parralèle / IPC / crypto ...
- ▶ multimédia (images, son, IHM)
- ▶ débuggeur, tests unitaires...
- ▶ gestion des dates, traductions...

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Batteries included

Module sys

Module os

Mathématiques

Expressions régulières

Base de données

XML

JSON

Interaction réseau

Archivage des fichiers

Aller plus loin

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# “Batteries included”

Matthieu Falce

Python est un langage avec beaucoup de fonctionnalités incluses par défaut

- ▶ gestion de fichiers et des OS (lecture / écriture, compression, diff...)
- ▶ programmation réseau / parralèle / IPC / crypto ...
- ▶ multimédia (images, son, IHM)
- ▶ débuggeur, tests unitaires...
- ▶ gestion des dates, traductions...

Il est aussi possible d'installer des modules tiers (très nombreux).

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Batteries included

Module sys

Module os

Mathématiques

Expressions régulières

Base de données

XML

JSON

Interaction réseau

Archivage des fichiers

Aller plus loin

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Batteries included

**Module sys**

Module os

Mathématiques

Expressions régulières

Base de données

XML

JSON

Interaction réseau

Archivage des fichiers

Aller plus loin

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

## 5- Bibliothèque standard

### 5.2. Module sys

# sys

Matthieu Falce

## Manipulation des variables en lien avec l'interpréteur.

```
import sys

# affiche les paramètres passés lors de l'appel du script
# par ex : python gros_calcul.py fichier_entree.mat
print(sys.argv)

# avoir des infos sur les nombres flottants
print(sys.float_info)

# afficher / manipuler le path
print(sys.path)

# afficher l'OS
if sys.platform == "linux":
    print("Ouiiii")
elif sys.platform == "win32":
    print("Oui")

# manipuler les fichiers d'entrée / sortie / erreur
sys.stdin
sys.stdout
sys.stderr

# version de python
# utiliser platform plutôt
if sys.version.startswith("3."):
    print("youpi python3")
else:
    print(":(')

# fermer le programme (optionnel)
sys.exit()
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Batteries included

Module sys

Module os

Mathématiques

Expressions régulières

Base de données

XML

JSON

Interaction réseau

Archivage des fichiers

Aller plus loin

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Batteries included

Module sys

**Module os**

Mathématiques

Expressions régulières

Base de données

XML

JSON

Interaction réseau

Archivage des fichiers

Aller plus loin

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

## 5- Bibliothèque standard

### 5.3. Module os

## Manipulation des variables en lien avec l'OS. Essaie d'avoir la même interface entre les différents OS.

```
import os

# accès aux variables d'environement
print(os.environ)

# permet de modifier le dossier courant
os.chdir()

# lister un dossier
# utiliser "glob" pour les choses plus complexes
os.listdir(".")

# séparateur de fichiers
print(os.sep)

# créer un dossier (et ceux qui manquent entre)
os.makedirs("/tmp/test_os/super_test/", exist_ok=True)

# exécuter une commande
# pour les choses plus compliquées utiliser "subprocess"
commande = "ls /tmp"
os.system(commande)

# compter le nombre de CPU
print(os.cpu_count())
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Batteries included

Module sys

Module os

Mathématiques

Expressions régulières

Base de données

XML

JSON

Interaction réseau

Archivage des fichiers

Aller plus loin

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

## Manipulation des variables en lien avec l'OS. Essaie d'avoir la même interface entre les différents OS.

```
# permet de manipuler les chemins de fichiers
# depuis 3.4 on peut utiliser "pathlib"
# qui est plus haut niveau

from os

# ne pas avoir à manipuler les séparateurs de dossiers
print(os.path.join("/", "tmp", "test_os_path"))

# afficher des parties communes de fichiers
os.path.commonpath(['/usr/lib', '/usr/local/lib'])

# normaliser les chemins
os.path.normpath(
    "/tmp/test_os_path/pas_ici/../../autre_test"
)

# avoir le dernier élément d'un chemin (fichier ?)
path, filename = os.path.split("/tmp/test_os_path/data.csv")

# faire l'expansion de l'utilisateur dans les chemins
expansion = os.path.expanduser(
    os.path.join("~", "test_os_path")
)
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Batteries included

Module sys

Module os

Mathématiques

Expressions régulières

Base de données

XML

JSON

Interaction réseau

Archivage des fichiers

Aller plus loin

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Batteries included

Module sys

Module os

**Mathématiques**

Expressions régulières

Base de données

XML

JSON

Interaction réseau

Archivage des fichiers

Aller plus loin

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

## 5- Bibliothèque standard

### 5.4. Mathématiques

# Outils mathématiques

Matthieu Falce

Ne pas forcément utiliser ceux là pour les calculs scientifiques.

Ils sont plus lents que ceux de numpy / scipy

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Batteries included

Module sys

Module os

Mathématiques

Expressions régulières

Base de données

XML

JSON

Interaction réseau

Archivage des fichiers

Aller plus loin

Création de modules

Interface graphiques

Code natif

Python scientifique

# Outils mathématiques

Matthieu Falce

```
import random, decimal, fractions, statistics

# nbs aléatoires
print(random.randint(1, 20))
print(random.random())

# choisir dans une liste
print("Jean Pierre, la réponse : ", random.choice(["a", "b", "c", "d"]))

# lois aléatoires...
print(random.lognormvariate(mu=10, sigma=2))
data = [random.uniform(1, 10) for _ in range(100)]
print("Moyenne", statistics.mean(data))
print("Ecart type", statistics.stdev(data))

D = decimal.Decimal
F = fractions.Fraction

# calculs exacts
fr = F(16, -10) # simplification
print(fr.numerator) # -8
print(F(1, 3) + F(1, 3) + F(1, 3))

print((1.1 + 2.2 - 3.3) * 1e19) # 4440.89...
print(D("1.1") + D("2.2") - D("3.3")) * int(1e19) # 0
print((D(1.1) + D(2.2) - D(3.3)) * int(1e19)) # 1776.356839
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Batteries included

Module sys

Module os

Mathématiques

Expressions régulières

Base de données

XML

JSON

Interaction réseau

Archivage des fichiers

Aller plus loin

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

## 5- Bibliothèque standard

### 5.5. Expressions régulières

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Batteries included

Module sys

Module os

Mathématiques

**Expressions régulières**

Base de données

XML

JSON

Interaction réseau

Archivage des fichiers

Aller plus loin

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Expressions Régulières ?

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Batteries included

Module sys

Module os

Mathématiques

Expressions régulières

Base de données

XML

JSON

Interaction réseau

Archivage des fichiers

Aller plus loin

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Chaîne de caractères, qui décrit, selon une syntaxe précise,  
un ensemble de chaînes de caractères possibles

---

[https://fr.wikipedia.org/wiki/Expression\\_r%C3%A9gul%C3%A8re](https://fr.wikipedia.org/wiki/Expression_r%C3%A9gul%C3%A8re)

# Syntaxe

Matthieu Falce

- ▶ tous les caractères sont valides
- ▶ quantificateurs (\*, ?, +)
- ▶ opérateur de choix ( $a|b$ ), listes de caractères [aeiou] et inversion de listes [^aeiou] ...
- ▶ caractères spéciaux (début de ligne : ^, fin de ligne : \$)
- ▶ ...

Vous pouvez les tester sur <https://regex101.com>

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Batteries included

Module sys

Module os

Mathématiques

Expressions régulières

Base de données

XML

JSON

Interaction réseau

Archivage des fichiers

Aller plus loin

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Exemples

Matthieu Falce

Expression	Chaînes capturées	Chaînes non capturées
ab	ab	a / b / ""
a b	a / b	ab / c / ...
a+	a / aa / aaaa...aa	"" / ab / b
a?	"" / a	aa / aaa..aa / ab / b
a*	"" / a / aa / aaaa...aa	ab / b
a	*a	tout le reste
[aeiou]	a / e / ...	"" / ae / z

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Batteries included

Module sys

Module os

Mathématiques

Expressions régulières

Base de données

XML

JSON

Interaction réseau

Archivage des fichiers

Aller plus loin

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Exemples

Matthieu Falce

Expression	Chaînes capturées	Chaînes non capturées
[^aeiou]	b / r / ... / 9 / -	"" / a / bc
a{1,3}	a / aa / aaa	tout le reste
[aeiou]	a / e / ...	"" / ae / z
ex-(a?e æ é)quo	ex-equo, ex-aequo, ex-équo et ex-æquo	ex-quo, ex-aquo, ex-aequo, ex-æéquo
^Section .+	Section 1 / Section a / Section a.a/2	"" / Sectionner / voir Section 1
[1234567890]+ (,[1234567890]+)?	2 / 42 / 2,32 / 0.432	3, / ,643 / ""

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Batteries included

Module sys

Module os

Mathématiques

Expressions régulières

Base de données

XML

JSON

Interaction réseau

Archivage des fichiers

Aller plus loin

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Cas d'usages

Matthieu Falce

## Quand les utiliser :

- ▶ traitements complexes
- ▶ tolérance sur des chaînes en entrée
- ▶ si le framework vous y oblige

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Batteries included

Module sys

Module os

Mathématiques

Expressions régulières

Base de données

XML

JSON

Interaction réseau

Archivage des fichiers

Aller plus loin

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Cas d'usages

Matthieu Falce

## Quand les utiliser :

- ▶ traitements complexes
- ▶ tolérance sur des chaînes en entrée
- ▶ si le framework vous y oblige

## Quand ne pas les utiliser :

- ▶ traitements simples (plutôt outils du langage)
- ▶ *parsing* compliqué (plutôt des outils sur des grammaires)

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Batteries included

Module sys

Module os

Mathématiques

Expressions régulières

Base de données

XML

JSON

Interaction réseau

Archivage des fichiers

Aller plus loin

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# En python

Python rajoute des caractères spéciaux pour des cas courants :

- ▶ \w : tous les caractères alphanumériques et underscore ([A-Za-z0-9\_])
- ▶ \W : ni caractères alphanumériques ni underscore (^[A-Za-z0-9\_])
- ▶ \d : chiffres (0-9)
- ▶ \D : autre chose qu'un chiffre (^0-9)
- ▶ \s : séparateur de texte ([\t \r \n \v \f])
- ▶ \S : non séparateur de texte (^[\t \r \n \v \f])
- ▶ \b : début ou fin de mot (attention il FAUT utiliser des "rawstrings" pour que ça marche)

<https://regex101.com> permet d'exporter le code python correspondant à vos expressions

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Batteries included

Module sys

Module os

Mathématiques

Expressions régulières

Base de données

XML

JSON

Interaction réseau

Archivage des fichiers

Aller plus loin

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# En python

Matthieu Falce

```
import re

regex = r"ch?at"
assert re.search(regex, "chat") is not None
assert re.search(regex, "cat") is not None
assert re.search(regex, "chien") is None

# match vs search
assert re.match(regex, "le chat") is None
assert re.search(regex, "le chat") is not None
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Batteries included

Module sys

Module os

Mathématiques

Expressions régulières

Base de données

XML

JSON

Interaction réseau

Archivage des fichiers

Aller plus loin

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# En python

Matthieu Falce

```
import re

regex = "(?P<bien>\w*) c'est bien, (?P<mieux>\w*) c'est mieux"
test_string = "Python c'est bien, Perl c'est mieux"

searched = re.search(regex, test_string)
assert searched.groupdict() == {"bien": "Python", "mieux": "Perl"}

# si la regex ne trouve rien, re.search vaut None
test_string = "Python 2 c'est bien, Python 3 c'est mieux"
assert re.search(regex, test_string) is None

# on modifie la regex pour gérer le nouveau cas
regex = "(?P<bien>[\w\s]*) c'est bien, (?P<mieux>[\w\s]*) c'est mieux"
test_string = "Python 2.7 c'est bien, Python 3.6 c'est mieux"
searched = re.search(regex, test_string)
assert searched.groupdict() == {"bien": "Python 2.7", "mieux": "Python 3.6"}

# comment faire quand il y a plusieurs match dans la chaîne
multiple = re.findall("ch?at", "chat -- dog -- cat")
assert multiple == ["chat", "cat"]

# python_version_pattern = "Python (?P<major>\d*).(?P<minor>\d*)"
# test_string = "Python 2.4 -- Python 3.5 -- Python 0.11 -- Python 32.34224"
# searched = re.findall(regex, test_string)
# assert searched == [('2', '4'), ('3', '5'), ('0', '11'), ('32', '34224')]
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Batteries included

Module sys

Module os

Mathématiques

Expressions régulières

Base de données

XML

JSON

Interaction réseau

Archivage des fichiers

Aller plus loin

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Batteries included

Module sys

Module os

Mathématiques

Expressions régulières

**Base de données**

XML

JSON

Interaction réseau

Archivage des fichiers

Aller plus loin

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

## 5- Bibliothèque standard

### 5.6. Base de données

# Accès aux bases de données

Matthieu Falce

- ▶ Python permet de se connecter à des bases de données
- ▶ Normalisation avec la DB API (database API) <sup>37</sup>
  - ▶ comme un pilote d'imprimante ⇒ on lui dit ce qu'on veut imprimer, il s'occupe des spécificités
  - ▶ augmente la compréhension du code
  - ▶ facilite le changement de SGBD
  - ▶ inspirée de Open Database Connectivity (ODBC) et Java Database Connectivity (JDBC)

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Batteries included

Module sys

Module os

Mathématiques

Expressions régulières

Base de données

XML

JSON

Interaction réseau

Archivage des fichiers

Aller plus loin

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

---

37.<https://www.python.org/dev/peps/pep-0249/>

# Présentation DB API

Matthieu Falce

## Avec SQLite

```
import sqlite3

print("Paramstyle:", sqlite3.paramstyle) # Paramstyle: qmark

# connexion à la base et récupération du curseur
db = sqlite3.connect(':memory:')
cursor = db.cursor()
cursor.execute("""
    CREATE TABLE IF NOT EXISTS users(
        id INTEGER PRIMARY KEY AUTOINCREMENT UNIQUE,
        name TEXT,
        age INTEGER)
""")

# On applique les modifications avec commit
db.commit()

cursor.execute("""INSERT INTO users(name, age) VALUES(?, ?)""", ("matthieu", 323))
db.commit()

cursor.execute(''':SELECT * FROM users;''')
# récupérer le premier
user1 = cursor.fetchone()
print(user1) # (1, 'matthieu', 323)

# on ferme tout à la fin
cursor.close()
db.close()
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Batteries included

Module sys

Module os

Mathématiques

Expressions régulières

Base de données

XML

JSON

Interaction réseau

Archivage des fichiers

Aller plus loin

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Présentation DB API

## Avec Mysql

```
# avant d'installer avec pip faire: sudo apt install libmysqlclient-dev
# sur windows, il y a un wheel avec les bons binaires
import MySQLdb

print("Paramstyle:", MySQLdb.paramstyle) # Paramstyle: format

# connexion à la base et récupération du curseur
# pas de mot de passe et compte root de MySQL, ne faites pas ça...
db = MySQLdb.connect(host="127.0.0.1", user="root", db="formation")
cursor=db.cursor()
cursor.execute("""
    CREATE TABLE IF NOT EXISTS users(
        id INTEGER PRIMARY KEY AUTO_INCREMENT UNIQUE,
        name TEXT,
        age INTEGER);
""")

# On applique les modifications avec commit
db.commit()

cursor.execute("""INSERT INTO users(name, age) VALUES(%s, %s);""", ("matthieu", 323))
db.commit()

cursor.execute(''':SELECT * FROM users;''')
# récupérer le premier
user1 = cursor.fetchone()
print(user1) # (1, 'matthieu', 323)

# on ferme tout à la fin
cursor.close()
db.close()
```

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Batteries included

Module sys

Module os

Mathématiques

Expressions régulières

Base de données

XML

JSON

Interaction réseau

Archivage des fichiers

Aller plus loin

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Présentation DB API

Matthieu Falce

## En résumé

- ▶ même structure et méthodes appelées
- ▶ différence de syntaxe des paramètres
- ▶ différences au niveau du SQL supporté...
- ▶ si l'on ne commite pas on ne stocke pas les données en base
  - ▶ curseurs globaux à une connexion ⇒ données potentiellement non enregistrées accessibles

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Batteries included

Module sys

Module os

Mathématiques

Expressions régulières

Base de données

XML

JSON

Interaction réseau

Archivage des fichiers

Aller plus loin

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Insérer / récupérer des données

Matthieu Falce

```
import sqlite3

db = sqlite3.connect(':memory:')
cursor = db.cursor()
cursor.execute("""
    CREATE TABLE IF NOT EXISTS users(
        id INTEGER PRIMARY KEY AUTOINCREMENT UNIQUE, name TEXT, age INTEGER)
""")
db.commit()

# insérer des données en mode batch
users = [
    ("olivier", 30), ("jean-louis", 90), ("luc", 32),
    ("matthieu", 24), ("pierre", 54), ("françois", 78)
]
cursor.executemany("""
    INSERT INTO users(name, age) VALUES(?, ?)""", users)

# récupérer toutes les données
print("----- Tous -----")
cursor.execute("""SELECT id, name, age FROM users""")
rows = cursor.fetchall()
for row in rows:
    print('{0} : {1} - {2}'.format(row[0], row[1], row[2]))

# récupérer une sélection les données
print("----- Selection -----")
cursor.execute("""SELECT id, name, age FROM users WHERE age > 30""")
for row in cursor.fetchall():
    print('{0} : {1} - {2}'.format(row[0], row[1], row[2]))
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Batteries included

Module sys

Module os

Mathématiques

Expressions régulières

Base de données

XML

JSON

Interaction réseau

Archivage des fichiers

Aller plus loin

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Supprimer / mettre à jour des données

Matthieu Falce

```
import sqlite3

db = sqlite3.connect(':memory:')
cursor = db.cursor()
cursor.execute("""
    CREATE TABLE IF NOT EXISTS users(
        id INTEGER PRIMARY KEY AUTOINCREMENT UNIQUE, name TEXT, age INTEGER)
""")
db.commit()

# insérer des données en mode batch
users = [
    ("olivier", 30), ("jean-louis", 90), ("luc", 32),
    ("matthieu", 24), ("pierre", 54), ("françois", 78)
]
cursor.executemany("""INSERT INTO users(name, age) VALUES(?, ?)""", users)
db.commit()

# on va modifier les jeunes pour leur rajouter un préfixe
# || pour concaténer des chaînes en SQLite
cursor.execute("""UPDATE users SET name = name || ' Jr' WHERE age < 30 ;""")
db.commit()

# on va supprimer les gens qui ont un nom de plus de 5 caractères
cursor.execute("""DELETE FROM users WHERE length(name)>6 ;""")
db.commit()
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Batteries included

Module sys

Module os

Mathématiques

Expressions régulières

Base de données

XML

JSON

Interaction réseau

Archivage des fichiers

Aller plus loin

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Erreurs et exceptions

Matthieu Falce

## Taxonomie des exceptions d'après la PEP 249

`StandardError`

`|__Warning`

`|__Error`

`|__InterfaceError`

`|__DatabaseError`

`|__DataError`

`|__OperationalError`

`|__IntegrityError`

`|__InternalError`

`|__ProgrammingError`

`|__NotSupportedError`

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Batteries included

Module sys

Module os

Mathématiques

Expressions régulières

Base de données

XML

JSON

Interaction réseau

Archivage des fichiers

Aller plus loin

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Erreurs et exceptions

Matthieu Falce

## Quelles données en base à la fin du script ?

```
import sqlite3

db = sqlite3.connect('/tmp/test.db')
cursor = db.cursor()
cursor.execute("""
    CREATE TABLE IF NOT EXISTS users(
        id INTEGER PRIMARY KEY AUTOINCREMENT UNIQUE, name TEXT UNIQUE, age INTEGER)
""")
db.commit()

# utilisateurs avec des noms identiques
users = [("matthieu", 30), ("matthieu", 90)]

try:
    for user in users:
        cursor.execute("""INSERT INTO users(name, age) VALUES(?, ?)""", user)
except sqlite3.IntegrityError as e:
    print("Integrity Error, roll back")
    db.rollback()
finally:
    # Close the db connection
    db.commit()
    db.close()
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Batteries included

Module sys

Module os

Mathématiques

Expressions régulières

Base de données

XML

JSON

Interaction réseau

Archivage des fichiers

Aller plus loin

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Erreurs et exceptions

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Batteries included

Module sys

Module os

Mathématiques

Expressions régulières

Base de données

XML

JSON

Interaction réseau

Archivage des fichiers

Aller plus loin

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

```
import sqlite3

db = sqlite3.connect('/tmp/test.db')
cursor = db.cursor()
cursor.execute("""
    CREATE TABLE IF NOT EXISTS users(
        id INTEGER PRIMARY KEY AUTOINCREMENT UNIQUE, name TEXT UNIQUE, age INTEGER)
""")
db.commit()

# utilisateurs avec des noms identiques
users = [("matthieu", 30), ("matthieu", 90)]

try:
    for user in users:
        cursor.execute("""INSERT INTO users(name, age) VALUES(?, ?)""", user)
        db.commit()
except sqlite3.IntegrityError as e:
    print("Integrity Error, roll back")
    db.rollback()
finally:
    # Close the db connection
    db.commit()
    db.close()
```

## Quelles données en base à la fin du script ?

# Bibliographie / Aller plus loin

Matthieu Falce

- ▶ <https://wiki.python.org/moin/DbApiCheatSheet>
- ▶ <http://sweetohm.net/article/python-dbapi.html>
- ▶ <https://apprendre-python.com/page-database-database-donnees-query-sql-mysql-postgre-sqlite>
- ▶ <https://www.sqlitetutorial.net/sqlite-python/>
- ▶ comment gérer le *multithreading* ?
  - ▶ curseurs non *thread safe*
  - ▶ une connexion par thread
- ▶ ORM<sup>38</sup> ⇒ abstraire les différences entre moteurs
  - ▶ SQLAlchemy
  - ▶ Pewee
  - ▶ PonyORM
  - ▶ ORM Django

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Batteries included

Module sys

Module os

Mathématiques

Expressions régulières

Base de données

XML

JSON

Interaction réseau

Archivage des fichiers

Aller plus loin

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

38.<https://www.fullstackpython.com/object-relational-mappers-orms.html>

## 5- Bibliothèque standard

### 5.7. XML

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Batteries included

Module sys

Module os

Mathématiques

Expressions régulières

Base de données

**XML**

JSON

Interaction réseau

Archivage des fichiers

Aller plus loin

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

```
import xml.etree.cElementTree as ET

# écriture
root = ET.Element("root")
doc = ET.SubElement(root, "doc")

ET.SubElement(doc, "field1", name="blah").text = "some value1"
ET.SubElement(doc, "field2", name="asdfasd").text = "some value2"

tree = ET.ElementTree(root)
tree.write("filename.xml")
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Batteries included

Module sys

Module os

Mathématiques

Expressions régulières

Base de données

XML

JSON

Interaction réseau

Archivage des fichiers

Aller plus loin

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

```
import io
from xml.dom import minidom

# lecture d'un XML

data = """
<data> <items>
    <item name="item1"></item> <item name="item2"></item>
    <item name="item3"></item> <item name="item4"></item>
</items></data>"""

# parse attend un fichier, on crée un StringIO pour le duper

file_like_from_str = io.StringIO(data)
xmldoc = minidom.parse(file_like_from_str)
itemlist = xmldoc.getElementsByTagName('item')
print(len(itemlist))
print(itemlist[0].attributes['name'].value)
for s in itemlist:
    print(s.attributes['name'].value)
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Batteries included

Module sys

Module os

Mathématiques

Expressions régulières

Base de données

XML

JSON

Interaction réseau

Archivage des fichiers

Aller plus loin

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

## 5- Bibliothèque standard

### 5.8. JSON

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Batteries included

Module sys

Module os

Mathématiques

Expressions régulières

Base de données

XML

**JSON**

Interaction réseau

Archivage des fichiers

Aller plus loin

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# JSON

Matthieu Falce

```
import json

# créer un JSON
donnees_test = {
    "chaine": "dictionnaire",
    "liste": [1, 2, 3]
}

# crée le fichier test.json
json.dump(donnees_test, open("test.json", "w"))

# stocke le résultat dans une chaîne
representation_json = json.dumps(donnees_test)

# lire un json

# depuis un fichier
data = json.load(open("test.json"))

# depuis une chaîne
data2 = json.loads(representation_json)

assert data == donnees_test
assert data2 == donnees_test
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Batteries included

Module sys

Module os

Mathématiques

Expressions régulières

Base de données

XML

JSON

Interaction réseau

Archivage des fichiers

Aller plus loin

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique



Certaines données ne sont pas JSON sérialisables. Il faut créer son propre serialiseur JSON dans ce cas. <sup>40</sup>

```
from json import dumps
from datetime import date, datetime

def json_serial(obj):
    """JSON serializer for objects not serializable
    by default json code"""
    if isinstance(obj, (datetime, date)):
        return obj.isoformat()
    raise TypeError("Type %s not serializable" % type(obj))

print(dumps(datetime.now(), default=json_serial))
```

---

40. <https://stackoverflow.com/questions/11875770/how-to-overcome-datetime-datetime-not-json-serializable>

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Batteries included

Module sys

Module os

Mathématiques

Expressions régulières

Base de données

XML

JSON

Interaction réseau

Archivage des fichiers

Aller plus loin

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# CSV – excel

Matthieu Falce

```
#####
# version quick and dirty

# écrire
data = [[1, 2], [3, 4, 5]]
open("eggs.csv", "w").write(
    "\n".join(["\t".join(map(str, line)) for line in data])
)

# lire
data = [line.strip().split("\t") for line in open("eggs.csv", "r")]
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Batteries included

Module sys

Module os

Mathématiques

Expressions régulières

Base de données

XML

JSON

Interaction réseau

Archivage des fichiers

Aller plus loin

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# CSV – excel

Matthieu Falce

```
import csv

# écrire le fichier

data = [
    ["Spam"] * 5 + ["Baked Beans"],
    ['Spam', 'Lovely Spam', 'Wonderful Spam'],
    ["Avec des accents éàù", "ça marche"]
]

with open('eggs.csv', 'w') as csvfile:
    spamwriter = csv.writer(
        csvfile, delimiter=' ', quotechar='|', quoting=csv.QUOTE_MINIMAL
    )
    for row in data:
        spamwriter.writerow(row)

# lire le fichier
with open('eggs.csv', 'r') as csvfile:
    spamreader = csv.reader(csvfile, delimiter=' ', quotechar='|')
    for row in spamreader:
        print(', '.join(row))
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Batteries included

Module sys

Module os

Mathématiques

Expressions régulières

Base de données

XML

JSON

Interaction réseau

Archivage des fichiers

Aller plus loin

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# CSV – excel

Matthieu Falce

On peut utiliser xlrd, openpyxl ou pandas (qui se base sur ces dernières) <sup>41</sup>

```
# pip install pandas xlrd openpyxl
import pandas as pd

xl = pd.ExcelFile("./fichiers_a_lire/excel_plusieurs_feuilles.xlsx")
names = xl.sheet_names

df = xl.parse(names[0])
df2 = xl.parse(names[1])
print(df.head())
print(df2.head())

df = pd.read_excel("./fichiers_a_lire/excel_une_feuille.xlsx")
print(df.head())

# écrire
df.to_excel(
    'fichiers_a_lire/test.xlsx',
    sheet_name='sheet1',
    index=False
)
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Batteries included

Module sys

Module os

Mathématiques

Expressions régulières

Base de données

XML

JSON

Interaction réseau

Archivage des fichiers

Aller plus loin

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

---

41. <http://www.python-excel.org/>

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Batteries included

Module sys

Module os

Mathématiques

Expressions régulières

Base de données

XML

JSON

**Interaction réseau**

Archivage des fichiers

Aller plus loin

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

## 5- Bibliothèque standard

### 5.9. Interaction réseau

# Web – http

Matthieu Falce

## Avec la lib standard

```
# pip install requests
import urllib.request
import urllib.parse
import pprint, json

urlopen = urllib.request.urlopen

url = 'https://httpbin.org/anything'

values = {
    'name': 'Michael Foord',
    'location': 'Northampton',
    'language': 'Python'
}

# requête get simple
with urlopen(url) as response:
    pprint.pprint(json.loads(response.read()))

# GET avec paramètres
url_values = urllib.parse.urlencode(values)
full_url = url + '?' + url_values
with urlopen(full_url) as response:
    pprint.pprint(json.loads(response.read()))

# requête post avec paramètres
data = urllib.parse.urlencode(values)
data = data.encode('ascii') # data should be bytes
req = urllib.request.Request(url, data)
with urlopen(req) as response:
    pprint.pprint(json.loads(response.read()))
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Batteries included

Module sys

Module os

Mathématiques

Expressions régulières

Base de données

XML

JSON

Interaction réseau

Archivage des fichiers

Aller plus loin

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

## Avec requests

```
# pip install requests
import requests
import pprint

url = 'https://httpbin.org/anything'

values = {
    'name': 'Michael Foord',
    'location': 'Northampton',
    'language': 'Python'
}

# requête GET simple
r = requests.get(url)
pprint.pprint(r.json())

# requête GET avec paramètres
r = requests.get(url, data=values)
pprint.pprint(r.json())

# requête POST avec paramètres
r = requests.post(url, data=values)
pprint.pprint(r.json())
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Batteries included

Module sys

Module os

Mathématiques

Expressions régulières

Base de données

XML

JSON

Interaction réseau

Archivage des fichiers

Aller plus loin

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

	Vue d'ensemble
	Langage Python
	Programmation Orientée objet (POO)
	Bonnes pratiques
	Bibliothèque standard
Batteries included	
Module sys	
Module os	
Mathématiques	
Expressions régulières	
Base de données	
XML	
JSON	
Interaction réseau	
Archivage des fichiers	
Aller plus loin	
Création de modules	
Interface graphiques	
Code natif	
Python scientifique	

# Sockets

Matthieu Falce

```
# Requête HTTP à la main

# exemple socket client
import socket

HOST = 'google.com'      # The remote host
PORT = 80                 # The same port as used by the server
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((HOST, PORT))
    s.sendall(
        b"GET / HTTP/1.1\r\nHost: google.com\r\n\r\n"
    )
    data = s.recv(1024)
print('Received', repr(data))

=====
# Echo server program
# test avec `echo -en "1\n2\n" | nc localhost 50007 -ql`
HOST = ''                  # Symbolic name meaning all available interfaces
PORT = 50007                # Arbitrary non-privileged port
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind((HOST, PORT))
    s.listen(1)
    conn, addr = s.accept()
    with conn:
        print('Connected by', addr)
        while True:
            data = conn.recv(1024)
            if not data:
                break
            conn.sendall(data)
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Batteries included

Module sys

Module os

Mathématiques

Expressions régulières

Base de données

XML

JSON

Interaction réseau

Archivage des fichiers

Aller plus loin

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Batteries included

Module sys

Module os

Mathématiques

Expressions régulières

Base de données

XML

JSON

Interaction réseau

**Archivage des fichiers**

Aller plus loin

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

## 5- Bibliothèque standard

### 5.10. Archivage des fichiers

# Manipulation de fichiers (archivage)

Matthieu Falce

```
import zipfile

# créer une archive
filename = "test_zip.py"
with zipfile.ZipFile('example.zip', mode='w') as zf:
    print('adding ', filename)
    zf.write(filename)

# lister les fichiers d'une archive
with zipfile.ZipFile('example.zip', 'r') as zf:
    print(zf.namelist())

# extraire les fichiers d'une archive
with zipfile.ZipFile('example.zip') as zf:
    for filename in [filename, 'notthere.txt']:
        try:
            data = zf.read(filename)
        except KeyError:
            print('ERROR: Did not find {} in zip file'.format(
                filename))
        else:
            print(filename, ':')
            print(data)
    print()
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Batteries included

Module sys

Module os

Mathématiques

Expressions régulières

Base de données

XML

JSON

Interaction réseau

Archivage des fichiers

Aller plus loin

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Manipulation de fichiers (archivage)

Matthieu Falce

```
from shutil import make_archive
import os

archive_name = os.path.expanduser(os.path.join('~', 'myarchive'))
root_dir = os.path.expanduser(os.path.join('~', '.ssh'))
make_archive(archive_name, 'gztar', root_dir)
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Batteries included

Module sys

Module os

Mathématiques

Expressions régulières

Base de données

XML

JSON

Interaction réseau

Archivage des fichiers

Aller plus loin

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Batteries included

Module sys

Module os

Mathématiques

Expressions régulières

Base de données

XML

JSON

Interaction réseau

Archivage des fichiers

Aller plus loin

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

## 5- Bibliothèque standard

### 5.11. Aller plus loin

# Autres modules intéressants I

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Batteries included

Module sys

Module os

Mathématiques

Expressions régulières

Base de données

XML

JSON

Interaction réseau

Archivage des fichiers

Aller plus loin

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

La librairie standard regorge de modules intéressants en plus des précédents (“python is batteries included”).

# Autres modules intéressants II

Matthieu Falce

En voici quelques un :

- ▶ `copy` : copie les objets, récursivement (utile pour les conteneurs et objets custom)
- ▶ `logging` : permet d'effectuer le logging des applications.  
Extrêmement complet
- ▶ `datetime` : permet de gérer les dates (additions, parsing...), des alternatives tierces existent pour les cas complexes
- ▶ `argparse` : permet de gérer les arguments en ligne de commande (des alternatives tierces plus complètes existent)
- ▶ `functools` : permet de manipuler les fonctions d'ordres supérieurs
- ▶ `itertools` : permet de manipuler les itérables et de faciliter les constructions paresseuses

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Batteries included

Module sys

Module os

Mathématiques

Expressions régulières

Base de données

XML

JSON

Interaction réseau

Archivage des fichiers

Aller plus loin

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

# Création de modules

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

**Création de  
modules**

Setuptools / Distutils

Structure d'un module

Installation

Mise en ligne PyPI

Programmes autonomes

Bibliographie

Interface  
graphiques

Code natif

Python scientifique

## 6- Création de modules

### 6.1. Setuptools / Distutils

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Setuptools / Distutils

Structure d'un module

Installation

Mise en ligne PyPI

Programmes autonomes

Bibliographie

Interface  
graphiques

Code natif

Python scientifique

# SetupTools / Distutils

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

SetupTools / Distutils

Structure d'un module

Installation

Mise en ligne PyPI

Programmes autonomes

Bibliographie

Interface  
graphiques

Code natif

Python scientifique

Permettent d'empaqueter un module python.

Coexistence de **SetupTools** et **Distutils** → confusion

# Setuptools / Distutils

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Setuptools / Distutils

Structure d'un module

Installation

Mise en ligne PyPI

Programmes autonomes

Bibliographie

Interface  
graphiques

Code natif

Python scientifique

Permettent d'empaqueter un module python.

Coexistence de Setuptools et Distutils → confusion

On va utiliser setuptools

[https:](https://)

//stackoverflow.com/questions/25337706/setuptools-vs-distutils-why-is-distutils-still-a-thing

# Setuptools / Distutils

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Setuptools / Distutils

Structure d'un module

Installation

Mise en ligne PyPI

Programmes autonomes

Bibliographie

Interface  
graphiques

Code natif

Python scientifique

Permettent d'empaqueter un module python.

Coexistence de Setuptools et Distutils → confusion

On va utiliser setuptools

[https:](https://)

//stackoverflow.com/questions/25337706/setuptools-vs-distutils-why-is-distutils-still-a-thing

Arborescence et fichiers spécifiques à respecter

## 6- Création de modules

### 6.2. Structure d'un module

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Setupools / Distutils

**Structure d'un module**

Installation

Mise en ligne PyPI

Programmes autonomes

Bibliographie

Interface  
graphiques

Code natif

Python scientifique

# Structure d'un module

Code (`__init__.py`) :

```
def joke():
    return (
        'Tu connais la blague du petit dej ? \n'
        'Pas de bol'
    )
```

Arborescence :

```
minimal/
    funniest/
        __init__.py
    setup.py
```

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Setuptools / Distutils

Structure d'un module

Installation

Mise en ligne PyPI

Programmes autonomes

Bibliographie

Interface  
graphiques

Code natif

Python scientifique

# Structure d'un module

Code (`__init__.py`) :

```
def joke():
    return (
        'Tu connais la blague du petit dej ? \n'
        'Pas de bol'
    )
```

Arborescence :

```
minimal/
    funniest/
        __init__.py
    setup.py
```



Matthieu Falce

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Setuptools / Distutils

Structure d'un module

Installation

Mise en ligne PyPI

Programmes autonomes

Bibliographie

Interface  
graphiques

Code natif

Python scientifique

dossier du projet		minimal/
dossier du code		funniest/
code du programme		__init__.py
configuration		setup.py

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Setuptools / Distutils

Structure d'un module

Installation

Mise en ligne PyPI

Programmes autonomes

Bibliographie

Interface  
graphiques

Code natif

Python scientifique

## 6- Création de modules

### 6.3. Installation

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Setuptools / Distutils

Structure d'un module

**Installation**

Mise en ligne PyPI

Programmes autonomes

Bibliographie

Interface  
graphiques

Code natif

Python scientifique

# Installation

Matthieu Falce

## Contenu de setup.py

```
from setuptools import setup

setup(
    name='funniest',
    version='0.1',
    description='Super blague',
    url='http://example.com/ahahah/',
    author='Matthieu Falce',
    author_email='clown@example.com',
    license='MIT',
    packages=['funniest'],
    zip_safe=False
)
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Setuptools / Distutils

Structure d'un module

Installation

Mise en ligne PyPI

Programmes autonomes

Bibliographie

Interface  
graphiques

Code natif

Python scientifique

## Installation avec pip

PIP va se charger des copies → endroits connus dans le  
PYTHONPATH

```
# on se place à l'endroit du setup.py
pip install .      # mode normal

# mode ''édition'': évite de réinstaller en continue
# quand on modifie le module
pip install -e .
```

## Utilisation du module

Depuis n'importe où sur l'ordinateur

```
import funniest
print(funniest.joke)
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Setuptools / Distutils

Structure d'un module

Installation

Mise en ligne PyPI

Programmes autonomes

Bibliographie

Interface  
graphiques

Code natif

Python scientifique

# Exemple plus complet

Matthieu Falce

- ▶ déclaration des dépendances
- ▶ programme plus gros
- ▶ utilisation de fichiers “autres” (données...)
- ▶ points d'entrées

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Setuptools / Distutils

Structure d'un module

Installation

Mise en ligne PyPI

Programmes autonomes

Bibliographie

Interface  
graphiques

Code natif

Python scientifique

# Exemple plus complet

Arborescence :

```
tree complet --charset=ASCII -I "__pycache__"
```

```
complet
|-- funniest
|   |-- command_line.py
|   |-- data
|   |   '-- blagues.txt
|   |   '-- __init__.py
|   |   '-- texput.log
|   '-- text.py
|-- MANIFEST.in
`-- setup.py
```

2 directories, 7 files

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Setuptools / Distutils

Structure d'un module

Installation

Mise en ligne PyPI

Programmes autonomes

Bibliographie

Interface  
graphiques

Code natif

Python scientifique

# Exemple plus complet

Matthieu Falce

## Contenu de `text.py` :

```
from pathlib import Path
from markdown import markdown
import os

# test lecture fichiers de données
module_path = Path(
    os.path.dirname(os.path.abspath(__file__)))
)
print(open(module_path / "data/blagues.txt").readlines())

def joke():
    return markdown(
        'Tu connais la *blague du petit dej* ? \n'
        '_Pas de bol_'
    )
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Setuptools / Distutils

Structure d'un module

Installation

Mise en ligne PyPI

Programmes autonomes

Bibliographie

Interface  
graphiques

Code natif

Python scientifique

# Exemple plus complet

Matthieu Falce

Contenu de  
`__init__.py` :

```
from .text import joke
```

Contenu de `blagues.txt` :

"Un jour Dieu dit à Casto de ramer. Et depuis, castorama..."

Contenu de `MANIFEST.in` (pour les fichiers statiques) :

```
# Include the data files
include funniest/data/blagues.txt
```

Contenu de  
`command_line.py` :

```
import funniest

def main():
    print(funniest.joke())
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Setuptools / Distutils

Structure d'un module

Installation

Mise en ligne PyPI

Programmes autonomes

Bibliographie

Interface  
graphiques

Code natif

Python scientifique

# Exemple plus complet

Matthieu Falce

## Contenu de setup.py :

```
from setuptools import setup

setup(
    name='funniest_bLyR4',
    version='0.1',
    description='Super blague / Utilisé dans des formations',
    url='http://example.com/ahahah/',
    author='Matthieu Falce',
    author_email='clown@example.com',
    packages=['funniest'],
    license='MIT',

    install_requires=[
        'markdown',
    ],
    entry_points = { # commandes qui seront installées
        'console_scripts': [
            'funniest-joke=funniest.command_line:main'
        ],
    },
    include_package_data=True,
    zip_safe=False
)
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Setuptools / Distutils

Structure d'un module

Installation

Mise en ligne PyPI

Programmes autonomes

Bibliographie

Interface  
graphiques

Code natif

Python scientifique

## 6- Création de modules

### 6.4. Mise en ligne PyPI

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Setuptools / Distutils

Structure d'un module

Installation

**Mise en ligne PyPI**

Programmes autonomes

Bibliographie

Interface  
graphiques

Code natif

Python scientifique

# Qui / Où ?

Matthieu Falce

- ▶ tout le monde peut téléverser sur PyPI
- ▶ il y a une plateforme de test : <https://test.pypi.org/> (que nous allons utiliser)
- ▶ il faut créer un compte et le valider

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Setuptools / Distutils

Structure d'un module

Installation

Mise en ligne PyPI

Programmes autonomes

Bibliographie

Interface  
graphiques

Code natif

Python scientifique

# Qui / Où ?

Matthieu Falce

On peut créer un fichier qui va contenir le mot de passe du compte (pour ne pas le retaper).

A placer dans `~/.pypirc`

```
[distutils]
index-servers=
    testpypi
    pypi

[testpypi]
repository = https://test.pypi.org/legacy/
username = name_of_the_user
password = hunter2

[pypi]
repository = https://pypi.python.org/pypi
username = name_of_the_user
password = hunter2
```

Source : <https://blog.jetbrains.com/pycharm/2017/05/how-to-publish-your-package-on-pypi/>

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Setuptools / Distutils

Structure d'un module

Installation

Mise en ligne PyPI

Programmes autonomes

Bibliographie

Interface  
graphiques

Code natif

Python scientifique

# Pas à pas

Matthieu Falce

```
# on installe twine qui va servir à faire l'upload
pip install twine

# on construit les sdist et bdist_wheel
python setup.py sdist bdist_wheel

# on upload tout dist avec twine, en prenant la
# configuration de testpypi
twine upload -r testpypi dist/*

# on peut installer dans un autre venv
# j'ai dû changer le nom du projet pour pouvoir l'uploader
pip install \
    --index-url https://test.pypi.org/simple/ \
    --default-timeout=100
funniest_bLyR4
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Setuptools / Distutils

Structure d'un module

Installation

Mise en ligne PyPI

Programmes autonomes

Bibliographie

Interface  
graphiques

Code natif

Python scientifique

# 6- Création de modules

## 6.5. Programmes autonomes

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Setuptools / Distutils

Structure d'un module

Installation

Mise en ligne PyPI

**Programmes autonomes**

Bibliographie

Interface  
graphiques

Code natif

Python scientifique

Ce que nous avons vu jusqu'à présent ne permet pas d'avoir des programmes autonomes :

- ▶ ce n'est bien que pour les développeurs
- ▶ il faut avoir python installé sur la machine
- ▶ il faut installer les dépendances

Comment distribuer à des utilisateurs finaux ?

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Setuptools / Distutils

Structure d'un module

Installation

Mise en ligne PyPI

Programmes autonomes

Bibliographie

Interface  
graphiques

Code natif

Python scientifique

Il existe des outils permettant de créer des "exécutables" à partir de scripts python

- ▶ pas besoin d'installer python / les dépendances séparément
- ▶ peuvent être installés sur l'OS
- ▶ cependant la solution est plus lourde que de distribuer des modules

Plusieurs outils existent :

- ▶ **pyinstaller**
- ▶ **cx\_freeze**
- ▶ **py2exe**

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Setuptools / Distutils

Structure d'un module

Installation

Mise en ligne PyPI

Programmes autonomes

Bibliographie

Interface  
graphiques

Code natif

Python scientifique

# Pyinstaller

Matthieu Falce

L'outil le plus simple et que je conseille :  
<https://pyinstaller.org/en/stable/>

- ▶ fonctionne sous mac, Linux, windows
- ▶ détecte les dépendances automatiquement

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Setuptools / Distutils

Structure d'un module

Installation

Mise en ligne PyPI

Programmes autonomes

Bibliographie

Interface  
graphiques

Code natif

Python scientifique

- Vue d'ensemble
  - Langage Python
  - Programmation Orientée objet (POO)
  - Bonnes pratiques
  - Bibliothèque standard
  - Création de modules
    - Setuptools / Distutils
    - Structure d'un module
    - Installation
    - Mise en ligne PyPI
    - Programmes autonomes
    - Bibliographie
  - Interface graphiques
  - Code natif
  - Python scientifique
- Utiliser pyinstaller :
- ▶ simplement : `pyinstaller myscript.py`
  - ▶ si plus complexe : `pyinstaller myscript.spec` (le fichier `.spec` permet de configurer l'exécutable comme on le souhaite :  
<https://pyinstaller.org/en/stable/spec-files.html>)

## Utiliser pyinstaller :

- ▶ simplement : `pyinstaller myscript.py`
- ▶ si plus complexe : `pyinstaller myscript.spec` (le fichier `.spec` permet de configurer l'exécutable comme on le souhaite :

<https://pyinstaller.org/en/stable/spec-files.html>



Quand le script essaie d'accéder à des fichiers relatifs, il faut faire attention aux chemins relatifs (plus d'informations ici :  
<https://pyinstaller.org/en/stable/runtime-information.html>)

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Setuptools / Distutils

Structure d'un module

Installation

Mise en ligne PyPI

Programmes autonomes

Bibliographie

Interface  
graphiques

Code natif

Python scientifique

## 6- Création de modules

### 6.6. Bibliographie

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Setupools / Distutils

Structure d'un module

Installation

Mise en ligne PyPI

Programmes autonomes

**Bibliographie**

Interface  
graphiques

Code natif

Python scientifique

# Bibliographie I

Matthieu Falce

- ▶ Packaging / installation
  - ▶ Informations packaging officielles
  - ▶ Exemple officiel (PyPA) de setup.py
  - ▶ Exemple de packaging de A à Z
  - ▶ Exemple d'utilisation de pyinstaller de A à Z

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Setuptools / Distutils

Structure d'un module

Installation

Mise en ligne PyPI

Programmes autonomes

Bibliographie

Interface  
graphiques

Code natif

Python scientifique

# Interface graphiques

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

**Interface  
graphiques**

QT

Conclusion

Code natif

Python scientifique

# 7- Interface graphiques

## 7.1. QT

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

QT

Contexte

Exemples

Conclusion

Code natif

Python scientifique

# Contexte

Matthieu Falce

Qt (prononcé officiellement en anglais cute mais couramment prononcé Q.T.) est une API orientée objet et développée en C++, conjointement par The Qt Company et Qt Project. Qt offre des composants d'interface graphique (widgets), d'accès aux données, de connexions réseaux, de gestion des fils d'exécution, d'analyse XML, etc. Par certains aspects, elle ressemble à un framework lorsqu'on l'utilise pour concevoir des interfaces graphiques ou que l'on conçoit l'architecture de son application en utilisant les mécanismes des signaux et slots par exemple.

---

<https://fr.wikipedia.org/wiki/Qt>

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

QT

Contexte

Exemples

Conclusion

Code natif

Python scientifique

# Contexte

Matthieu Falce

- ▶ développé en C++ avec des bindings dans de nombreux langages
- ▶ utilise fortement l'orienté objet pour décrire une arborescence (entre autres) de widgets
- ▶ Qt a un système de licence assez particulier (à considérer pour des applications propriétaires)
- ▶ a 2 bindings python : pyside (maintenue par RiverBank Computing) et pyqt (maintenu par Nokia), la différence tient principalement à la licence des bibliothèques (autres différences ici :  
<https://www.pythonguis.com/faq/pyqt5-vs-pyside2/>)
- ▶ Qt utilise un mécanisme particulier pour faire communiquer ses éléments : les signaux et les slots
- ▶ Qt permet d'avoir des outils de prototypage rapide pour construire facilement des interfaces graphiques visuellement

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

QT

Contexte

Exemples

Conclusion

Code natif

Python scientifique

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

QT

Contexte

Exemples

Conclusion

Code natif

Python scientifique

Une nouvelle version majeure de Qt est sortie en 2021 : Qt6. Il y a des différences entre Qt5 et Qt6 et donc également dans les versions Python. Cette page liste les modifications à effectuer :

<https://www.pythonguis.com/faq/pyqt5-vs-pyqt6/>.

Par quoi commencer ?

- ▶ Les ressources sont plus nombreuses avec Qt5 pour l'instant.
- ▶ je recommande de commencer avec la version Qt5, puis, une fois habitué, passer à Qt6 en faisant les changements.

# Signaux et slots

Matthieu Falce

- ▶ mécanisme central de QT et absent des autres frameworks graphiques
- ▶ système de communication entre les objets
- ▶ permet d'organiser proprement un ensemble de callbacks
- ▶ un signal est émis pour signaler un événement, un slot est la fonction qui est appelée lors de cet événement (il peut y en avoir plusieurs), le mécanisme de lien entre les 2 est la connexion
- ▶ les objets Qt viennent avec leurs propres signaux / slots, mais on peut en rajouter

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

QT

Contexte

Exemples

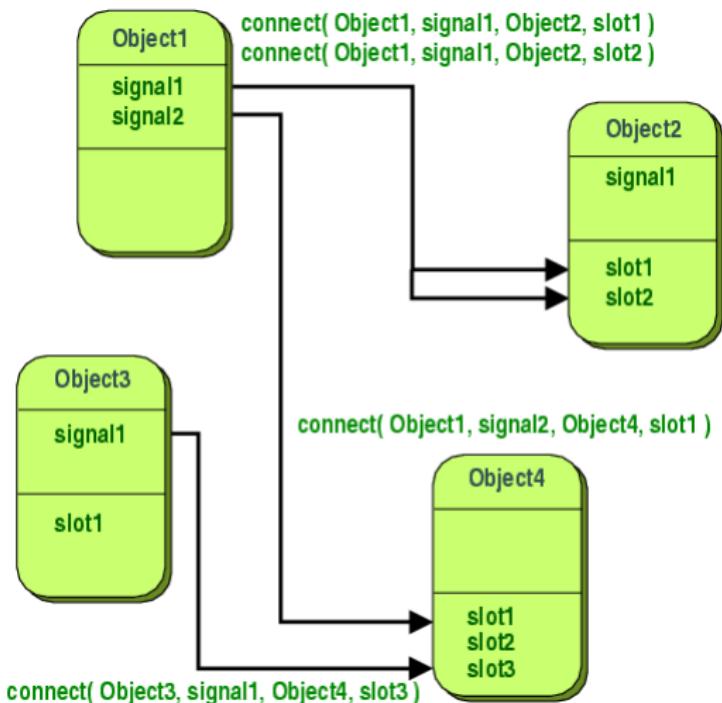
Conclusion

Code natif

Python scientifique

# Signaux et slots

Matthieu Falce



Mécanisme de communication entre objets (source :  
<https://doc.qt.io/qt-5/signalsandslots.html>)

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

QT

Contexte

Exemples

Conclusion

Code natif

Python scientifique

# Exemples de code

Matthieu Falce



# Exemples de code

Matthieu Falce

```
# Source : https://www.pythonguis.com/tutorials/pyqt-signals-slots-events/  
  
import sys  
from PyQt5.QtWidgets import QApplication, QMainWindow, QPushButton  
  
  
class MainWindow(QMainWindow):  
    def __init__(self):  
        super(MainWindow, self).__init__()  
  
        self.setWindowTitle("My App")  
  
app = QApplication(sys.argv)  
  
window = MainWindow()  
window.show()  
  
app.exec()
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

QT

Contexte

Exemples

Conclusion

Code natif

Python scientifique

# Exemples de code

Matthieu Falce

```
# source: https://www.pythonguis.com/tutorials/pyqt-signals-slots-events/
import sys
from PyQt5.QtWidgets import QApplication, QMainWindow, QPushButton

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()

        self.button_is_checked = True

        self.setWindowTitle("My App")

        button = QPushButton("Press Me!")
        button.setCheckable(True)
        button.clicked.connect(self.the_button_was_toggled)
        button.setChecked(self.button_is_checked)

        self.setCentralWidget(button)

    def the_button_was_toggled(self, checked):
        self.button_is_checked = checked

        print(self.button_is_checked)

app = QApplication(sys.argv)

window = MainWindow()
window.show()

app.exec()
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

QT

Contexte

Exemples

Conclusion

Code natif

Python scientifique

## 7- Interface graphiques

### 7.2. Conclusion

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

QT

**Conclusion**

Choix du framework

Autres bibliothèques

Code natif

Python scientifique

# Elements à considérer

Matthieu Falce

- ▶ il n'y a pas de framework qui soit systématiquement à privilégier
- ▶ cela dépend des conditions d'utilisation / complexité de l'application
- ▶ est-il pertinent de réaliser une application
  - ▶ lourde (accessible depuis une application) / web (accessible depuis un navigateur)
  - ▶ native (spécifique à un OS) ou multi-plateforme (généraliste mais peut être moins adapté)

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

QT

Conclusion

Choix du framework

Autres bibliothèques

Code natif

Python scientifique

# Comparaison

Matthieu Falce

Avantages

Qt

- \* Multi-plateforme / widgets spécifiques
- \* Flexible / permet d'organiser le code
- \* Qt creator (création d'interfaces en glissé déposé)
- \* Fourni un écosystème d'outils (connexion aux bases de données, threads, fichiers...)
- \* Nombreux widgets
- \* Beaucoup de ressources en ligne

Tkinter

- \* Disponible de base en python sans rien installer
- \* Facile à prendre en main

Inconvénients

- \* Complexé (POO, il faut chercher la documentation pour le C++)
- \* Mécanisme de licence compliqué quand on ne fait pas de l'open source
- \* Doit être installé

- \* Pas de widgets avancés (un tableau par exemple)
- \* Intégration au style de l'OS compliquée
- \* Gestion de la complexité compliquée

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

QT

Conclusion

Choix du framework

Autres bibliothèques

Code natif

Python scientifique

Avantage / inconvénients des solutions (source :  
<https://dev.to/amigosmaker/python-gui-pyqt-vs-tkinter-5hdd>)

# Listing

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

QT

Conclusion

Choix du framework

Autres bibliothèques

Code natif

Python scientifique

Il existe d'autre framework d'interfaces graphiques

- ▶ GTK
- ▶ wxPython
- ▶ Kivy

Il existe également des bibliothèques permettant d'abstraire le choix du framework qui peuvent être intéressantes :

<https://pysimplegui.readthedocs.io/en/latest/> (tk, qt, wxpython et web)

# Code natif

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

## Code natif

Ctypes

Cython

Embarquer du code Python  
dans du C

Bibliographie

Python scientifique

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

**Ctypes**

Cython

Embarquer du code Python  
dans du C

Bibliographie

Python scientifique

## 8- Code natif

### 8.1. Ctypes

# Ctypes

Matthieu Falce

Permet de manipuler des DLL / so et d'appeler leurs fonctions

## test1.c

```
#include <stdio.h>
#include <stdlib.h>

void format_hello(char* res, char* name, uint size){
    snprintf(res, size-1, "Hello %s !\n", name);
}
```

---

## test2.c

```
long factorielle(int n){
    long res = 1;
    while(n > 0){
        res *= n;
        n--;
    }
    return res;
}
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Ctypes

Cython

Embarquer du code Python  
dans du C

Bibliographie

Python scientifique

# Ctypes

Matthieu Falce

```
test1.so: test1.c
    gcc -shared -o libtest1.so -fPIC -Wall test1.c

main1: main1.c test1.so
    gcc main1.c -Wall -ldl -o main

main1_2: main1_2.c test1.so
    gcc main1_2.c -Wall -ltest1 -L. -o main1_2

test2.so: test2.c
    gcc -shared -o libtest2.so -fPIC -Wall test2.c

main2: main2.c test2.so
    gcc main2.c -Wall -ltest2 -L. -o main2
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Ctypes

Cython

Embarquer du code Python  
dans du C

Bibliographie

Python scientifique

# Ctypes

Matthieu Falce

## main1.c

```
#include <stdio.h>
#include <stdlib.h>
#include <dlfcn.h>

#define STR_LEN 40

int main(){
    void* test1_lib;
    void (*format_hello)(char*, char*, uint);

    test1_lib = dlopen("./libtest1.so", RTLD_LAZY);
    if ( test1_lib == NULL )
        fprintf(stderr, "Error opening the library\n");

    *(void **)(&format_hello) = dlsym(test1_lib, "format_hello");

    char res[STR_LEN];
    format_hello(res, "Matthieu", STR_LEN-1);
    printf("%s\n", res);

    return EXIT_SUCCESS;
}
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Ctypes

Cython

Embarquer du code Python  
dans du C

Bibliographie

Python scientifique

# Ctypes

Matthieu Falce

main2.c

```
#include <stdio.h>
#include <stdlib.h>
#include <dlfcn.h>

#define STR_LEN 40

void format_hello(char*, char*, uint);

int main(){
    char res[STR_LEN];
    format_hello(res, "Matthieu", STR_LEN-1);
    printf("%s\n", res);

    return EXIT_SUCCESS;
}
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Ctypes

Cython

Embarquer du code Python  
dans du C

Bibliographie

Python scientifique

# Ctypes

Matthieu Falce

## main.py

```
from ctypes import (
    CDLL, c_char_p, create_string_buffer, c_int
)

def main_factorielle():
    lib_factorielle = CDLL('./libtest2.so')
    factorielle = lib_factorielle.factorielle

    for i in range(10):
        print("factorielle {} : {}".format(
            i, factorielle(i)))
    )

def main_hello():
    lib_hello = CDLL('./libtest1.so')

    res = create_string_buffer(40)

    format_hello = lib_hello.format_hello
    format_hello.argtypes = [c_char_p, c_char_p, c_int]

    name = "Matthieu" * 202
    format_hello(res, name.encode(), 40 - 1)

    print(res.value)
    print(res.raw)

if __name__ == '__main__':
    main_hello()
    main_factorielle()
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Ctypes

Cython

Embarquer du code Python  
dans du C

Bibliographie

Python scientifique

# Ctypes

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Ctypes

Cython

Embarquer du code Python  
dans du C

Bibliographie

Python scientifique

Pratique pour intégrer rapidement du code depuis une bibliothèque native.

Assez compliqué à maintenir.

Pas de construction graduelle vers le C.

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Ctypes

Cython

Embarquer du code Python  
dans du C

Bibliographie

Python scientifique

## 8- Code natif

### 8.2. Cython

Cython est un compilateur statique / langage permettant :

- ▶ de compiler du code python vers du C / une DLL
- ▶ de faire de l'optimisation / typage progressif
- ▶ manipuler et échanger des données entre python et C
- ▶ ...

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Ctypes

Cython

Embarquer du code Python  
dans du C

Bibliographie

Python scientifique

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Ctypes

Cython

Embarquer du code Python  
dans du C

Bibliographie

Python scientifique

Cython est un compilateur statique / langage permettant :

- ▶ de compiler du code python vers du C / une DLL
- ▶ de faire de l'optimisation / typage progressif
- ▶ manipuler et échanger des données entre python et C
- ▶ ...

Cython permet l'amélioration progressive du code. Essayez  
`cython -a mon_fichier.pyx`

# Cython

Matthieu Falce

tools.c :

```
#include "stdio.h"
#include "stdlib.h"
#include <math.h>
#include <stdint.h>
#include <string.h>

#define STRING_SIZE 50

void format_hello(char* res, char* name){
    strcat(res, name);
    strcat(res, " ! \n");
}

double somme_elements(double *A, int m, int n)
{
    double somme = 0;
    for (int i = 0; i < m; i++)
        for (int j = 0; j < n; j++)
            somme += A[i*m + j];
    return somme;
}

int main(void){
    char hello[40] = "Hello ";
    format_hello(hello, "Matthieu");
    printf("%s", hello);
    return 0;
}
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Ctypes

Cython

Embarquer du code Python  
dans du C

Bibliographie

Python scientifique

# Cython

Matthieu Falce

## wrapper.pyx :

```
from libc.stdlib cimport malloc, free
from libc.stdlib cimport rand, RAND_MAX
cimport numpy as np

cdef extern from "tools.c":
    void format_hello(char* res, char* name)
    double somme_elements(double *A, int m, int n)

cpdef str hello(str name):
    """
    http://docs.cython.org/en/latest/src/tutorial/strings.html
    """
    cdef char res[40]
    res[:6] = "Hello "

    # protection stack overflow
    if len(name) > 40 - 6 - 1:
        raise MemoryError

    byte_name = name.encode()
    cdef char* c_name = byte_name
    format_hello(res, c_name)
    cdef bytes py_string = res
    return py_string.decode().strip()

cpdef double sum_np_array(np.ndarray[double, ndim=2, mode="c"] np_array):
    cdef int m, n
    m, n = np_array.shape[0], np_array.shape[1]
    return somme_elements(<double*> np_array.data, m, n)
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Ctypes

Cython

Embarquer du code Python  
dans du C

Bibliographie

Python scientifique

setup.py (python setup.py build\_ext -inplace) :

```
from distutils.core import setup
from distutils.extension import Extension
from Cython.Distutils import build_ext

ext_modules = [
    Extension(
        "tools_wrapper",
        [
            "tools_wrapper.pyx"
        ],
        libraries=["m"],
        extra_compile_args=["-ffast-math", "-fopenmp", "-O3"],
        extra_link_args=["-fopenmp"],
    )
]

setup(
    name="tools_wrapper",
    cmdclass={"build_ext": build_ext},
    ext_modules=ext_modules
)
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Ctypes

Cython

Embarquer du code Python  
dans du C

Bibliographie

Python scientifique

# Cython

Matthieu Falce

main.py :

```
import numpy as np

from tools_wrapper import hello, sum_np_array

a = np.arange(100).reshape((10, 10))
a = a / sum(a) # on veut que la somme fasse 1
print(sum_np_array(a))

name = "Matthieu -- from C with love"
print(hello(name))
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Ctypes

Cython

Embarquer du code Python  
dans du C

Bibliographie

Python scientifique

## Résultat du cython -a tools\_wrapper.pyx :

Generated by Cython 0.27.3

Yellow lines hint at Python interaction.  
Click on a line that starts with a "+" to see the C code that Cython generated for it.

Raw output: [tools wrapper.c](#)

```
+01: from libc.stdlib cimport malloc, free
02: from libc.stdlib cimport rand, RAND_MAX
03:
04: cdef extern from "tools.c":
05:     void format_hello(char* res, char* name)
06:     double somme_elements(double* A, int m, int n)
07:     cimport numpy as np
08:
09:
+10: cpdef str hello(str name):
11:     """
12:     http://docs.cython.org/en/latest/src/tutorial/strings.html
13:     """
14:     cdef char res[40]
15:     res[:6] = "Hello "
16:
17:     # protection stack overflow
+18:     if len(name) > 40 - 6 - 1:
+19:         raise MemoryError
20:
21:     byte_name = name.encode()
+22:     cdef char* c_name = byte_name
+23:     format_hello(res, c_name)
+24:     cdef bytes py_string = res
+25:     return py_string.decode().strip()
26:
+27: cpdef double sum_np_array(np.ndarray[double, ndim=2, mode="c"] np_array):
28:     cdef int m, n
+29:     m, n = np_array.shape[0], np_array.shape[1]
+30:     return somme_elements(
31:         <double*> np_array.data,
32:         m, n
33:     )
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Types

Cython

Embarquer du code Python  
dans du C

Bibliographie

Python scientifique

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Ctypes

Cython

**Embarquer du code Python  
dans du C**

Bibliographie

Python scientifique

## 8- Code natif

### 8.3. Embarquer du code Python dans du C

# Explications

Matthieu Falce

Il y a deux façons de faire cohabiter Python et C

- ▶ augmenter Python avec des routines C (ce que l'on a vu)
- ▶ embarquer l'interpréteur Python dans le C (ce que l'on va voir)

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Ctypes

Cython

Embarquer du code Python  
dans du C

Bibliographie

Python scientifique

# Explications

Matthieu Falce

Il y a deux façons de faire cohabiter Python et C

- ▶ augmenter Python avec des routines C (ce que l'on a vu)
- ▶ embarquer l'interpréteur Python dans le C (ce que l'on va voir)



Nécessite de connaître suffisamment le C pour comprendre  
l'API C de Python

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Ctypes

Cython

Embarquer du code Python  
dans du C

Bibliographie

Python scientifique

# Embarquer du code

Matthieu Falce

Il existe 3 niveaux d'embarquement :

- ▶ vu que l'on initialise un interpréteur, on peut appeler des chaînes de code directement
- ▶ on peut appeler des fonctions python et récupérer leur valeurs (échange des paramètres et des valeurs retournées)
- ▶ on peut mettre à disposition des variables C dans un module que l'on importe dans le code interprété

Toutes les infos sont ici : <https://docs.python.org/3/extending/embedding.html>

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Ctypes

Cython

Embarquer du code Python  
dans du C

Bibliographie

Python scientifique

# Exemple d'embarquement de code

Matthieu Falce

```
#define PY_SSIZE_T_CLEAN
#include <Python.h>

int
main(int argc, char *argv[])
{
    wchar_t *program = Py_DecodeLocale(argv[0], NULL);
    if (program == NULL) {
        fprintf(stderr, "Fatal error: cannot decode argv[0]\n");
        exit(1);
    }
    Py_SetProgramName(program); /* optional but recommended */
    Py_Initialize();
    PyRun_SimpleString("from time import time,ctime\n"
                       "print('Today is', ctime(time()))\n");
    if (Py_FinalizeEx() < 0) {
        exit(120);
    }
    PyMem_RawFree(program);
    return 0;
}
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Ctypes

Cython

Embarquer du code Python  
dans du C

Bibliographie

Python scientifique

## Embarquer une chaîne de Python

# Exemple d'embarquement de code

Matthieu Falce

```
...
Py_Initialize();
pName = PyUnicode_DecodeFSDefault(argv[1]);
pModule = PyImport_Import(pName);
Py_DECREF(pName);

if (pModule != NULL) {
    pFunc = PyObject_GetAttrString(pModule, argv[2]);
    /* pFunc is a new reference */

    if (pFunc && PyCallable_Check(pFunc)) {
        pArgs = PyTuple_New(argc - 3);
        for (i = 0; i < argc - 3; ++i) {
            pValue = PyLong_FromLong(atoi(argv[i + 3]));
            // ... removed check if not pValue
            PyTuple_SetItem(pArgs, i, pValue);
        }
        pValue = PyObject_CallObject(pFunc, pArgs);
        Py_DECREF(pArgs);
        if (pValue != NULL) {
            printf("Result of call: %ld\n", PyLong_AsLong(pValue));
            ...
    }
}
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Ctypes

Cython

Embarquer du code Python  
dans du C

Bibliographie

Python scientifique

Appeler un module Python (attention au PYTHONPATH)

# Gotchas



Matthieu Falce

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Ctypes

Cython

Embarquer du code Python  
dans du C

Bibliographie

Python scientifique

- ▶ cette partie fonctionne sous Linux (Ubuntu au moins), pour Windows je n'ai pas testé
- ▶ il faut déterminer les paramètres de compilation pour sa machine :
  - ▶ `CFLAGS` : lancer `python-config --cflags`
  - ▶ `LDLFLAGS` : lancer `python-config --ldflags`
- ▶ l'interpréteur embarqué ne semble pas régler `PYTHONPATH` avec le dossier courant, il faut le faire à la main, sinon  
`ImportError ( PyRun_SimpleString("import sys, os;  
sys.path.append(os.getcwd())"));`
- ▶ l'intégration de code Python et C est vue comme de la *magie noire*. Ce n'est pas vrai, c'est faisable, cependant, cela nécessite de bonnes connaissances dans les deux langages.

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Ctypes

Cython

Embarquer du code Python  
dans du C

**Bibliographie**

Python scientifique

## 8- Code natif

### 8.4. Bibliographie

# Bibliographie I

Matthieu Falce

- ▶ étendre python avec du C
  - ▶ <https://wiki.python.org/moin/IntegratingPythonWithOtherLanguages>
  - ▶ <https://docs.scipy.org/doc/numpy/user/c-info.python-as-glue.html>
  - ▶ <https://stackoverflow.com/questions/145270/calling-c-c-from-python>
  - ▶ SIP (binding Qt et GTK)
  - ▶ [www.swig.org](http://www.swig.org)
  - ▶ <http://sametmax.com/appeler-du-code-c-depuis-python-avec-ctypes/>
  - ▶ [http://www.boost.org/doc/libs/1\\_49\\_0/libs/python/doc/](http://www.boost.org/doc/libs/1_49_0/libs/python/doc/)
  - ▶ <https://github.com/pybind/pybind11>
  - ▶ <https://cffi.readthedocs.io/en/latest/overview.html#simple-example-abi-level-in-line>
  - ▶ <http://sametmax.com/introduction-aux-extentions-python-avec-cffi>

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Ctypes

Cython

Embarquer du code Python  
dans du C

Bibliographie

Python scientifique

# Bibliographie II

Matthieu Falce

- ▶ sur Windows : <https://docs.python.org/3/extending/windows.html>
  - ▶ <https://docs.python.org/3/extending/building.html>
  - ▶ embarquer python dans du C
    - ▶ <https://docs.python.org/3/c-api/>
    - ▶ <https://docs.python.org/3/extending/embedding.html>
- Vue d'ensemble
- Langage Python
- Programmation Orientée objet (POO)
- Bonnes pratiques
- Bibliothèque standard
- Création de modules
- Interface graphiques
- Code natif
- Ctypes
- Cython
- Embarquer du code Python dans du C
- Bibliographie
- Python scientifique

# Python scientifique

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

## Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

Performances

Dask

Bibliographie

# 9- Python scientifique

## 9.1. Écosystème scientifique

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

Performances

Dask

Bibliographie

# Écosystème

Matthieu Falce

- ▶ écosystème très riche
- ▶ utilisé aussi bien par les scientifiques que les entreprises
- ▶ origine : développeurs et scientifiques
- ▶ sponsorisé par de grandes entreprises (google, facebook,  
**Enthought, Anaconda Inc**)
- ▶ tout ce que vous cherchez existe probablement déjà

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

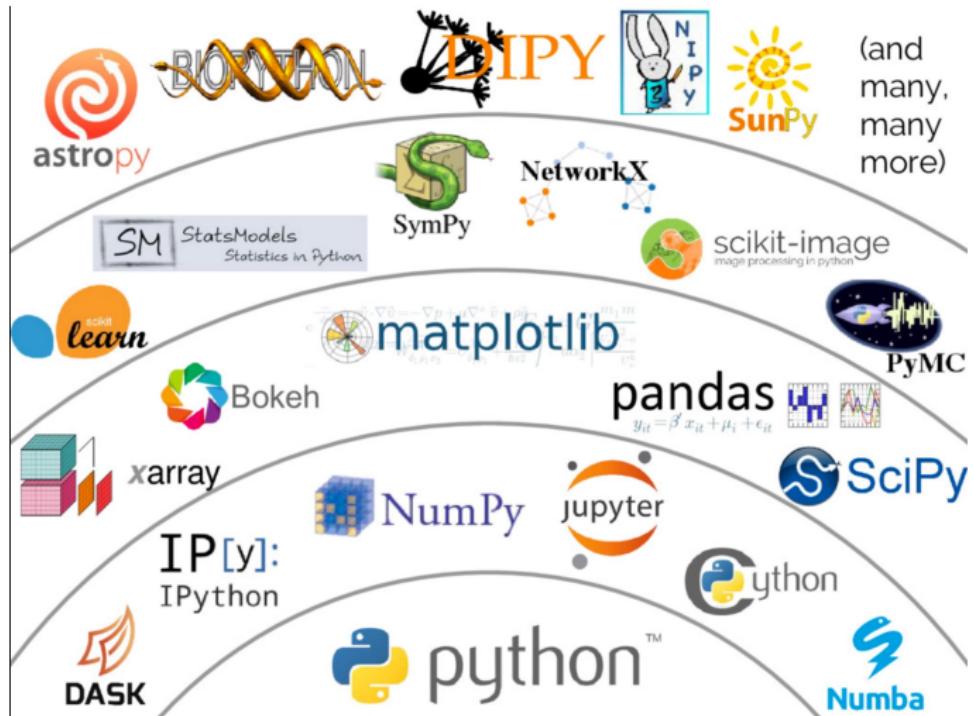
Performances

Dask

Bibliographie

# Écosystème

Matthieu Falce



Source : <https://www.datacamp.com/community/blog/python-scientific-computing-case>

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

Performances

Dask

Bibliographie

# Écosystème

Matthieu Falce

## Calculs :

- ▶ **numpy** <sup>42</sup>
- ▶ **scipy** <sup>43</sup>
- ▶ **pandas** <sup>44</sup>
- ▶ ...

## Plotting :

- ▶ **matplotlib** <sup>45</sup>
- ▶ **seaborn** <sup>46</sup>
- ▶ **bokeh** <sup>47</sup>
- ▶ ...

---

42.<http://www.numpy.org/>

43.<https://www.scipy.org/>

44.<https://pandas.pydata.org/>

45.<https://matplotlib.org/>

46.<https://seaborn.pydata.org/>

47.<https://bokeh.pydata.org/en/latest/>

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique  
Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

Performances

Dask

Bibliographie

# Python scientifique

Matthieu Falce

```
import numpy as np
```

```
xs = np.arange(-2*np.pi, 2*np.pi, 100)
ys = np.sin(xs) - 3*xs + 2
```

```
#####
```

```
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])
print(A.dot(B))
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

Performances

Dask

Bibliographie

# 9- Python scientifique

## 9.2. Jupyter

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

Performances

Dask

Bibliographie

# IPython – Jupyter

Matthieu Falce

- ▶ shell interactif avec auto-complétion
- ▶ fonctions *magique* (mesure du temps, infos shell...)
- ▶ notebook (et maintenant lab) → programmation littérale, IDE en ligne
- ▶ utilisation d'autres "noyaux" (R, Julia, C, Haskell...)
- ▶ calcul parallèle
- ▶ présentation des résultats
- ▶ ...

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

Performances

Dask

Bibliographie

# IPython – Jupyter

Matthieu Falce

- ▶ shell interactif avec auto-complétion
- ▶ fonctions *magique* (mesure du temps, infos shell...)
- ▶ notebook (et maintenant lab) → programmation littérale, IDE en ligne
- ▶ utilisation d'autres "noyaux" (R, Julia, C, Haskell...)
- ▶ calcul parallèle
- ▶ présentation des résultats
- ▶ ...



Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

Performances

Dask

Bibliographie

# 9- Python scientifique

## 9.3. Numpy

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

**Numpy**

Présentation

Tableaux

Structure des tableaux

Création des tableaux

Manipulations usuelles

Broadcasting

Fonctions universelles

Scipy

Pandas

Sympy

# Présentation

Matthieu Falce

## Paquet fondateur de la stack scientifique Python.

- ▶ propose un type de tableaux multidimensionnels
- ▶ met les performances au premier plan
- ▶ manipulation vectorielles / matricielles faciles
- ▶ outils pour manipuler ces tableaux (algèbre linéaire, traitement du signal, ...)

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Présentation

Tableaux

Structure des tableaux

Création des tableaux

Manipulations usuelles

Broadcasting

Fonctions universelles

Scipy

Pandas

Sympy

## Paquet fondateur de la stack scientifique Python.

- ▶ propose un type de tableaux multidimensionnels
- ▶ met les performances au premier plan
- ▶ manipulation vectorielles / matricielles faciles
- ▶ outils pour manipuler ces tableaux (algèbre linéaire, traitement du signal, ...)

Utilise des bibliothèques Fortran ou C/C++ → bonnes performances

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Présentation

Tableaux

Structure des tableaux

Création des tableaux

Manipulations usuelles

Broadcasting

Fonctions universelles

Scipy

Pandas

Sympy

# Tableaux numpy

Matthieu Falce

- ▶ tableaux multidimensionnels (`NDarray`)
- ▶ facilement indexable
- ▶ vectorisation du code (`UFunc` et `broadcasting`)

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Présentation

**Tableaux**

Structure des tableaux

Création des tableaux

Manipulations usuelles

Broadcasting

Fonctions universelles

Scipy

Pandas

Sympy

# Tableaux numpy

Matthieu Falce

- ▶ tableaux multidimensionnels (`NDarray`)
- ▶ facilement indexable
- ▶ vectorisation du code (`UFunc` et `broadcasting`)

Même manipulation que Matlab

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Présentation

**Tableaux**

Structure des tableaux

Création des tableaux

Manipulations usuelles

Broadcasting

Fonctions universelles

Scipy

Pandas

Sympy

# Tableaux numpy

Matthieu Falce

- ▶ tableaux multidimensionnels (NDarray)
- ▶ facilement indexable
- ▶ vectorisation du code (UFunc et broadcasting)

```
import numpy as np

xs = np.arange(-2*np.pi, 2*np.pi, 100)
ys = np.sin(xs) - 3*xs + 2

#####
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])
print(A.dot(B))
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Présentation

Tableaux

Structure des tableaux

Création des tableaux

Manipulations usuelles

Broadcasting

Fonctions universelles

Scipy

Pandas

Sympy

# Structure

Matthieu Falce

- ▶ un seul type de données par tableau (ou objet python)
- ▶ métadonnées + données linéaires
- ▶ orientation (C ou Fortran) → important pour les performances

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Présentation

Tableaux

**Structure des tableaux**

Création des tableaux

Manipulations usuelles

Broadcasting

Fonctions universelles

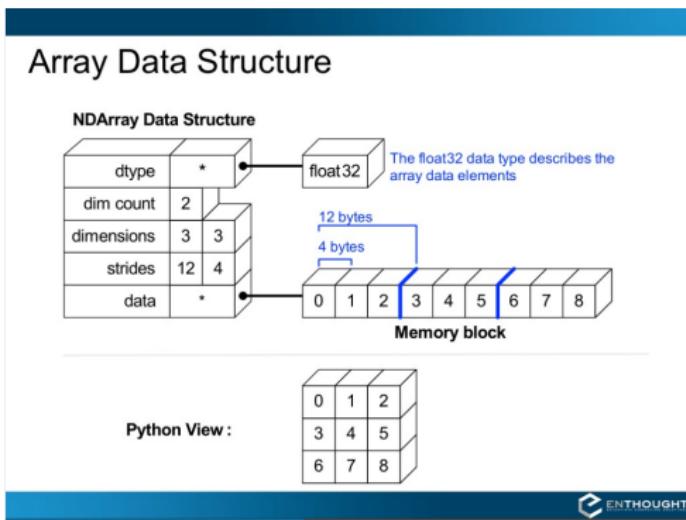
Scipy

Pandas

Sympy

# Structure

- ▶ un seul type de données par tableau (ou objet python)
- ▶ métadonnées + données linéaires
- ▶ orientation (C ou Fortran) → important pour les performances



<https://www.slideshare.net/enthought/numpy-talk-at-siam>

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Présentation

Tableaux

Structure des tableaux

Création des tableaux

Manipulations usuelles

Broadcasting

Fonctions universelles

Scipy

Pandas

Sympy

# Structure

Matthieu Falce

- ▶ un seul type de données par tableau (ou objet python)
- ▶ métadonnées + données linéaires
- ▶ orientation (C ou Fortran) → important pour les performances

Row-major order

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

Column-major order

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

By Cmglee - Own work, CC BY-SA 4.0, <https://commons.wikimedia.org/w/index.php?curid=65107030>

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Présentation

Tableaux

Structure des tableaux

Création des tableaux

Manipulations usuelles

Broadcasting

Fonctions universelles

Scipy

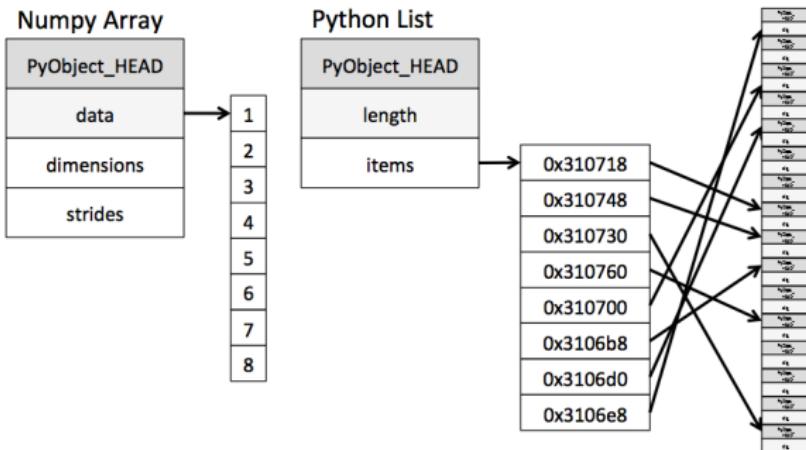
Pandas

Sympy

# Structure

Matthieu Falce

- ▶ un seul type de données par tableau (ou objet python)
- ▶ métadonnées + données linéaires
- ▶ orientation (C ou Fortran) → important pour les performances



<https://jakevdp.github.io/PythonDataScienceHandbook/02.01-understanding-data-types.html>

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Présentation

Tableaux

Structure des tableaux

Création des tableaux

Manipulations usuelles

Broadcasting

Fonctions universelles

Scipy

Pandas

Sympy

# Création des tableaux

Matthieu Falce

```
import numpy as np

# à partir d'une liste / liste de liste...
xs = np.array([i for i in range(10)])
A = np.array([[1, 2], [3, 4]])

# équivalent de range
rs = np.arange(10, 50, 2)

# valeurs régulièrement espacées
es = np.linspace(-3.65, np.pi, 100)

# forcer les types
ts = np.linspace(-3.65, np.pi, 100, dtype=np.int)
ts2 = np.linspace(-3.65, np.pi, 100, dtype=np.complex) + 2j

# tous les utilitaires classiques
ones = np.ones((3, 1))
diag = np.eye(3)
full = np.full((3, 2), np.pi)
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Présentation

Tableaux

Structure des tableaux

Création des tableaux

Manipulations usuelles

Broadcasting

Fonctions universelles

Scipy

Pandas

Sympy

# Création des tableaux

Matthieu Falce

## Creation

Code	Result	Code	Result
<code>Z = zeros(9)</code>		<code>Z = zeros((5,9))</code>	
<code>Z = ones(9)</code>		<code>Z = ones((5,9))</code>	
<code>Z = array([0,0,0,0,0,0,0,0])</code>		<code>Z = array([[0,0,0,0,0,0,0,0,0], [0,0,0,0,0,0,0,0,0], [0,0,0,0,0,0,0,0,0], [0,0,0,0,0,0,0,0,0], [0,0,0,0,0,0,0,0,0]])</code>	
<code>Z = arange(9)</code>		<code>Z = arange(5*9).reshape(5,9)</code>	
<code>Z = random.uniform(0,1,9)</code>		<code>Z = random.uniform(0,1,(5,9))</code>	

<http://www.labri.fr/perso/nrougier/teaching/numpy/numpy.html>

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Présentation

Tableaux

Structure des tableaux

Création des tableaux

Manipulations usuelles

Broadcasting

Fonctions universelles

Scipy

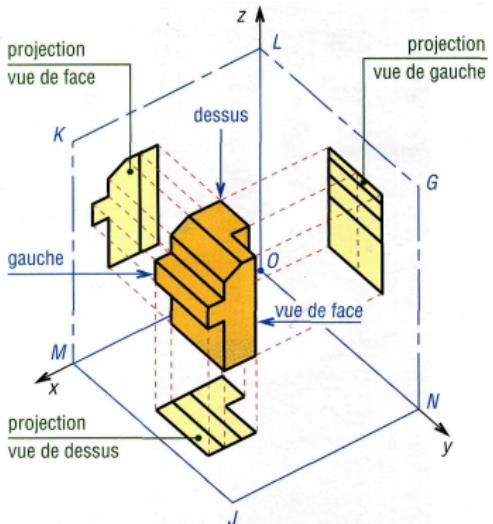
Pandas

Sympy

# Changements de forme / dimensions

Matthieu Falce

Certaines opérations vont s'opérer sur certaines dimensions.  
On peut les visualiser comme étant des projections.



[http://www.zpag.net/Tecnologies\\_Industrielles/projections\\_orthogonales\\_normalis.htm](http://www.zpag.net/Tecnologies_Industrielles/projections_orthogonales_normalis.htm)

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Présentation

Tableaux

Structure des tableaux

Création des tableaux

Manipulations usuelles

Broadcasting

Fonctions universelles

Scipy

Pandas

Sympy

# Changements de forme / dimensions

Matthieu Falce

```
import numpy as np

# changement du nombre de dimensions
A = np.arange(1, 3 * 4 + 1).reshape((3, 4))
print(A)
# [[ 1  2  3  4]
# [ 5  6  7  8]
# [ 9 10 11 12]]

# on peut effectuer des actions selon
# certaines dimensions
B = np.arange(1, (3 * 4 * 5) + 1).reshape((3, 4, 5))
B.mean(axis=0)
B.mean(axis=1)
B.mean(axis=2)

B.sum(axis=1)
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Présentation

Tableaux

Structure des tableaux

Création des tableaux

**Manipulations usuelles**

Broadcasting

Fonctions universelles

Scipy

Pandas

Sympy

# Changements de forme / dimensions

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Présentation

Tableaux

Structure des tableaux

Création des tableaux

Manipulations usuelles

Broadcasting

Fonctions universelles

Scipy

Pandas

Sympy

## Reshaping

Code	Result	Code	Result
<code>Z[2,2] = 1</code>		<code>Z = Z.reshape(1,12)</code>	
<code>Z = Z.reshape(4,3)</code>			
<code>Z = Z.reshape(6,2)</code>		<code>Z = Z.reshape(12,1)</code>	
<code>Z = Z.reshape(2,6)</code>			

<http://www.labri.fr/perso/nrougier/teaching/numpy/numpy.html>

# Indexing

Matthieu Falce

```
import numpy as np

# découpage
D = np.arange((100 * 5)).reshape((100, 5))
split = 30
test, train = D[:split, :], D[split:, :]

# indexation booléen
# on met toutes les valeurs paires à 0
A = np.arange(1, 3 * 4 + 1).reshape((3, 4))
pairs = (A % 2 == 0)
pairs = pairs.astype(bool)
A[pairs] = 0

# slicing et réassiguation partielle
B = np.arange(100, dtype=np.uint8).reshape((10, 10))
B[:, ::2, ::2] = B[1::2, 1::2] / 2
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Présentation

Tableaux

Structure des tableaux

Création des tableaux

Manipulations usuelles

Broadcasting

Fonctions universelles

Scipy

Pandas

Sympy

# Indexing

Matthieu Falce

## Slicing

Code	Result	Code	Result
<code>  z</code>		<code>  z[...] = 1</code>	
<code>  z[1,1] = 1</code>		<code>  z[:,0] = 1</code>	
<code>  z[0,:] = 1</code>		<code>  z[2:,2:] = 1</code>	
<code>  z[:,::2] = 1</code>		<code>  z[::2,:,:] = 1</code>	
<code>  z[:-2,:-2] = 1</code>		<code>  z[2:4,2:4] = 1</code>	
<code>  z[::2,:,:2] = 1</code>		<code>  z[3::2,3::2] = 1</code>	

<http://www.labri.fr/perso/nrougier/teaching/numpy/numpy.html>

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Présentation

Tableaux

Structure des tableaux

Création des tableaux

Manipulations usuelles

Broadcasting

Fonctions universelles

Scipy

Pandas

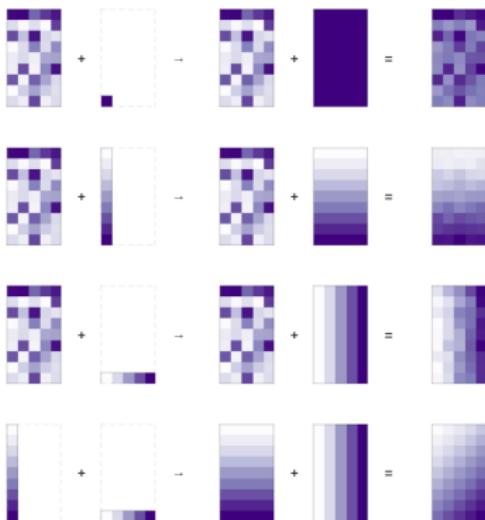
Sympy

# Broadcasting

Matthieu Falce

Le broadcasting permet de manipuler entre eux des tableaux de tailles différentes.

Broadcasting



[http://www.labri.fr/perso/nrougier/teaching\(numpy/numpy.html](http://www.labri.fr/perso/nrougier/teaching(numpy/numpy.html)

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Présentation

Tableaux

Structure des tableaux

Création des tableaux

Manipulations usuelles

**Broadcasting**

Fonctions universelles

Scipy

Pandas

Sympy

# Broadcasting

Le broadcasting permet de manipuler entre eux des tableaux de tailles différentes.



Ce n'est pas magique

```
In [10]: A = np.arange(9).reshape((3, 3))
In [11]: B = np.arange(4)
In [12]: A + B
```

---

```
ValueError      Traceback (most recent call last)
<ipython-input-12-151064de832d> in <module>()
----> 1 A + B
ValueError: operands could not be broadcast
together with shapes (3,3) (4,)
```

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Présentation

Tableaux

Structure des tableaux

Création des tableaux

Manipulations usuelles

**Broadcasting**

Fonctions universelles

Scipy

Pandas

Sympy

# Fonctions universelles

Matthieu Falce

Fonctions universelles s'appliquent sur les éléments d'un tableau de façon vectorisée (sans boucles apparentes).

Exemple : `y = np.sin(x)`

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Présentation

Tableaux

Structure des tableaux

Création des tableaux

Manipulations usuelles

Broadcasting

**Fonctions universelles**

Scipy

Pandas

Sympy

# Fonctions universelles

Matthieu Falce

Fonctions universelles s'appliquent sur les éléments d'un tableau de façon vectorisée (sans boucles apparentes).

Exemple : `y = np.sin(x)`

On peut créer ses propres fonctions vectorisées avec `np.frompyfunc` ou `np.vectorize`.



Ce n'est pas pour ça qu'elles seront plus rapides.

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Présentation

Tableaux

Structure des tableaux

Création des tableaux

Manipulations usuelles

Broadcasting

Fonctions universelles

Scipy

Pandas

Sympy

# 9- Python scientifique

## 9.4. Scipy

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Présentation

Scikits

Fonctionnalités

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

## Méthodes additionnelles à Numpy pour le calcul scientifique.

- ▶ fonctions spéciales
- ▶ intégration
- ▶ optimisation
- ▶ équations différentielles
- ▶ traitement du signal
- ▶ algèbre linéaire
- ▶ ...

Si des routines sont partagées avec Numpy → plus complètes  
dans Scipy<sup>48</sup>

Scipy utilise LAPACK, Numpy pas forcément<sup>49</sup>

48.<https://docs.scipy.org/doc/numpy/reference/routines.dual.html>

49.<https://www.scipy.org/scipylib/faq.html#what-is-the-difference-between-numpy-and-scipy>

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Présentation

Scikits

Fonctionnalités

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

## *SciPy Toolkits* → extensions pour Scipy indépendantes

Liste non exhaustive<sup>50</sup>

- ▶ scikit-learn
- ▶ scikit-image
- ▶ scikit-bio
- ▶ ...

---

50.liste complète : <https://scikits.appspot.com/scikits>

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Présentation

Scikits

Fonctionnalités

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

```
import numpy as np
import matplotlib.pyplot as plt

t = np.arange(256)
sig = np.sin(t)

# sp est un tableau de complexes
spl = np.fft.fft(sig)
freq1 = np.fft.fftfreq(t.shape[-1])
plt.plot(freq1, spl.real, freq1, spl.imag)
plt.show()

# comparaisons fréquences
sig2 = np.sin(2 * t)
sp2 = np.fft.fft(sig2)
freq2 = np.fft.fftfreq(t.shape[-1])

plt.plot(freq2, sp2.real, label="F2")
plt.plot(freq1, spl.real, label="F1")
plt.legend()
plt.show()
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standardCréation de  
modulesInterface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Présentation

Scikits

Fonctionnalités

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

```
from scipy.optimize import minimize

minimize(lambda x: x**2, x0=1000)

#      fun: 5.713415792109052e-17
# hess_inv: array([[0.50000012]])
#      jac: array([-2.16266884e-10])
# message: 'Optimization terminated successfully.'
#      nfev: 24
#      nit: 3
#      njev: 8
#      status: 0
# success: True
#      x: array([-7.55871404e-09])
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Présentation

Scikits

Fonctionnalités

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

```
from scipy.optimize import minimize

minimize(lambda x: x[0]**2 + x[1] ** 2, x0=(1000, 33))

#      fun: 1.3011523813214424e-13
# hess_inv: array([[0.54198343, 0.00254437],
#                  [0.00254437, 0.5001542 ]])
#      jac: array([7.35719908e-07, 4.45876263e-08])
# message: 'Optimization terminated successfully.'
#      nfev: 40
#      nit: 5
#      njev: 10
#      status: 0
# success: True
#      x: array([3.60409374e-07, 1.48432326e-08])
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Présentation

Scikits

Fonctionnalités

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

```
from scipy.optimize import minimize

minimize(lambda x: x[0]**2 + x[1] ** 2, x0=(1000, 33))

#      fun: 1.3011523813214424e-13
# hess_inv: array([[0.54198343, 0.00254437],
#                  [0.00254437, 0.5001542 ]])
#      jac: array([7.35719908e-07, 4.45876263e-08])
# message: 'Optimization terminated successfully.'
#      nfev: 40
#       nit: 5
#      njev: 10
#    status: 0
#   success: True
#      x: array([3.60409374e-07, 1.48432326e-08])
```

On peut mettre des contraintes sur les paramètres et changer de méthode d'optimisation aussi.

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Présentation

Scikits

Fonctionnalités

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

- ▶ en 2D
- ▶ splines et courbes paramétrées
- ▶ ...

```
import numpy as np
from scipy.interpolate import interp1d
import matplotlib.pyplot as plt

x = np.linspace(0, 10, num=11, endpoint=True)
y = np.cos(-x ** 2 / 9.0)
f = interp1d(x, y)
f2 = interp1d(x, y, kind='cubic')

xnew = np.linspace(0, 10, num=41, endpoint=True)
plt.plot(x, y, 'o', xnew, f(xnew), '--', xnew, f2(xnew), '---')
plt.legend(['data', 'linear', 'cubic'], loc='best')
plt.show()
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Présentation

Scikits

Fonctionnalités

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

# Intégration – équations différentielles

54 55

Matthieu Falce

- ▶ en 1D
- ▶ en 2D
- ▶ ... jusqu'à N dimensions
- ▶ paramétrages complexes
- ▶ ODE non stiff et stiff, pas adaptatif...

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Présentation

Scikits

Fonctionnalités

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

---

54.<https://docs.scipy.org/doc/scipy/reference/tutorial/integrate.html>

55.<https://docs.scipy.org/doc/scipy/reference/integrate.html>

# Intégration – équations différentielles

54 55

Matthieu Falce

```
from scipy.integrate import quad

def fonction_a_integrer(x, a, b):
    return a * x ** 3 + b

a = 1
b = 0
surface = quad(
    fonction_a_integrer,
    -10,
    10,
    args=(a, b)
)
print(surface)
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Présentation

Scikits

Fonctionnalités

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

54.<https://docs.scipy.org/doc/scipy/reference/tutorial/integrate.html>

55.<https://docs.scipy.org/doc/scipy/reference/integrate.html>

# Intégration – équations différentielles

54 55

Matthieu Falce

```
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt

def carre_der(t, x):
    return 2*x

ts = np.arange(10)
ys = odeint(carre_der, 2, ts)
plt.plot(ts, ys)
plt.show()
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Présentation

Scikits

Fonctionnalités

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

54.<https://docs.scipy.org/doc/scipy/reference/tutorial/integrate.html>

55.<https://docs.scipy.org/doc/scipy/reference/integrate.html>

“Tout” pour le traitement du signal :<sup>56</sup>

- ▶ convolutions
- ▶ conceptions / analyses de filtres (FIR, FII) / analyse réponse
- ▶ analyses de spectres
- ▶ détections de pics

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Présentation

Scikits

Fonctionnalités

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

56.<https://docs.scipy.org/doc/scipy/reference/signal.html#module-scipy.signal>

57.<https://docs.scipy.org/doc/scipy/reference/tutorial/signal.html>



Fonctions potentiellement légèrement différentes de celles de  
Numpy  
Compilées avec BLAS et LAPACK

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Présentation

Scikits

**Fonctionnalités**

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

Numpy supporte les matrices creuses de différents types :

- ▶ csc\_matrix: Compressed Sparse Column format
- ▶ csr\_matrix: Compressed Sparse Row format
- ▶ bsr\_matrix: Block Sparse Row format
- ▶ lil\_matrix: List of Lists format
- ▶ dok\_matrix: Dictionary of Keys format
- ▶ coo\_matrix: COOrdinate format (aka IJV, triplet format)
- ▶ dia\_matrix: DIAGONAL format

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Présentation

Scikits

Fonctionnalités

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

59.<https://docs.scipy.org/doc/scipy-0.14.0/reference/sparse.html>

60.<https://docs.scipy.org/doc/scipy/reference/tutorial/arpack.html>

Numpy supporte les matrices creuses de différents types :

- ▶ csc\_matrix: Compressed Sparse Column format
- ▶ csr\_matrix: Compressed Sparse Row format
- ▶ bsr\_matrix: Block Sparse Row format
- ▶ lil\_matrix: List of Lists format
- ▶ dok\_matrix: Dictionary of Keys format
- ▶ coo\_matrix: COOrdinate format (aka IJV, triplet format)
- ▶ dia\_matrix: DIAGONAL format

Utiliser les méthodes de `scipy.sparse.linalg` pour faire des opérations et garder des matrices creuses. Selon le type de matrices utilisées ; différentes possibilités (perte du slicing...)

59.<https://docs.scipy.org/doc/scipy-0.14.0/reference/sparse.html>

60.<https://docs.scipy.org/doc/scipy/reference/tutorial/arpack.html>

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Présentation

Scikits

Fonctionnalités

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

- ▶ distributions discrètes et continues
- ▶ analyses de données statistiques (kurtosis, zscores...)
- ▶ tests statistiques
- ▶ ...

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Présentation

Scikits

Fonctionnalités

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

---

61.<https://docs.scipy.org/doc/scipy/reference/tutorial/linalg.html>

62.<https://docs.scipy.org/doc/scipy/reference/stats.html#module-scipy.stats>

# Manipulation d'images

Matthieu Falce

Les images (rasterisées ou pixelisées) sont des tableaux *numpy* classiques.

Voici les principales bibliothèques pour en faire :

- ▶ *Pillow*<sup>63</sup> : plutôt utilisée pour les manipulations classiques (redimensionnement, rotation, changement de format, ...)
- ▶ *numpy / scipy* classique<sup>64</sup> : permet de manipuler les tableaux de données de pixels directement
- ▶ *scikit image*<sup>65 66</sup> : Propose de nombreux algorithmes pour faire du traitement d'images
- ▶ *openCV*<sup>67</sup> : le port de la Bibliothèque d'analyse d'image openCV. Permet de travailler en temps réel (à partir d'une caméra par exemple)

---

63.<https://pillow.readthedocs.io/en/stable/>

64. Plus d'infos ici : [http://scipy-lectures.org/advanced/image\\_processing/](http://scipy-lectures.org/advanced/image_processing/)

65.<https://scikit-image.org/>

66.<https://scipy-lectures.org/packages/scikit-image/index.html>

67.[https://docs.opencv.org/master/d6/d00/tutorial\\_py\\_root.html](https://docs.opencv.org/master/d6/d00/tutorial_py_root.html)

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Présentation

Scikits

Fonctionnalités

Pandas

Sympy

Analyse de réseaux

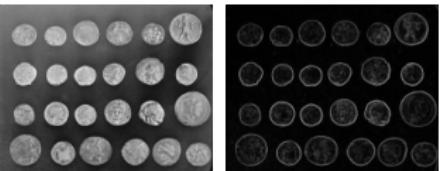
Machine learning /  
statistiques

Graphiques

# Manipulation d'images

Matthieu Falce

```
from skimage import data, io, filters  
  
image = data.coins()  
edges = filters.sobel(image)  
io.imshow(edges)  
io.show()
```



Manipulation d'images avec scikit image.  
Source : <https://scikit-image.org/>

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Présentation

Scikits

Fonctionnalités

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

# Manipulation d'images

Matthieu Falce

```
import cv2 as cv

# Read image from your local file system
original_image = cv.imread("path/to/your-image.jpg")

# Convert color image to grayscale for Viola-Jones
grayscale_image = cv.cvtColor(original_image, cv.COLOR_BGR2GRAY)

# Load the classifier and create a cascade object for face detection
face_cascade = cv.CascadeClassifier("path/to/haarcascade_frontalface_alt.xml")

detected_faces = face_cascade.detectMultiScale(grayscale_image)

for (column, row, width, height) in detected_faces:
    cv.rectangle(
        original_image, (column, row), (column + width, row + height), (0, 255, 0), 2
    )

cv.imshow("Image", original_image)
cv.waitKey(0)
cv.destroyAllWindows()
```

Détection de visages avec openCV.

Source : <https://realpython.com/traditional-face-detection-python/>

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Présentation

Scikits

Fonctionnalités

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

## 9- Python scientifique

### 9.5. Pandas

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

**Pandas**

Présentation

Dataframes

Manipulations de  
dataframes

Sympy

Analyse de réseaux

Machine learning /  
statistiques

# Présentation

Matthieu Falce

Bibliothèque essentielle à l'analyse de données.

Données structurées (lignes / colonnes à la SQL) et séries temporelles.

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

**Présentation**

Dataframes

Manipulations de  
dataframes

Sympy

Analyse de réseaux

Machine learning /  
statistiques

# Dataframes et séries

Matthieu Falce

- ▶ `series` : suite de données à 1 dimension (comme un tableau numpy avec d'autres fonctionnalités)
- ▶ `dataframes` : regroupement de plusieurs séries de même taille (comme un tableau)

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Présentation

**Dataframes**

Manipulations de  
dataframes

Sympy

Analyse de réseaux

Machine learning /  
statistiques

# Manipulations de dataframes

Matthieu Falce

## Création de dataframes et de séries

```
import numpy as np
import pandas as pd

# créer une série
s = pd.Series(np.random.randn(5), index=["a", "b", "c", "d", "e"])

# créer un dataframe
d = {
    "one": pd.Series([1.0, 2.0, 3.0], index=["a", "b", "c"]),
    "two": pd.Series([1.0, 2.0, 3.0, 4.0], index=["a", "b", "c", "d"]),
}
df = pd.DataFrame(d)
df["three"] = s

print("description de df :")
df.describe()
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Présentation

Dataframes

Manipulations de  
dataframes

Sympy

Analyse de réseaux

Machine learning /  
statistiques

# Manipulations de dataframes

Matthieu Falce

## Indexation et accès aux données

```
# https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html

import numpy as np
import pandas as pd

# créer une série
s = pd.Series(np.random.randn(5), index=["a", "b", "c", "d", "e"])
df = pd.DataFrame({"one": s, "two": s[1:], "three": s[2:]})

# accès aux colonnes
one, two = df["one"], df.two
df[["one", "two"]]

# accès aux lignes
df[:1] # slicing
a, bcd, adb = df.loc["a"], df.loc["b"], df.loc[["a", "b", "d"]]
df.iloc[2:]
type(df[1:2]) # pandas.core.frame.DataFrame
type(df.iloc[1]) # pandas.core.series.Series

# accès aux éléments
df.loc["a", "one"] # index ligne, colonne

# échantillonage
seed = 1
df.sample(n=3, replace=True, random_state=seed)
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Présentation

Dataframes

Manipulations de  
dataframes

Sympy

Analyse de réseaux

Machine learning /  
statistiques

# Manipulations de dataframes

Matthieu Falce

## Aparté sur les performances d'indexation

```
## vitesse
# # bad practice
# In [89]: %timeit df["one"]["a"]
# 8.9 µs ± 107 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)

# # high level loc
# In [90]: %timeit df.loc["a", "one"]
# 6.6 µs ± 230 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)

# # low level at
# In [91]: %timeit df.at["a", "one"]
# 3.88 µs ± 33.3 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Présentation

Dataframes

Manipulations de  
dataframes

Sympy

Analyse de réseaux

Machine learning /  
statistiques

# Manipulations de dataframes

Matthieu Falce

## Lecture de CSV et nettoyage de données

```
import pandas as pd
import matplotlib.pyplot as plt

url = ("http://facweb.cs.depaul.edu/mobasher/classes"
       "/csc478/Data/titanic-trimmed.csv")
titanic = pd.read_csv(url)
titanic.head(10)

age_mean = titanic.age.mean()
titanic.age.fillna(age_mean, axis=0, inplace=True)
titanic.dropna(axis=0, inplace=True)
titanic.age.describe()

titanic.age.plot.kde()
plt.show()
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Présentation

Dataframes

Manipulations de  
dataframes

Sympy

Analyse de réseaux

Machine learning /  
statistiques

# Manipulations de dataframes

Matthieu Falce

## Exemples de graphiques possibles

```
import numpy as np
import pandas as pd
from pandas.plotting import lag_plot
from pandas.plotting import autocorrelation_plot
from matplotlib import pyplot as plt

xs = np.linspace(-1, 1, 100)
ys = np.cos(xs)
ys2 = np.random.random(100)

df = pd.DataFrame({"cos": ys, "rand": ys2})

lag_plot(df.cos)
plt.show()
lag_plot(df.rand)
plt.show()

autocorrelation_plot(df.rand)
plt.show()
autocorrelation_plot(df.cos)
plt.show()

ax = df.cos.plot()
fig = ax.get_figure()
fig.savefig("cos.png")
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Présentation

Dataframes

Manipulations de  
dataframes

Sympy

Analyse de réseaux

Machine learning /  
statistiques

# Manipulations de dataframes

Matthieu Falce

## Opération sur des groupes de données

```
import pandas as pd
import numpy as np

df = pd.DataFrame(
    [
        ("bird", "Falconiformes", 389.0),
        ("bird", "Psittaciformes", 24.0),
        ("mammal", "Carnivora", 80.2),
        ("mammal", "Primates", np.nan),
        ("mammal", "Carnivora", 58),
    ],
    index=["falcon", "parrot", "lion", "monkey", "leopard"],
    columns=("class", "order", "max_speed"),
)

grouped1 = df.groupby("class")
grouped2 = df.groupby("order", axis="columns")
grouped3 = df.groupby(["class", "order"])

for n, g in grouped3:
    print(n)
    print(g)
    print(type(g))
    print("")
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Présentation

Dataframes

Manipulations de  
dataframes

Sympy

Analyse de réseaux

Machine learning /  
statistiques

# Manipulations de dataframes

Matthieu Falce

## Manipulation de séries temporelles

```
import pandas as pd
import numpy as np

# time index
dti = pd.date_range("2018-01-13", periods=3, freq="H")
dti = dti.tz_localize("UTC")
dti.tz_convert("US/Pacific")

## offsets
# https://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html#timeseries-offset-aliases
start, end = "2019-01-12", "2019-12-25"
pd.date_range(start, end, freq="BM")

# conversion
## https://docs.python.org/3/library/datetime.html#strftime-and-strptime-behavior
pd.to_datetime("12-11-2010 00:00", format="%d-%m-%Y %H:%M")

# resampling
idx = pd.date_range("2018-01-01", periods=48, freq="H")
ts = pd.Series(range(len(idx)), index=idx)
ts.resample("2H").mean()

s = pd.Series(range(len(idx)), index=idx)
for i in s.resample("6H"):
    print(i)
```

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Création de modules

Interface graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Présentation

Dataframes

Manipulations de dataframes

Sympy

Analyse de réseaux

Machine learning / statistiques

# Indexation (représentation graphique)

Plus d'informations sur les différents modes d'accès ici :

<https://stackoverflow.com/questions/28757389/pandas-loc-vs-iloc-vs-at-vs-iat>

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Présentation

Dataframes

Manipulations de  
dataframes

Sympy

Analysé de réseaux

Machine learning /  
statistiques

## Python Pandas Selections and Indexing

### .iloc selections - position based selection

data.iloc[<row selection>], <column selection>]

Integer list of rows: [0,1,2]

Slice of rows: [4:7]

Single values: 1

Integer list of columns: [0,1,2]

Slice of columns: [4:7]

Single column selections: 1

### loc selections - position based selection

data.loc[<row selection>], <column selection>]

Index/Label value: 'john'

List of labels: ['john', 'sarah']

Logical/Boolean index: data['age'] == 10

Named column: 'first\_name'

List of column names: ['first\_name', 'age']

Slice of columns: 'first\_name':'address'

Source : <https://www.shanelynn.ie/select-pandas-dataframe-rows-and-columns-using-iloc-loc-and-ix/>

# Indexation (représentation graphique)

Matthieu Falce

Plus d'informations sur les différents modes d'accès ici :

<https://stackoverflow.com/questions/28757389/pandas-loc-vs-iloc-vs-at-vs-iat>

## Pandas Select Row

	Morning	Noon	Evening	Midnight	
df.iloc[2]	1999-12-30	1.764052	0.400157	0.978738	2.240893
df.loc['2000-01-01']	1999-12-31	1.867558	-0.977278	0.950088	-0.151357
	2000-01-01	-0.103219	0.410599	0.144044	1.454274
	2000-01-02	0.761038	0.121675	0.443863	0.333674
	2000-01-03	1.494079	-0.205158	0.313068	-0.854096
	2000-01-04	-2.552990	0.653619	0.864436	-0.742165
	2000-01-05	2.269755	-1.454366	0.045759	-0.187184

df.loc[2]  
df.loc['2000-01-01']

df.loc[2]  
df.loc['2000-01-01']

© Matt Harasymczuk, 2020, CC-BY-SA-4.0

Source : <https://python.astrotech.io/pandas/dataframe/loc.html>

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique  
Jupyter

Numpy

Scipy

Pandas

Présentation

Dataframes

Manipulations de  
dataframes

Sympy

Analyse de réseaux

Machine learning /  
statistiques

# Indexation (représentation graphique)

Matthieu Falce

Plus d'informations sur les différents modes d'accès ici :

<https://stackoverflow.com/questions/28757389/pandas-loc-vs-iloc-vs-at-vs-iat>

Pandas Select Column

	Morning	Noon	Evening	Midnight
1999-12-30	1.764052	0.400157	0.978738	2.240893
1999-12-31	1.867558	-0.977278	0.950088	-0.151357
2000-01-01	-0.103219	0.410599	0.144044	1.454274
2000-01-02	0.761038	0.121675	0.443863	0.333674
2000-01-03	1.494079	-0.205158	0.313068	-0.854096
2000-01-04	-2.552990	0.653619	0.864436	-0.742165
2000-01-05	2.269755	-1.454366	0.045759	-0.187184

`df['Evening']`  
`df.Evening`  
`df.loc[:, 'Evening']`

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Présentation

Dataframes

Manipulations de  
dataframes

Sympy

Analyse de réseaux

Machine learning /  
statistiques

© Matt Harasymczuk, 2020, CC-BY-SA-4.0

Source : <https://python.astrotech.io/pandas/dataframe/loc.html>

# 9- Python scientifique

## 9.6. Sympy

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

**Sympy**

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

Performances

Dask

Bibliographie

Sympy<sup>63</sup> est une bibliothèque permettant le calcul symbolique :

- ▶ simplification d'expression
- ▶ analyse (dérivation, intégration)
- ▶ affichage des sorties en  $\text{\LaTeX}$

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

**Sympy**

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

Performances

Dask

Bibliographie

---

63.<http://www.sympy.org>

# Calcul symbolique

Matthieu Falce

```
from sympy import *
# permet l'affichage des formules
init_session()

# on déclare des variables
x, a, b = symbols("x a b")

# on définit une intégrale
a = Integral(cos(x) * exp(x), x)
print(a)
latex(a) # on affiche son code latex

# on va simplifier des expressions
simplify(sin(x) ** 2 + cos(x) ** 2)
simplify(x ** a * x ** b)
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

Performances

Dask

Bibliographie

## 9- Python scientifique

### 9.7. Analyse de réseaux

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

**Analyse de réseaux**

Machine learning /  
statistiques

Graphiques

Performances

Dask

Bibliographie

# Présentation

Matthieu Falce

Il existe plusieurs bibliothèques pour manipuler des graphes (réseaux) en python :

- ▶ **igraph** <sup>64</sup>
- ▶ **networkx** <sup>65</sup>
- ▶ **graph-tool** <sup>66</sup>

La plus connue est networkX mais peut être lente (codée en pur python), les autres sont potentiellement plus rapide.

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

Performances

Dask

Bibliographie

---

64.<http://igraph.org>

65.<http://networkx.github.io>

66.<http://graph-tool.skewed.de>

# Manipulation de graphes

Matthieu Falce

```
import networkx as nx

# on crée un graphe en 2 parties
G = nx.Graph()
G.add_edges_from([(1, 2), (1, 3)])
G.add_node(4)

# on calcule des propriétés du graphe
nx.connected_components(G)
list(nx.connected_components(G))
sorted(d for n, d in G.degree)
nx.clustering(G)
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

Performances

Dask

Bibliographie

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

**Machine learning /  
statistiques**

Graphiques

Performances

Dask

Bibliographie

## 9- Python scientifique

### 9.8. Machine learning / statistiques

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

Performances

Dask

Bibliographie

Il existe plusieurs bibliothèques dédiées apprentissage :

- ▶ modèles statistiques (régressions, tests statistiques, méthodes sur séries temporelles) : `statsmodels`<sup>67</sup>
- ▶ algorithmes de *machine learning* : `scikit-learn`<sup>68</sup>
- ▶ *curve feating* : `prophet`<sup>69</sup> (analyse de séries temporelles)
- ▶ *deep learning* : `tensorflow`<sup>70</sup>, `keras`<sup>71</sup>

---

67.<https://www.statsmodels.org/stable/index.html>

68.<https://scikit-learn.org/stable/>

69.<https://facebook.github.io/prophet/>

70.<https://www.tensorflow.org/>

71.<https://keras.io/>

# 9- Python scientifique

## 9.9. Graphiques

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

**Graphiques**

Matplotlib

Personnalisation

Seaborn

# Graphiques

Matthieu Falce

1. amélioration et diversification des outils
2. réalisation de graphiques de qualité
3. écosystème riche (seuls R et Javascript ont de meilleures bibliothèques à mon avis)

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

**Graphiques**

Matplotlib

Personnalisation

Seaborn

# Matplotlib

Matthieu Falce

Bibliothèques de plot la plus célèbre

API inspirée de Matlab

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

**Matplotlib**

Personnalisation

Seaborn

# Matplotlib

Bibliothèques de plot la plus célèbre

API inspirée de Matlab

```
from matplotlib import pyplot as plt
import numpy as np

xs = np.linspace(-2*np.pi, 2*np.pi, 100)
ys1 = np.sinc(xs)
ys2 = np.sin(xs)

plt.plot(ys1, c="r", label="Sinc")
plt.plot(ys2, c="b", label="Sin")
plt.xlabel("X")
plt.ylabel("f(x)")
plt.legend()
plt.show()
```

plt agit comme une machine à état

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

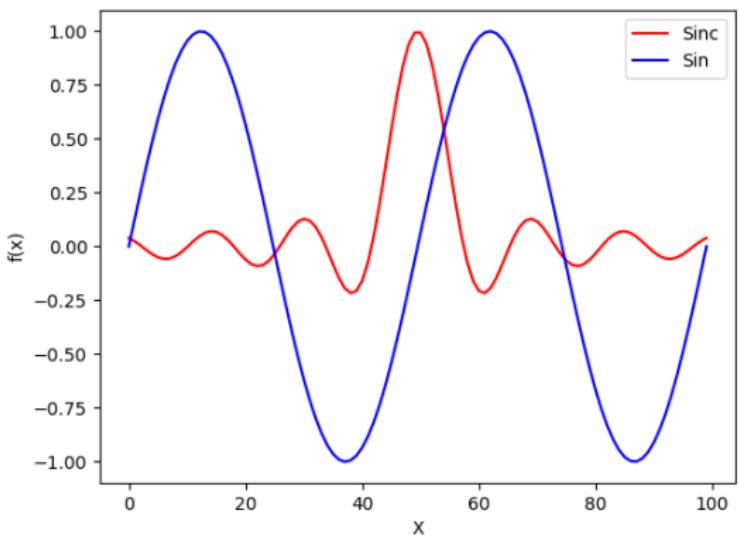
Matplotlib

Personnalisation

Seaborn

## Bibliothèques de plot la plus célèbre

API inspirée de Matlab  
Le style aussi



Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standardCréation de  
modulesInterface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

**Matplotlib**

Personnalisation

Seaborn

# Matplotlib – types de plots

Matplotlib est une bibliothèque graphique → peut tout faire.

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

**Matplotlib**

Personnalisation

Seaborn

# Matplotlib – types de plots

Matplotlib est une bibliothèque graphique → peut tout faire.



Si l'on s'en donne la peine

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

**Matplotlib**

Personnalisation

Seaborn

# Matplotlib – types de plots

Matplotlib est une bibliothèque graphique → peut tout faire.



Si l'on s'en donne la peine

- ▶ lineplot
- ▶ scatterplot
- ▶ cartographie
- ▶ hexbin
- ▶ nuages de mots
- ▶ 3D
- ▶ animations
- ▶ ...

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

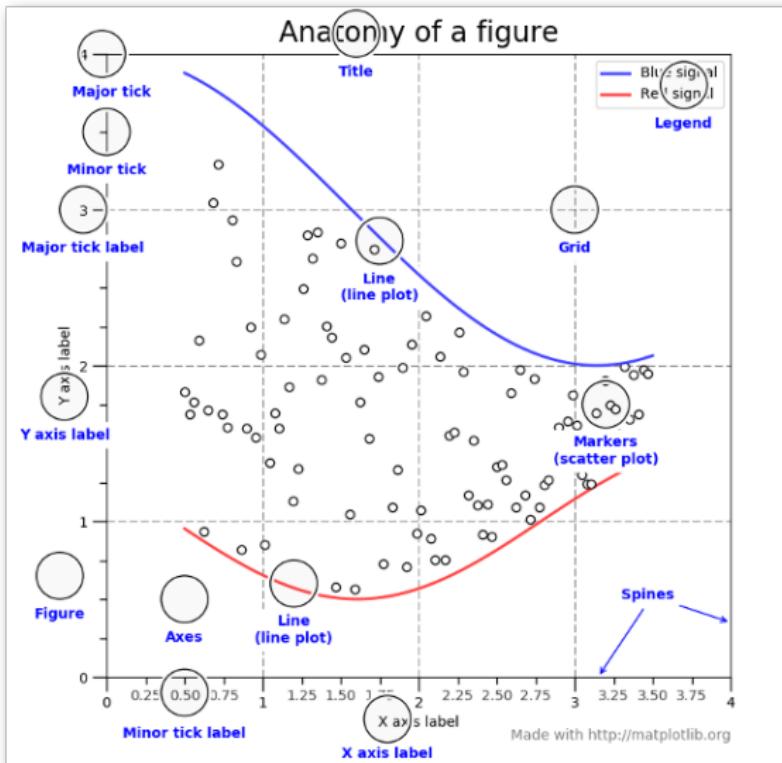
**Matplotlib**

Personnalisation

Seaborn

# Anatomie d'une figure

Matthieu Falce



[https://matplotlib.org/faq/usage\\_faq.html](https://matplotlib.org/faq/usage_faq.html)

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

Matplotlib

Personnalisation

Seaborn

# Anatomie d'une figure

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

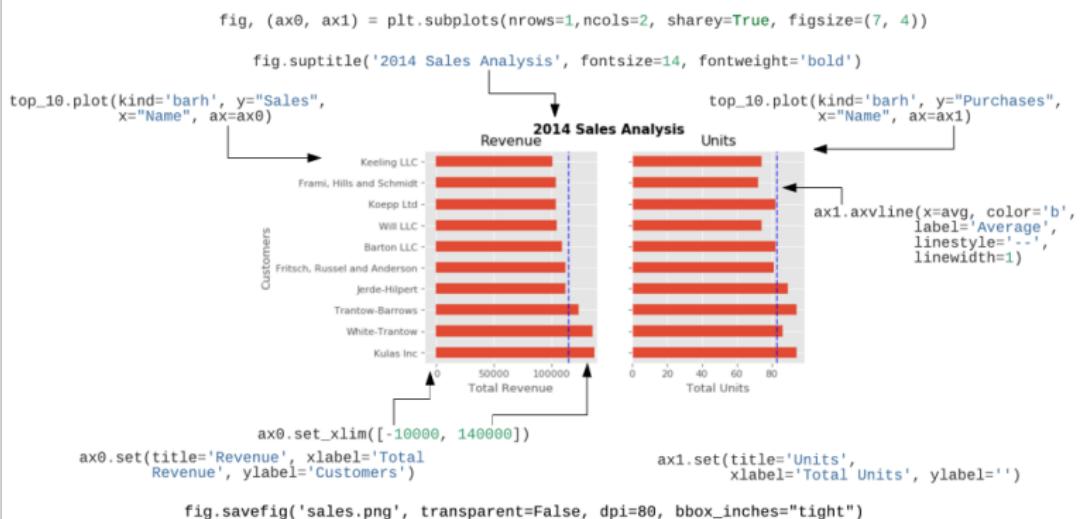
Machine learning /  
statistiques

Graphiques

Matplotlib

Personnalisation

Seaborn



pbpython.com

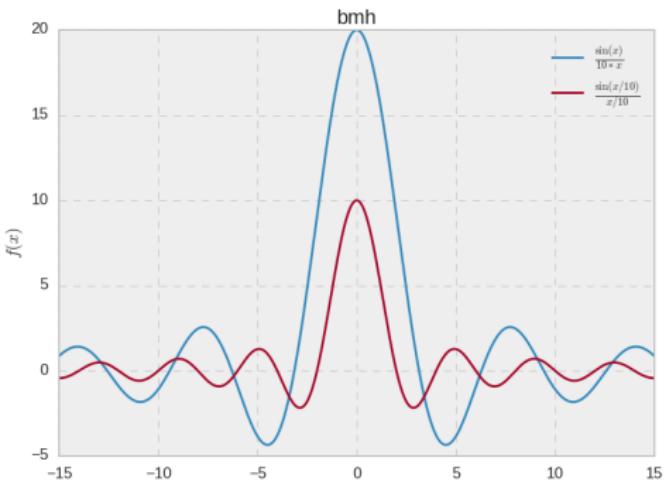
<http://pbpython.com/effective-matplotlib.html>

# Personnalisation

Matthieu Falce

Vous pouvez surcharger la configuration de Matplotlib :

- ▶ passer des paramètres à la fonction de dessin
- ▶ changer de style
- ▶ modifier le `pyplot.rcParams`



Thème : `bmh`

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

Matplotlib

Personnalisation

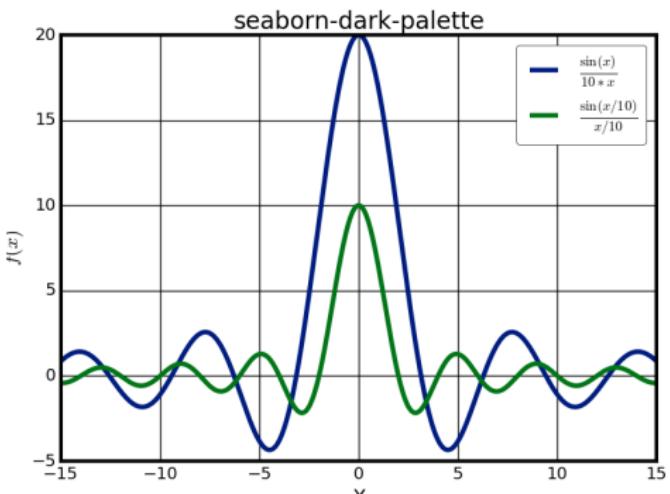
Seaborn

# Personnalisation

Matthieu Falce

Vous pouvez surcharger la configuration de Matplotlib :

- ▶ passer des paramètres à la fonction de dessin
- ▶ changer de style
- ▶ modifier le `pyplot.rcParams`



Thème : seaborn-dark-palette

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

Matplotlib

Personnalisation

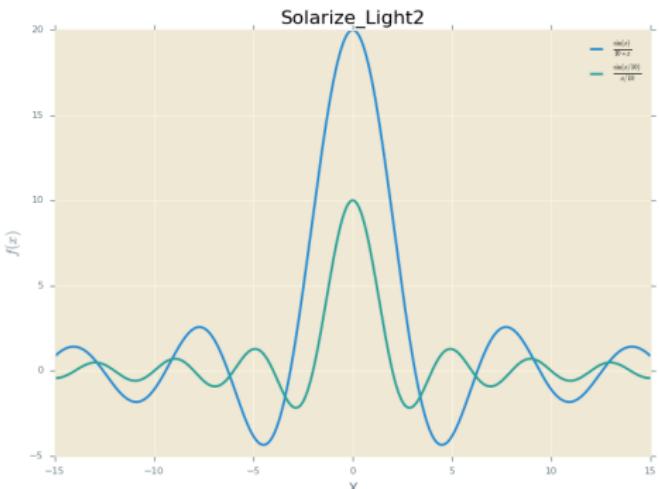
Seaborn

# Personnalisation

Matthieu Falce

Vous pouvez surcharger la configuration de Matplotlib :

- ▶ passer des paramètres à la fonction de dessin
- ▶ changer de style
- ▶ modifier le `pyplot.rcParams`



Thème : Solarize\_Light2

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

Matplotlib

Personnalisation

Seaborn

# Personnalisation

Matthieu Falce

Vous pouvez surcharger la configuration de Matplotlib :

- ▶ passer des paramètres à la fonction de dessin
- ▶ changer de style
- ▶ modifier le `pyplot.rcParams`

```
from matplotlib import pyplot as plt
import numpy as np

plt.rcParams['font.family'] = 'serif'
plt.rcParams['font.serif'] = 'Ubuntu'
plt.rcParams['ytick.labelsize'] = 10
plt.rcParams['legend.fontsize'] = 10
plt.rcParams['figure.titlesize'] = 20
plt.rcParams['lines.linewidth'] = 10

xs = np.linspace(-15, 15, 1000)
ys1 = 20 * np.sin(xs) / xs
ys2 = 10 * np.sinc(xs / 2)

plt.plot(xs, ys1, label=r"\frac{\sin(x)}{10 * x}")
plt.plot(xs, ys2, label=r"\frac{\sin(x/10)}{x/10}")

plt.title("Courbes")
plt.legend()
plt.xlabel("X")
plt.ylabel("$f(x)$")
plt.title("Surcharge rcParams")
plt.savefig("styles/rcParams.png")
plt.show()
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

Matplotlib

Personnalisation

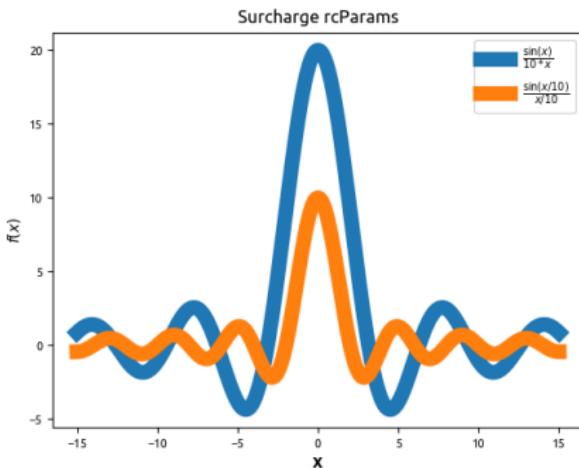
Seaborn

# Personnalisation

Matthieu Falce

Vous pouvez surcharger la configuration de Matplotlib :

- ▶ passer des paramètres à la fonction de dessin
- ▶ changer de style
- ▶ modifier le `pyplot.rcParams`



Surcharge de `rcParams`

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

Matplotlib

Personnalisation

Seaborn

# Seaborn

Matthieu Falce

Seaborn se base sur matplotlib.

Il rajoute des styles et des fonctionnalités (surtout utile en statistiques).

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

Matplotlib

Personnalisation

Seaborn

# Seaborn

Matthieu Falce

Seaborn se base sur matplotlib.

Il rajoute des styles et des fonctionnalités (surtout utile en statistiques).

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

```
R = np.random.random((5, 5))
sns.heatmap(R)
plt.savefig("sns_heatmap.png")
```

```
plt.clf()
A = np.random.normal(10, 1, 100)
B = np.random.normal(6, 5, 100)
sns.boxplot(x=["A", "B"], y=[A, B])
plt.savefig("sns_boxplot.png")
```

```
plt.clf()
sns.kdeplot(A, shade=True, label="A")
sns.distplot(B, label="B")
plt.legend()
plt.savefig("sns_distplot.png")
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

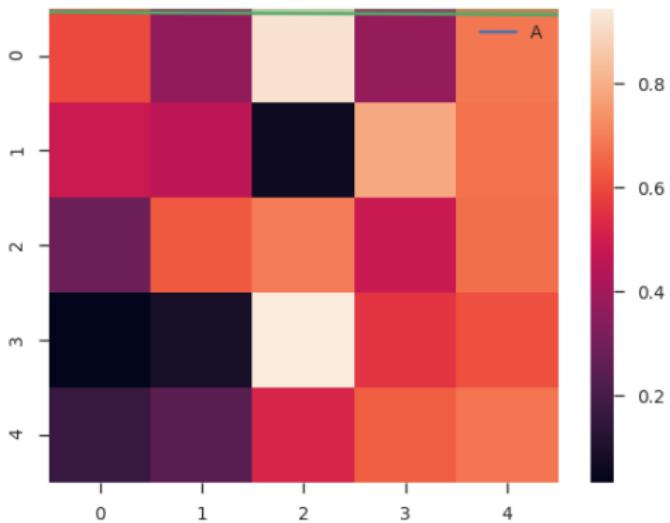
Matplotlib

Personnalisation

Seaborn

Seaborn se base sur matplotlib.

Il rajoute des styles et des fonctionnalités (surtout utile en statistiques).



Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

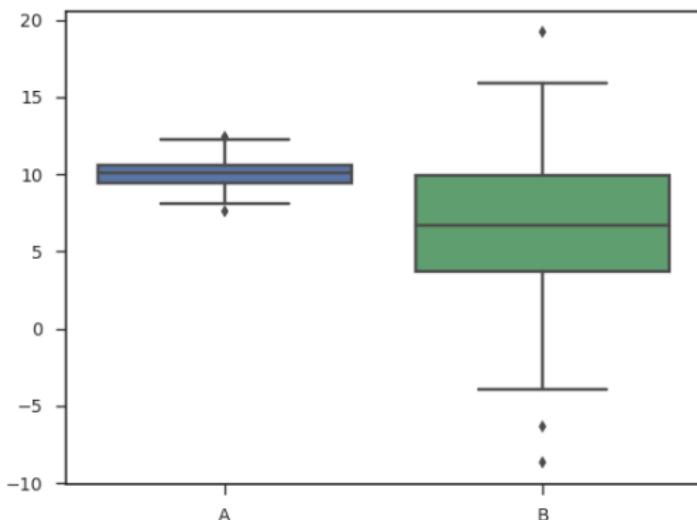
Matplotlib

Personnalisation

Seaborn

Seaborn se base sur matplotlib.

Il rajoute des styles et des fonctionnalités (surtout utile en statistiques).



Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

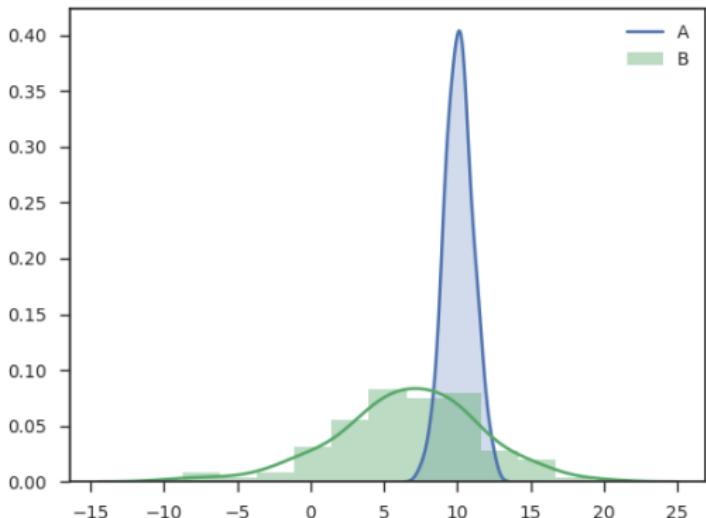
Matplotlib

Personnalisation

Seaborn

Seaborn se base sur matplotlib.

Il rajoute des styles et des fonctionnalités (surtout utile en statistiques).



Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

Matplotlib

Personnalisation

Seaborn

# Concurrents 2D

Matthieu Falce

Matplotlib n'est pas la seule bibliothèque de graphiques pour Python :

- ▶ (seaborn)
- ▶ plotly (figures web interactives)
- ▶ mplot3d (transforme une figure mpl en Javascript)
- ▶ bokeh (figures web interactive)
- ▶ plotly (figures web interactive)
- ▶ ggplot (port de la bibliothèque ggplot2 de R)

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

Matplotlib

Personnalisation

Seaborn

# Concurrent : *plotly*

Matthieu Falce

- ▶ interactive
- ▶ web charts

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

Matplotlib

Personnalisation

Seaborn

# Concurrent : *plotly*

Matthieu Falce

- ▶ interactive
- ▶ web charts

```
import plotly.express as px
```

```
xs = [i for i in range(100)]
fig = px.scatter(x=xs, y=[i ** 2 for i in xs])
fig.show()
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

Matplotlib

Personnalisation

Seaborn

# Concurrent : *plotly*

Matthieu Falce

```
import plotly.graph_objects as go\n\nfig = go.Figure(\n    data=go.Scatter(\n        x=[1, 2, 3, 4],\n        y=[10, 11, 12, 13],\n        mode="markers",\n        marker=dict(\n            size=[40, 60, 80, 100],\n            color=[0, 1, 2, 3]),\n    ))\n\nfig.show()
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

Matplotlib

Personnalisation

Seaborn

# Concurrent : *plotly*

Matthieu Falce

```
import matplotlib.pyplot as plt
import plotly
from plotly.tools import mpl_to_plotly

fig, ax = plt.subplots()
ax.plot([1, 2, 3], [1, 4, 9], "o")

plotly_fig = mpl_to_plotly(fig)
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

Matplotlib

Personnalisation

Seaborn

On peut utiliser Dash<sup>72</sup>

- ▶ utilise plotly
- ▶ basé sur le framework web flask
- ▶ permet de créer des dashboards web interactifs sans faire de HTML / JS
- ▶ bindings en R et Python
- ▶ exemples : <https://dash-gallery.plotly.host/Portal/>

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

Matplotlib

Personnalisation

Seaborn

---

72.<https://plot.ly/dash/>

# Graphiques 3D

Matthieu Falce

```
# source
# https://matplotlib.org/examples/mplot3d/lines3d_demo.html

import matplotlib as mpl
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import matplotlib.pyplot as plt

mpl.rcParams['legend.fontsize'] = 10

fig = plt.figure()
ax = fig.gca(projection='3d')
theta = np.linspace(-4 * np.pi, 4 * np.pi, 100)
z = np.linspace(-2, 2, 100)
r = z**2 + 1
x = r * np.sin(theta)
y = r * np.cos(theta)
ax.plot(x, y, z, label='parametric curve')
ax.legend()

plt.savefig("test_3d.png")
plt.show()
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

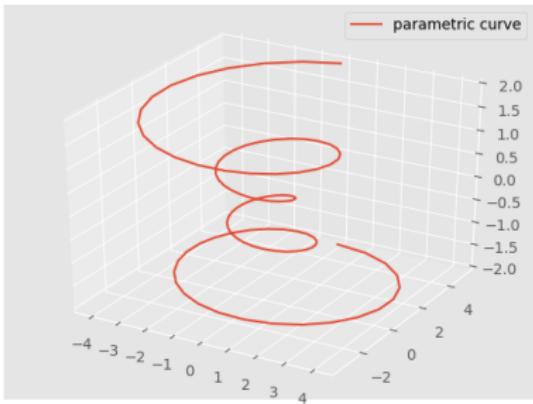
Matplotlib

Personnalisation

Seaborn

# Graphiques 3D

Matthieu Falce



## Résultat graphique 3D

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

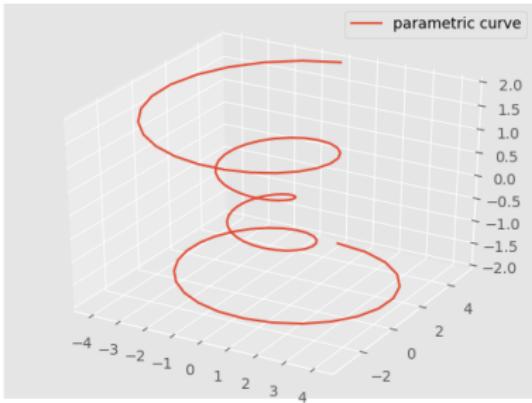
Matplotlib

Personnalisation

Seaborn

# Graphiques 3D

Matthieu Falce



Résultat graphique 3D



Ce n'est pas de la "vraie" 3D... (pas de notion de volumes)

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

Matplotlib

Personnalisation

Seaborn

# Graphiques 3D

Matthieu Falce

- Pour faire de la vraie 3d :
  - ▶ mayavi
  - ▶ <https://lorensen.github.io/VTKExamples/site/Python/>
  - ▶ (ParaView)
  - ▶ moteurs de jeu 3D
- Vue d'ensemble
- Langage Python
- Programmation Orientée objet (POO)
- Bonnes pratiques
- Bibliothèque standard
- Création de modules
- Interface graphiques
- Code natif
- Python scientifique
  - Écosystème scientifique
  - Jupyter
  - Numpy
  - Scipy
  - Pandas
  - Sympy
  - Analyse de réseaux
  - Machine learning / statistiques
  - Graphiques
    - Matplotlib
    - Personnalisation
    - Seaborn

## 9- Python scientifique

### 9.10. Performances

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

**Performances**

Numexpr

Dask



## N'optimisez que ce qui est nécessaire

- ▶ faites des tests de performances (“profiling”)
- ▶ n'optimisez que ce qui est nécessaire
- ▶ ne commencez que quand tout fonctionne et est testé
- ▶ évitez les copies et les mauvaises structures mémoires
- ▶ utilisez de bons algorithmes
- ▶ préférez les méthodes de Scipy souvent plus rapide que celles de Numpy
- ▶ zen of Numpy
- ▶ ...

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

Performances

Numexpr

Dask

Les calculs Numpy se font en générant des tableaux intermédiaires. Numexpr permet de les supprimer en effectuant les calculs directement

```
import numpy as np
import numexpr as ne

a = np.arange(1e6)
b = np.arange(1e6)

c = ne.evaluate("a + 1")
# %timeit c = ne.evaluate("a + 1")
# 866 µs ± 74.6 µs per loop
# (mean ± std. dev. of 7 runs, 1000 loops each)

# %timeit c = a + 1
# 845 µs ± 37.2 µs per loop
# (mean ± std. dev. of 7 runs, 1000 loops each)

d = ne.evaluate("sin(a) + arcsinh(a/b)")
# %timeit np.sin(a) + np.arcsinh(a/b)
# The slowest run took 6.65 times longer than the fastest.
# This could mean that an intermediate result is being cached.
# 154 ms ± 139 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)

# %timeit ne.evaluate("sin(a) + arcsinh(a/b)")
# 66.2 ms ± 2.11 ms per loop
# (mean ± std. dev. of 7 runs, 10 loops each)
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

Performances

Numexpr

Dask

# 9- Python scientifique

## 9.11. Dask

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

Performances

**Dask**

Autres

## Framework de parallélisme.

Pour les *medium data* (ne tient plus en RAM mais sur un SSD)

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

Performances

Dask

Autres

## Framework de parallélisme.

Pour les *medium data* (ne tient plus en RAM mais sur un SSD)

S'intègre avec (en réutilisant les mêmes API) :

- ▶ numpy
- ▶ pandas
- ▶ scikit learn

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

Performances

Dask

Autres

## Framework de parallélisme.

Pour les *medium data* (ne tient plus en RAM mais sur un SSD)

S'intègre avec (en réutilisant les mêmes API) :

- ▶ numpy
- ▶ pandas
- ▶ scikit learn

2 concepts :

- ▶ scheduler : exécute des graphes de calculs (comme make, luigi, celery...)
- ▶ big data collections : partitionnement des données ne tenant pas en RAM

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

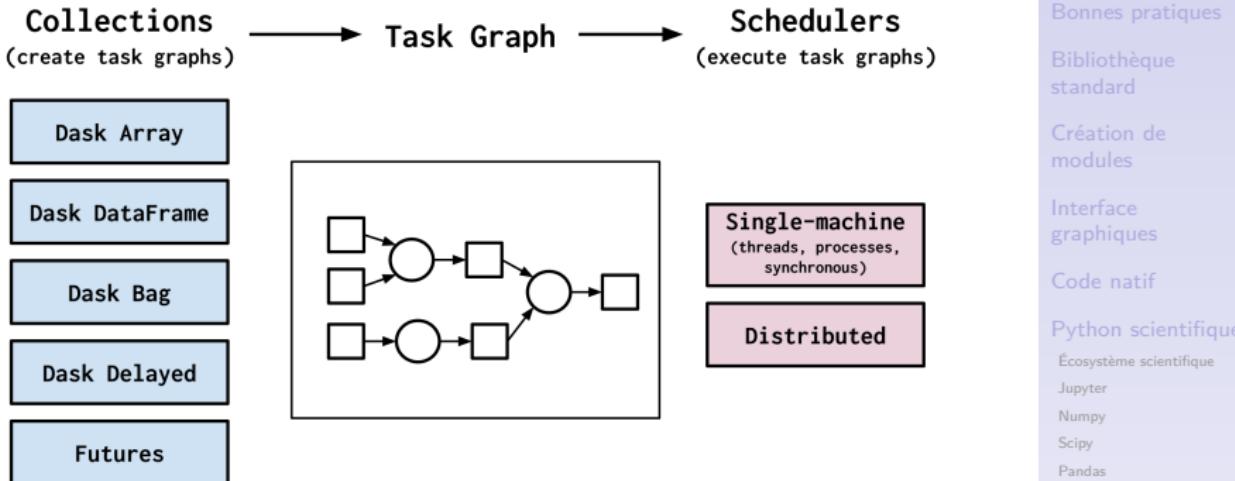
Performances

Dask

Autres

## Dask : vue d'ensemble

Source: <https://docs.dask.org/en/latest/>



Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

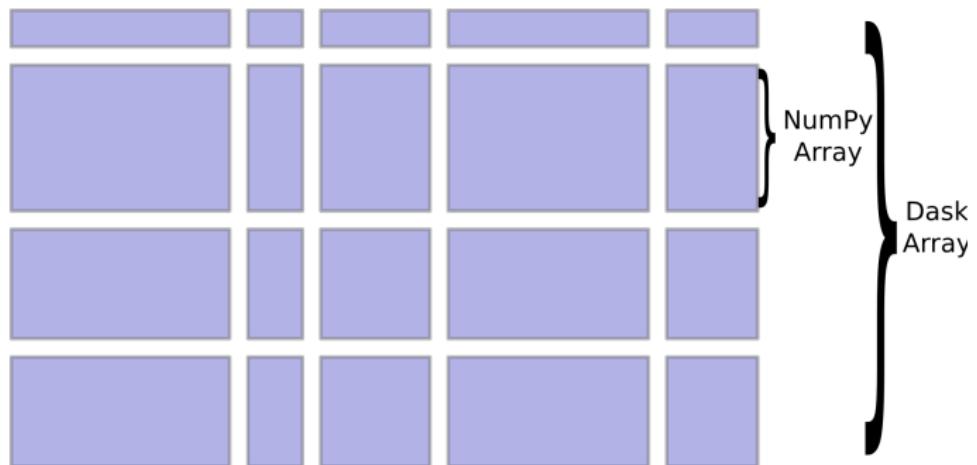
Performances

Dask

Autres

## Dask : structure d'un *dask array*

Source: <https://docs.dask.org/en/latest/array.html>



Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

Performances

**Dask**

Autres

# Dask

Matthieu Falce

```
from dask.distributed import Client, progress

client = Client(
    n_workers=2,
    threads_per_worker=2,
    memory_limit="1GB"
) # workers configuration
# we can change for process workers
# to deal with GIL perf issues

# go to : http://127.0.0.1:8787
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

Performances

**Dask**

Autres

# Dask

Matthieu Falce

```
# source : https://examples.dask.org/dataframe.html
import dask
import dask.dataframe as dd

# lazy operation, they are only performed when we need the result
df = dask.datasets.timeseries()
df.head()

df2 = df[df.y > 0]
df3 = df2.groupby("name").x.std()

computed_df = df3.compute()
type(computed_df)

df[["x", "y"]].resample("24h").mean().compute().plot()
df[["x", "y"]].rolling(window="24h").mean().head()

# display the call graph
df[["x", "y"]].resample("24h").mean().visualize()

# store in RAM for faster computation
df = df.persist()
```

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

Performances

Dask

Autres

# Autres techniques

Matthieu Falce

D'autres techniques, plus ou moins matures permettent d'améliorer les temps de calculs également :

- ▶ Numba
- ▶ Pythran
- ▶ (Theano)
- ▶ distribution python par Intel

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

Performances

Dask

Autres

## 9- Python scientifique

### 9.12. Bibliographie

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

Performances

Dask

Bibliographie

# Bibliographie I

Matthieu Falce

## ► graphiques

- ▶ <http://www.labri.fr/perso/nrougier/teaching/matplotlib/matplotlib.html#id8>
- ▶ <https://python-graph-gallery.com/matplotlib/>
- ▶ <https://matplotlib.org/gallery.html>
- ▶ <http://pbpython.com/effective-matplotlib.html>
- ▶ [https://matplotlib.org/faq/usage\\_faq.html](https://matplotlib.org/faq/usage_faq.html)
- ▶ <http://futurile.net/2016/02/27/matplotlib-beautiful-plots-with-style/#id16>

## ► numpy

- ▶ <http://www.scipy-lectures.org/numpy/numpy.html>
- ▶ [http://www.scipy-lectures.org/advanced/advanced\\_numpy/#block-of-memory](http://www.scipy-lectures.org/advanced/advanced_numpy/#block-of-memory)
- ▶ <https://docs.scipy.org/doc/numpy/reference/internals.html>

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

Performances

Dask

Bibliographie

# Bibliographie II

Matthieu Falce

- ▶ <https://jakevdp.github.io/PythonDataScienceHandbook/02.01-understanding-data-types.html>
- ▶ <http://www.labri.fr/perso/nrougier/teaching/numpy/numpy.html>
- ▶ <http://www.labri.fr/perso/nrougier/teaching/numpy.100/index.html>
- ▶ **scipy**
  - ▶ <https://scipy-cookbook.readthedocs.io/>
  - ▶ <https://docs.scipy.org/doc/scipy/reference/tutorial/index.html>
  - ▶ <https://docs.scipy.org/doc/scipy/reference/index.html>
  - ▶ <https://makina-corpus.com/blog/metier/2017/presentation-de-lecosystème-python-scientifique>

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

Performances

Dask

Bibliographie

# Bibliographie III

Matthieu Falce

## ► pandas

- ▶ la feuille de triche pandas officielle :  
[https://github.com/pandas-dev/pandas/blob/master/doc/cheatsheet/Pandas\\_Cheat\\_Sheet.pdf](https://github.com/pandas-dev/pandas/blob/master/doc/cheatsheet/Pandas_Cheat_Sheet.pdf)
- ▶ inline vs copy operations : [https://pandas.pydata.org/pandas-docs/stable/user\\_guide/indexing.html#indexing-view-versus-copy](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#indexing-view-versus-copy)
- ▶ les explications sur les différentes méthodes d'indexation : <https://stackoverflow.com/questions/28757389/pandas-loc-vs-iloc-vs-at-vs-iat>
- ▶ <https://pandas.pydata.org/pandas-docs/stable/dsintro.html>
- ▶ <https://pandas.pydata.org/pandas-docs/stable/visualization.html>
- ▶ [http://falce.net/presentation/python\\_pandas\\_manoaco\\_parking](http://falce.net/presentation/python_pandas_manoaco_parking)
- ▶ [https://pandas.pydata.org/pandas-docs/stable/user\\_guide/cookbook.html#cookbook-resample](https://pandas.pydata.org/pandas-docs/stable/user_guide/cookbook.html#cookbook-resample)

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

Performances

Dask

Bibliographie

# Bibliographie IV

Matthieu Falce

## ► dask

- ▶ tutoriel sur comment charger de grandes quantités de données : <https://blog.dask.org/2019/06/20/load-image-data>
- ▶ notebooks d'exemples / tutos en lignes :  
<https://hub-binder.mybinder.ovh/user/dask-dask-examples-irbwzcm1/lab>
- ▶ cas d'usages réels :  
<https://stories.dask.org/en/latest/>
- ▶ spark vs dask vs base de données :  
<https://docs.dask.org/en/latest/spark.html>
- ▶ mise en place + vidéo :  
<https://docs.dask.org/en/latest/setup.html>

## ► manipulation d'images

- ▶ documentation d'openCV : [https://docs.opencv.org/master/d7/d4d/tutorial\\_py\\_thresholding.html](https://docs.opencv.org/master/d7/d4d/tutorial_py_thresholding.html)

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

Performances

Dask

Bibliographie

# Bibliographie V

Matthieu Falce

- ▶ les exemples sur `scipy-lectures.org/packages/scikit-image/index.html` et `http://scipy-lectures.org/advanced/image_processing/`

Vue d'ensemble

Langage Python

Programmation  
Orientée objet  
(POO)

Bonnes pratiques

Bibliothèque  
standard

Création de  
modules

Interface  
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /  
statistiques

Graphiques

Performances

Dask

Bibliographie