

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Formation Python pour OTS et MNP

Matthieu Falce

Novembre 2022

Au programme I

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Au programme II

Python scientifique

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

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

Interface
graphiques

Code natif

Python scientifique

- ▶ Qui suis-je ?

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

- ▶ 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

- ▶ 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

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

Où me trouver ?

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

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

Matthieu Falce

Vue d'ensemble

Historique
Philosophie
Open Source
Python, CPython, ...
Calcul scientifique
Cas d'utilisations de python
Installation
Environnement de développement

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

Vue d'ensemble

Un vieux langage ?

Matthieu Falce

Vue d'ensemble

Historique
Philosophie
Open Source
Python, CPython, ...
Calcul scientifique
Cas d'utilisations de python
Installation
Environnement de développement

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

Un vieux langage ?

Matthieu Falce

- ▶ Créateur (et bdfi) : Guido van Rossum
- ▶ 1ère version : 20 février 1991
- ▶ dernière version stable sortie : 3.10.7 (7 septembre 2022)



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

Vue d'ensemble

Historique
Philosophie
Open Source
Python, CPython, ...
Calcul scientifique
Cas d'utilisations de python
Installation
Environnement de développement

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

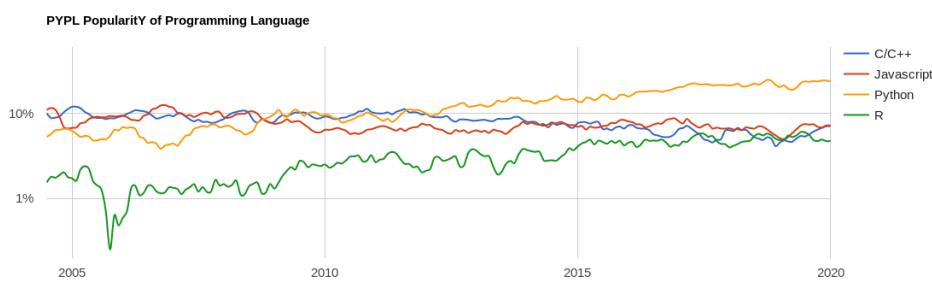
Code natif

Python scientifique

Un vieux langage ?

Matthieu Falce

- ▶ Créateur (et bdfi) : Guido van Rossum
- ▶ 1ère version : 20 février 1991
- ▶ dernière version stable sortie : 3.10.7 (7 septembre 2022)



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

Vue d'ensemble

Historique
Philosophie
Open Source
Python, CPython, ...
Calcul scientifique
Cas d'utilisations de python
Installation
Environnement de développement

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

Origine du nom

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

Matthieu Falce

Vue d'ensemble

Historique
Philosophie
Open Source
Python, CPython, ...
Calcul scientifique
Cas d'utilisations de python
Installation
Environnement de développement

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque standard

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
Open Source
Python, CPython, ...
Calcul scientifique
Cas d'utilisations de python
Installation
Environnement de développement

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque standard

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
Open Source
Python, CPython, ...
Calcul scientifique
Cas d'utilisations de python
Installation
Environnement de développement

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

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

```
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)

    ...

    def conjugate(self):
        return complex(self.re, -self.im)

    def __nonzero__(self):
        return self.re != 0 or self.im != 0

    def __len__(self):
        return 2
```

Vue d'ensemble

Historique
Philosophie
Open Source
Python, CPython, ...
Calcul scientifique
Cas d'utilisations de python
Installation
Environnement de développement

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Python 2 vs Python 3

Matthieu Falce

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
- ▶ ...

Vue d'ensemble

Historique
Philosophie
Open Source
Python, CPython, ...
Calcul scientifique
Cas d'utilisations de python
Installation
Environnement de développement

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

Python 2 vs Python 3

Matthieu Falce

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
Open Source
Python, CPython, ...
Calcul scientifique
Cas d'utilisations de python
Installation
Environnement de développement

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

Python 2 End Of Life

Fin du support de Python le 1er janvier 2020



Matthieu Falce

Vue d'ensemble

Historique
Philosophie
Open Source
Python, CPython, ...
Calcul scientifique
Cas d'utilisations de python
Installation
Environnement de développement

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

Python 2 End Of Life

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/>

Matthieu Falce

Vue d'ensemble

Historique
Philosophie
Open Source
Python, CPython, ...
Calcul scientifique
Cas d'utilisations de python
Installation
Environnement de développement

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

Zen of Python

Matthieu Falce

Vue d'ensemble

Historique

Philosophie

Open Source

Python, CPython, ...

Calcul scientifique

Cas d'utilisations de python

Installation

Environnement de développement

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque standard

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 20¹)²

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

Open Source

Python, CPython, ...

Calcul scientifique

Cas d'utilisations de python

Installation

Environnement de développement

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque standard

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/>

Open Source – Définition

Matthieu Falce

- ▶ logiciels dont la licence respecte les critères de l'*Open Source Initiative*
 - ▶ s'oppose au logiciel *fermé, propriétaire, privatiseur*
 - ▶ proche du concept de *free software* (gratuit et libre de droit)

Vue d'ensemble

Historique

Philosophie

Open Source

Python, CPython, ...

Calcul scientifique

Cas d'utilisations de python

Installation

Environnement de développement

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

Open Source – Business Models

Matthieu Falce

Le logiciel étant accessible librement, on peut l'utiliser gratuitement si on le souhaite (dans les conditions de la licence). Comment faire quelque chose de viable économiquement ?

- ▶ support et services (conseils, meilleure connaissance)
- ▶ partenariat publicité (firefox propose google comme navigateur par défaut)
- ▶ offre SaaS / PaaS (elastic cloud par exemple)
- ▶ vente de produits (le logiciel est Open Source mais le prix de vente compense)
- ▶ fonctionnalités payantes en plus (NGINX, MySQL, ...)
- ▶ double licence (Qt, Neo4j)
- ▶ projets open sourcés
- ▶ financement d'une fondation

Vue d'ensemble

Historique

Philosophie

Open Source

Python, CPython, ...

Calcul scientifique

Cas d'utilisations de python

Installation

Environnement de développement

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

Open Source – Intérêt

Matthieu Falce

- ▶ gratuité (dans la plupart des cas)
- ▶ qualité du code, tout le monde peut corriger (discutable en pratique)
- ▶ plus de choix de solutions

Vue d'ensemble

Historique

Philosophie

Open Source

Python, CPython, ...

Calcul scientifique

Cas d'utilisations de python

Installation

Environnement de

développement

Langage Python

Programmation

Orientée objet

(POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

Open Source – Licences

Matthieu Falce

Licences les plus connues³ :

- ▶ Non copyleft
 - ▶ BSD 3-Clause "New" or "Revised" license
 - ▶ BSD 2-Clause "Simplified" or "FreeBSD" license
 - ▶ MIT license
 - ▶ Apache License 2.0
- ▶ Copyleft (la licence se propage à ceux qui l'utilisent)
 - ▶ GNU General Public License (GPL)
 - ▶ GNU Library or "Lesser" General Public License (LGPL)
 - ▶ CeCILL

Il peut y avoir des problèmes avec le droit français, le mieux est d'en parler avec un juriste / avocat compétent⁴

Vue d'ensemble

Historique

Philosophie

Open Source

Python, CPython, ...

Calcul scientifique

Cas d'utilisations de python

Installation

Environnement de

développement

Langage Python

Programmation

Orientée objet

(POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

3.<https://opensource.org/licenses>

4.<https://www.journaldu.net.com/solutions/dsi/1141398-comment-se-reperer-dans-la-jungle-des-licences-open-source/>

Open Source – Exemples

Matthieu Falce

- ▶ Python
- ▶ GNU/Linux
- ▶ WordPress
- ▶ Apache
- ▶ ...

Et plein d'autres :

- ▶ github : 96M projets ⁵
- ▶ Ubuntu 19.10 : 30565⁶ logiciels dans les dépôts
- ▶ AUR : 47331⁷ logiciels dans les dépôts

5.<https://octoverse.github.com/>

6.<https://repology.org/>

7.<https://repology.org/>

Vue d'ensemble

Historique

Philosophie

Open Source

Python, CPython, ...

Calcul scientifique

Cas d'utilisations de python

Installation

Environnement de

développement

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

C'est quoi python au final ?

Matthieu Falce

Vue d'ensemble

Historique

Philosophie

Open Source

Python, CPython, ...

Calcul scientifique

Cas d'utilisations de python

Installation

Environnement de
développement

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

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.

Python c'est lent ?

Matthieu Falce

Oui

- ▶ langage dynamiquement typé
- ▶ interprété
- ▶ pas de gestion fine de la mémoire (garbage collection)
- ▶ probablement moins rapide que le programme équivalent en C/C++/Go/Rust/(Java)/insérer langage préféré⁸

Mais...⁹ ¹⁰

8.<https://benchmarks.game.alioth.debian.org/u64q/compare.php?lang=python3&lang2=gpp>

9.<https://www.quora.com/Why-is-Python-so-popular-despite-being-so-slow>

10.<https://www.quora.com/Why-is-Python-used-for-deep-learning-if-it-is-so-slow>

Python c'est lent ?

Matthieu Falce

En fait, c'est plus subtil :

- ▶ moins d'optimisations inutiles
- ▶ temps de développement réduits
- ▶ langage de haut niveau ⇒ moins de code ⇒ moins de bugs
- ▶ code proche de l'algorithme
- ▶ VM probablement mieux écrite que la solution naïve⁸
- ▶ possibilité d'optimiser les parties critiques

8.<https://stackoverflow.com/questions/9371238/why-is-reading-lines-from-stdin-much-slower-in-c-than-python?rq=1>

Pour les scientifiques¹⁰

- ▶ interface entre langages
- ▶ facilité de développement

Python a été pensé pour servir d'interface entre les langages :⁹

- ▶ manipulation d'autres programmes (à la bash)
- ▶ échange de données binaires directement entre langages

Exemple : lancer un calcul puis l'analyser en utilisant des méthodes en C ou Fortran.

Langage utilisé dans les milieux scientifiques. Quelques raisons :

- ▶ Prototypage
- ▶ Programmation accessible aux experts scientifiques, pas qu'aux développeurs
- ▶ "piles incluses" + écosystème riche

9.<https://www.python.org/doc/essays/omg-darpa-mcc-position/>

10 <https://www.datacamp.com/community/blog/python-scientific-computing-case>

Matthieu Falce

Vue d'ensemble

Historique

Philosophie

Open Source

Python, CPython, ...

Calcul scientifique

Cas d'utilisations de python

Installation

Environnement de développement

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

Interpréteur embarqué dans des logiciels

Matthieu Falce

Vue d'ensemble

Historique

Philosophie

Open Source

Python, CPython, ...

Calcul scientifique

Cas d'utilisations de python

Scripting

Exemples personnels

Installation

Environnement de développement

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

Python sert de langage de script dans de nombreux logiciels :

- ▶ blender¹¹
- ▶ qgis¹²
- ▶ autodesk¹³
- ▶ Vim¹⁴
- ▶ Minecraft¹⁵
- ▶ ...

11.<https://blender.org>

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

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

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

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

Exemples personnels

- ▶ électronique / projets *makers*
 - ▶ Artefact (un jeu d'énigmes tangible) ¹⁶ ¹⁷
 - ▶ *Real Full Stack Python* (du microcontrôleur à la page web en python) ¹⁸
 - ▶ Réalisation de souris / claviers / joysticks / touchpad USB HID
- ▶ Web
 - ▶ EdX ¹⁹ / OpenFUN ²⁰
- ▶ Analyse de données
 - ▶ analyse de séries temporelles
 - ▶ analyse géospatiale

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

17.http://falce.net/presentation/Artefact-LillePy/prez_artefact.slides.html

18.http://falce.net/presentation/IoT_Dashboard/index.html

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

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

Matthieu Falce

Vue d'ensemble

Historique

Philosophie

Open Source

Python, CPython, ...

Calcul scientifique

Cas d'utilisations de python

Scripting

Exemples personnels

Installation

Environnement de développement

Langage Python

Programmation
Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

Distributions ²¹

Matthieu Falce

Vue d'ensemble

Historique

Philosophie

Open Source

Python, CPython, ...

Calcul scientifique

Cas d'utilisations de python

Installation

Environnement de développement

Langage Python

Programmation
Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

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 l'installation officielle. Pour les data scientists possiblement anaconda.

21.<https://wiki.python.org/moin/PythonDistributions>

Editeurs |

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

- ▶ éditeurs de texte + extensions
 - ▶ Microsoft Studio Code
 - ▶ ViM / Emacs + plugins
- ▶ IDE
 - ▶ eclipse + mode python
 - ▶ PyCharm
- ▶ datascience
 - ▶ jupyter notebook
 - ▶ jupyter lab

Les IDE / éditeurs avancés permettent d'intégrer / faciliter une bonne partie des bonnes pratiques que nous verrons tout au long du cours.

Matthieu Falce

Vue d'ensemble

Historique
Philosophie
Open Source
Python, CPython, ...
Calcul scientifique
Cas d'utilisations de python
Installation

Environnement de développement

Langage Python

Programmation
Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

Matthieu Falce

Vue d'ensemble

Langage Python

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

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

Langage Python

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")
```

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

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

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Introspection

Bibliographie

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

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

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Introspection

Bibliographie

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

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

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

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

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

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

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Calculs

Matthieu Falce

- ▶ 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

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Chaînes

Matthieu Falce

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

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
```

Chaînes – performances

Matthieu Falce

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

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))
```

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:\\\\F00\\\\Bar\\\\Baz\\ ")
```

```
# Les chaines en Python 3 sont unicodes  
print("éàùù")
```

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Conteneurs

Matthieu Falce

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

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Conteneurs

Matthieu Falce

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

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

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 dépassements 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

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Conteneurs

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)
```

Matthieu Falce

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Chaînes comme conteneurs

```
print("Transformer un iterable en chaîne :")
elements = (1, 2, 3)
print("-".join([str(i) for i in elements]))

print("Transformer une chaîne en itérable :")
chaîne = "Il était \n une fois"
print(chaîne.split("\n"))

print("Les chaînes sont des conteneurs que l'on peut slicer :")
ma_chaîne = "Il était une fois"
print(ma_chaîne[5:10])
```

Matthieu Falce

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

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

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Passage par référence

Matthieu Falce

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique



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.

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

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

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

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

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

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Construction des conteneurs

Matthieu Falce

Structure mémoire d'une liste

PyListObject

type	list
refcount	1
value	
...	...

PyObject	
value	1
refcount	1
type	int
...	...

PyObject	
value	"a"
refcount	1
type	str
...	...

PyObject	
value	2
refcount	1
type	int
...	...

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

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

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Cycle de vie

Matthieu Falce

```
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
```

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

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
```

Structures de données

Matthieu Falce

Python permet de faire beaucoup avec les structures de données de sa bibliothèque standard.

- ▶ list
- ▶ set
- ▶ dict
- ▶ tuple

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Piles et files

Matthieu Falce

- ▶ comment implémenter une pile (*stack*)
 - ▶ list.pop
 - ▶ list.append
- ▶ comment implémenter un file (*queue*)
 - ▶ list.pop(0)
 - ▶ list.append
 - ▶ ou bien utiliser collection.dequeue

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Arbres

Les arbres peuvent se construire (entre autres) avec des dictionnaires et des listes

```
noise_ontology = {  
    "Mammalia": {  
        "Carnivora": {  
            "Canidae": {  
                "Canis": {  
                    "dog": "waf"  
                }  
            },  
            "Felidae": {  
                "Felis": {  
                    "cat": "miaou"  
                }  
            }  
        }  
    }  
}  
  
print(taxonomy['Mammalia']['Carnivora']['Felidae']['Felis']['cat'])
```

Matthieu Falce

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Arbres

Les arbres peuvent se construire (entre autres) avec des dictionnaires et des listes

```
# source https://gist.github.com/hrldcpr/2012250

import pprint
from collections import defaultdict

def tree():
    return defaultdict(tree)

def tree_to_dicts(t):
    return {k: tree_to_dicts(t[k]) if isinstance(t[k], defaultdict) else t[k]
            for k in t}

# exemple d'utilisation
taxonomy = tree()
taxonomy['Chordata']['Mammalia']['Carnivora']['Felidae']['Felis']['cat'] = "miaou"
taxonomy['Chordata']['Mammalia']['Carnivora']['Canidae']['Canis']['dog'] = "waf"

pprint.pprint(tree_to_dicts(taxonomy))
# {'Chordata': {'Mammalia': {'Carnivora': {'Felidae': {'Felis': {'cat': 'miaou'}}, 'Canidae': {'Canis': {'dog': 'waf'}}}}}}
```

Matthieu Falce

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Introspection

Bibliographie

Programmation

Orientée objet

(POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

Le duck typing ?

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

Matthieu Falce

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Introspection

Bibliographie

Programmation

Orientée objet

(POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

Le *duck typing* ?

Matthieu Falce

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

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>

Le *duck typing* ?

Matthieu Falce

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

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

Exemple

```
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"))))
```

Matthieu Falce

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Les *able

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
- ▶ ...

Matthieu Falce

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

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])

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

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Slicing

Matthieu Falce

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:]								

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]								

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
Structures de données
Duck typing
Slicing
Gestion des fichiers
Encodage des caractères
Contrôle de flux
Fonctions
Exceptions
Introspection
Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

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
Structures de données
Duck typing
Slicing
Gestion des fichiers
Encodage des caractères
Contrôle de flux
Fonctions
Exceptions
Introspection
Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

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

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Context Manager – gestionnaire de contexte

Matthieu Falce

```
# plutôt 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

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Encodage des caractères

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 »
# ...
```

Matthieu Falce

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Introspection

Bibliographie

Programmation

Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Boucles

```
# 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
```

Matthieu Falce

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Introspection

Bibliographie

Programmation

Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Boucles – contrôles

Matthieu Falce

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

On peut contrôler une boucle avec :

- ▶ **break** : sortir de la boucle
- ▶ **continue** : passer à l'élément suivant

Itération

Matthieu Falce

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

```
for number in [1, 2, 3]:  
    if number == 4:  
        print("trouvé !")  
        break  
    else:  
        print("pas trouvé :(")  
  
while number < 0:  
    number -= 1  
    if number == 4:  
        print("trouvé !")  
        break  
    else:  
        print("pas trouvé :(")
```

Boucles – “pythonique et non pythonique”

Matthieu Falce

[Vue d'ensemble](#)

[Langage Python](#)

Syntaxe

Types standards

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Introspection

Bibliographie

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

[Bonnes pratiques](#)

[Bibliothèque standard](#)

[Interface graphiques](#)

[Code natif](#)

[Python scientifique](#)



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])
```

Tuple unpacking

Matthieu Falce

[Vue d'ensemble](#)

[Langage Python](#)

Syntaxe

Types standards

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Introspection

Bibliographie

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

[Bonnes pratiques](#)

[Bibliothèque standard](#)

[Interface graphiques](#)

[Code natif](#)

[Python scientifique](#)

On peut déconstruire des tuples à la volée.

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

* en compréhension

Matthieu Falce

On peut construire / manipuler des itérables à la volée

On appelle ça les listes en compréhension ('list comprehension') ou 'dictionary 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)}
```

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Introspection

Bibliographie

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

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

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Introspection

Bibliographie

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

Tests et conditions – booléens

Matthieu Falce

[Vue d'ensemble](#)

[Langage Python](#)

Syntaxe

Types standards

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Introspection

Bibliographie

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

[Bonnes pratiques](#)

[Bibliothèque standard](#)

[Interface graphiques](#)

[Code natif](#)

[Python scientifique](#)

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
```

Paresse et générateurs

Matthieu Falce

[Vue d'ensemble](#)

[Langage Python](#)

Syntaxe

Types standards

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Introspection

Bibliographie

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

[Bonnes pratiques](#)

[Bibliothèque standard](#)

[Interface graphiques](#)

[Code natif](#)

[Python scientifique](#)

```
# 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
```

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

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Gestion des arguments

Gotchas

Higher order functions

Closures

Décorateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Déclaration d'une fonction

Matthieu Falce

```
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)
```

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Gestion des arguments

Gotchas

Higher order functions

Closures

Décorateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

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

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Gestion des arguments

Gotchas

Higher order functions

Closures

Décorateurs

Exceptions

Introspection

Bibliographie

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Arguments des fonctions

Matthieu Falce

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))
```

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Gestion des arguments

Gotchas

Higher order functions

Closures

Décorateurs

Exceptions

Introspection

Bibliographie

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Arguments des fonctions

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)
```

Matthieu Falce

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Gestion des arguments

Gotchas

Higher order functions

Closures

Décorateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Arguments des fonctions

Matthieu Falce

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Gestion des arguments

Gotchas

Higher order functions

Closures

Décorateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

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

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Gestion des arguments

Gotchas

Higher order functions

Closures

Décorateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

```
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

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Gestion des arguments

Gotchas

Higher order functions

Closures

Décorateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

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 })
```

Liens avec le unpacking

Matthieu Falce

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Gestion des arguments

Gotchas

Higher order functions

Closures

Décorateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

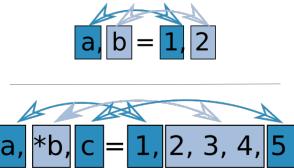
Bonnes pratiques

Bibliothèque
standard

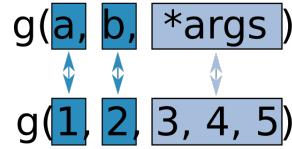
Interface
graphiques

Unpacking

Pour les variables



Pour les arguments



Arguments des fonctions – résumé

Matthieu Falce

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Gestion des arguments

Gotchas

Higher order functions

Closures

Décorateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

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

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

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Gestion des arguments

Gotchas

Higher order functions

Closures

Décorateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Problèmes classiques – éléments mutables²² ²³

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

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Gestion des arguments

Gotchas

Higher order functions

Closures

Décorateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Problèmes classiques – portée des variables

```
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()
```

Matthieu Falce

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Gestion des arguments

Gotchas

Higher order functions

Closures

Décorateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Problèmes classiques – portée des variables

Matthieu Falce

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Gestion des arguments

Gotchas

Higher order functions

Closures

Décorateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

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

```
# 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

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Gestion des arguments

Gotchas

Higher order functions

Closures

Décorateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

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

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Gestion des arguments

Gotchas

Higher order functions

Closures

Décorateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Closures / Fermeture

Matthieu Falce

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Gestion des arguments

Gotchas

Higher order functions

Closures

Décorateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

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

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Gestion des arguments

Gotchas

Higher order functions

Closures

Décorateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

```
# 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"
```

Closures – Exemples

```
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
```

Matthieu Falce

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Gestion des arguments

Gotchas

Higher order functions

Closures

Décorateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Décorateurs – Syntaxe

Matthieu Falce

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Gestion des arguments

Gotchas

Higher order functions

Closures

Décorateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

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

Décorateurs – Syntaxe

Matthieu Falce

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Gestion des arguments

Gotchas

Higher order functions

Closures

Décorateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

```
@decorateur
def fonction():
    pass

fonction = decorateur(fonction)
```

Décorateurs – Syntaxe

Matthieu Falce

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Gestion des arguments

Gotchas

Higher order functions

Closures

Décorateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

```
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())
```

Décorateurs – Syntaxe

Matthieu Falce

```
# comment accepter des paramètres

def ecrit_avant_apres(fonction_a_decorcer):
    """Cette fonction prend une fonction qu'elle va
    décorer.
"""

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

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

@ecrit_avant_apres
def test_deco_syntaxe(a, b, c=0):
    return "résultat 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

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Gestion des arguments

Gotchas

Higher order functions

Closures

Décorateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Décorateurs paramétrés

Matthieu Falce

```
# comment passer des paramètres au décorateur

def ecrit_avant_apres_plein_de_fois(nb_avant, nb_apres):
    # on rajoute un niveau, une fabrique de décorateur
    # permet d'avoir les paramètres par closure
    def ecrit_avant_apres(fonction_a_decorcer):
        def wrapper(*args, **kwargs):
            print("avant " * nb_avant)
            res = fonction_a_decorcer(*args, **kwargs)
            print("après " * nb_apres)
            return res
        return wrapper
    return ecrit_avant_apres

@ecrit_avant_apres_plein_de_fois(nb_avant=2, nb_apres=4)
def f(a, b):
    print("dans f ", a, b)

f(1, 3)
# avant avant
# dans f 1 3
# après après après après
```

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Gestion des arguments

Gotchas

Higher order functions

Closures

Décorateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Décorateurs paramétrés

Matthieu Falce

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Gestion des arguments

Gotchas

Higher order functions

Closures

Décorateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

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

```
@decorateur(a, b)
def fonction():
    pass

fonction = decorateur(a, b)(fonction)
```

Introspection ?

Matthieu Falce

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Gestion des arguments

Gotchas

Higher order functions

Closures

Décorateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

```
def inutile(fonction_a_decorcer):
    """Décorateur inutile."""

    def wrapper(*args, **kwargs):
        """Fonction wrapper."""
        return fonction_a_decorcer(*args, **kwargs)

    return wrapper

@inutile
def f(a):
    """Une super fonction !"""
    return "dans f"

help(f)
# Help on function wrapper in module __main__:
# wrapper(*args, **kwargs)
#     Fonction wrapper.
```

Introspection ?

Matthieu Falce

```
from functools import wraps

def inutile(fonction_a_decorer):
    """Décorateur inutile."""

    @wraps(fonction_a_decorrer) # on indique que wrapper entoure une fonction
    def wrapper(*args, **kwargs):
        """Fonction wrapper."""
        return fonction_a_decorrer(*args, **kwargs)

    return wrapper

@inutile
def f(a):
    """Une super fonction !"""
    return "dans f"

help(f)
# Help on function f in module __main__:
# f(a)
#     Une super fonction !
```

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Gestion des arguments

Gotchas

Higher order functions

Closures

Décorateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Introspection ?

Matthieu Falce

implémentée à travers des variables magiques

- ▶ `__qualname__` : nom qualifié (chemin depuis le module)²⁴
- ▶ `__name__` : nom de la fonction (pas de la variable qui la contient)
- ▶ `__doc__` : docstring de la fonction

Comment fonctionne `functools.wraps` d'après vous ?

24.<https://docs.python.org/3/glossary.html#term-qualified-name>

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

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Gestion des arguments

Gotchas

Higher order functions

Closures

Décorateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Exceptions

Matthieu Falce

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

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

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Introspection

Bibliographie

Programmation

Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

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

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

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

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Introspection

Bibliographie

Programmation

Orientée objet

(POO)

Bonnes pratiques

Bibliothèque standard

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

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Introspection

Bibliographie

Programmation

Orientée objet

(POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

Introspection ?

Matthieu Falce

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Introspection

Bibliographie

Programmation

Orientée objet

(POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

In everyday life, introspection is the act of self-examination. Introspection refers to the examination of one's own thoughts, feelings, motivations, and actions. The great philosopher Socrates spent much of his life in self-examination, encouraging his fellow Athenians to do the same. He even claimed that, for him, "the unexamined life is not worth living."

<https://www.ibm.com/developerworks/library/l-pyint/index.html>

Introspection ?

Matthieu Falce

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Introspection

Bibliographie

Programmation

Orientée objet

(POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

In computer programming, introspection is the ability to determine the type of an object at runtime. It is one of Python's strengths. Everything in Python is an object and we can examine those objects. Python ships with a few built-in functions and modules to help us.

http://book.pythontips.com/en/latest/object_introspection.html

Comment faire ?

Matthieu Falce

- ▶ `sys` : informations sur l'interpréteur
- ▶ `dir` : méthodes / fonctions d'une classe / module
- ▶ `id` : zone mémoire d'un objet
- ▶ `type` : type d'un objet
- ▶ `hasattr` / `getattr` : modification d'une instance
- ▶ `isinstance` / `issubclass` : savoir si un objet est d'un certain type
- ▶ méthode magique (`__name__`) : quel est le nom d'un objet
- ▶ ...

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Comment faire ?

Matthieu Falce

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Module `inspect`²⁵ de la bibliothèque standard

- ▶ avoir des informations sur le code source (fichier / module / ligne / ...)
- ▶ inspecter les signatures des *callables*
- ▶ analyser les classes et fonctions
- ▶ déterminer l'état de l'interpréteur (*function stack*, ...)
- ▶ ...

25. <https://docs.python.org/3/library/inspect.html>

A quoi ça sert ?

Matthieu Falce

- ▶ analyse des exceptions / *stacktraces*
- ▶ code dépendant du type d'un objet
- ▶ familiarisation avec un nouveau code (autocomplete dans le shell / analyse des attributs en direct...)
- ▶ ...

Vue d'ensemble

Langage Python

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

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

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/>

▶ Types de données :

- ▶ <https://docs.python.org/fr/3/tutorial/datastructures.html>
- ▶ http://python-prepa.github.io/information_theory.html

Vue d'ensemble

Langage Python

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

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Bibliographie II

- ▶ <https://deptinfo-ensip.univ-poitiers.fr/ENS/doku/doku.php/stu:algo2>
- ▶ <https://www.apprendre-en-ligne.net/info/structures/structures.pdf>
- ▶ Instrospection :
 - ▶ http://book.pythontips.com/en/latest/object_introspection.html
 - ▶ <https://www.ibm.com/developerworks/library/l-pyint/index.html>
 - ▶ https://python.developpez.com/cours/DiveIntoPython/php/frdriveintopython/power_of_introspection/index.php
 - ▶ <https://docs.python.org/3/library/inspect.html>
 - ▶ <http://sametmax.com/comprendre-les-decorateur-python-pas-a-pas-partie-2/>
- ▶ Variables :

Matthieu Falce

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Instrospection

Bibliographie

Programmation

Orientée objet

(POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

Bibliographie III

- ▶ <http://sametmax.com/valeurs-et-references-en-python/>
- ▶ <http://sametmax.com/id-none-et-bidouilleries-memoire-en-python/>
- ▶ <https://medium.com/@tyastropheus/tricky-python-i-memory-management-for-mutable-immutable-objects-21507d1e5b95>
- ▶ Clause else dans les itérations :
 - ▶ <https://stackoverflow.com/questions/3295938/else-clause-on-python-while-statement>
 - ▶ https://docs.python.org/2/reference/compound_stmts.html#the-while-statement
- ▶ Exceptions :
 - ▶ <http://sametmax.com/gestion-des-erreurs-en-python/>
 - ▶ <http://sametmax.com/comment-recruter-un-developpeur-python/>

Matthieu Falce

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Instrospection

Bibliographie

Programmation

Orientée objet

(POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

Bibliographie IV

- ▶ <http://sametmax.com/pourquoi-utiliser-un-mecanisme-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>
- ▶ *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
 - ▶ <http://sametmax.com/quest-ce-que-le-duck-typing-et-a-quoi-ca-sert/>

Matthieu Falce

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Introspection

Bibliographie

Programmation

Orientée objet

(POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

Bibliographie V

Matthieu Falce

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Introspection

Bibliographie

Programmation

Orientée objet

(POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Design Patterns

Bibliographie

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Programmation Orientée objet (POO)

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

Design Patterns

Bibliographie

Bonnes pratiques

Bibliothèque
standard

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

Vocabulaire

Matthieu Falce

- ▶ une classe définit 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

Design Patterns

Bibliographie

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Association entre classes

Matthieu Falce

2 grandes techniques pour associer des classes entre elles :

- ▶ *héritage* (*inheritance* 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*
- ▶ *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

Design Patterns

Bibliographie

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

UML

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Design Patterns

Bibliographie

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

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))

UML

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Design Patterns

Bibliographie

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

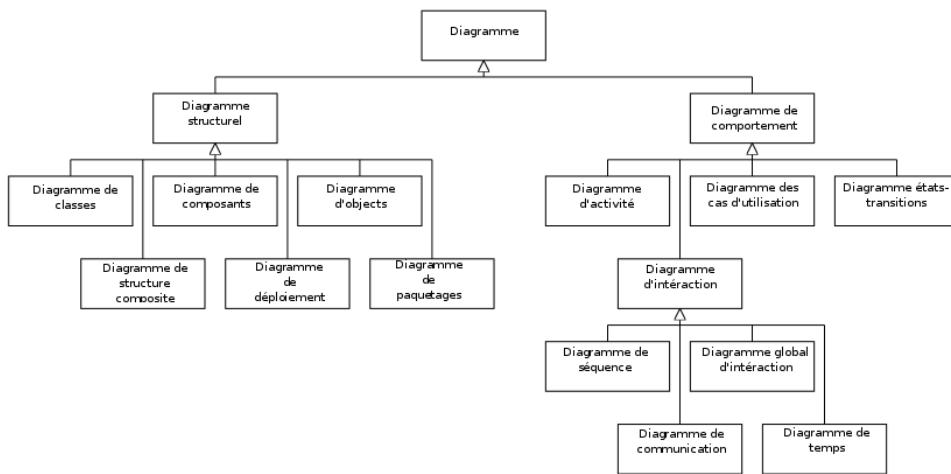
Python scientifique

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
- ▶ ...

UML

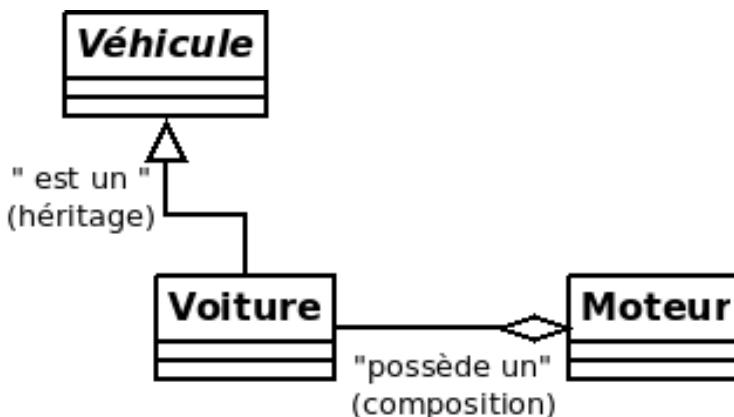
Diagramme montrant la hiérarchie de types de diagrammes UML



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)

Diagrammes de classe

Diagramme de classes montrant composition et héritage



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

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Design Patterns

Bibliographie

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Design Patterns

Bibliographie

Bonnes pratiques

Bibliothèque standard

Interface graphiques

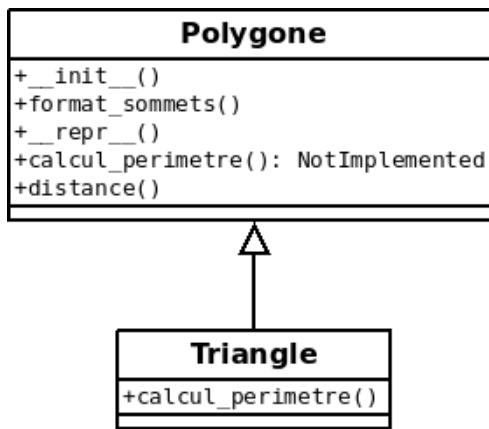
Code natif

Python scientifique

Héritage

Matthieu Falce

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

Design Patterns

Bibliographie

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

Créer 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

Design Patterns

Bibliographie

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

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

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

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

Design Patterns

Bibliographie

Bonnes pratiques

Bibliothèque
standard

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

Design Patterns

Bibliographie

Bonnes pratiques

Bibliothèque
standard

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

Design Patterns

Bibliographie

Bonnes pratiques

Bibliothèque standard

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

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Design Patterns

Bibliographie

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

```
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
```

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

Design Patterns

Bibliographie

Bonnes pratiques

Bibliothèque
standard

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

Design Patterns

Bibliographie

Bonnes pratiques

Bibliothèque
standard

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

Design Patterns

Bibliographie

Bonnes pratiques

Bibliothèque
standard

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

Capturer une exception

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Design Patterns

Bibliographie

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

```
# 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)
```

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
Design Patterns
Bibliographie
Bonnes pratiques
Bibliothèque standard
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
Design Patterns
Bibliographie
Bonnes pratiques
Bibliothèque standard
Interface graphiques
Code natif
Python scientifique

Quand utiliser une classe ?

```
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"))
```

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Design Patterns

Bibliographie

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Quand utiliser 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

Design Patterns

Bibliographie

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

► 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

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

Design Patterns

Bibliographie

Bonnes pratiques

Bibliothèque
standard

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

Design Patterns

Bibliographie

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

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

Design Patterns

Bibliographie

Bonnes pratiques

Bibliothèque standard

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

Design Patterns

Bibliographie

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

Différents types de méthodes

```
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)
```

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Design Patterns

Bibliographie

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

Différents types de méthodes

```
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
```

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Design Patterns

Bibliographie

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

Résumé

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())
```

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Design Patterns

Bibliographie

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Patrons de conception ?

Introduits par Gamma, Helm, Johnson et Vlissides (le Gang of Four) en 1994 par le livre *Design Patterns: Elements of Reusable Software*

En informatique, et plus particulièrement en développement logiciel, un patron de conception (souvent appelé design pattern) est un arrangement caractéristique de modules, reconnu comme bonne pratique en réponse à un problème de conception d'un logiciel. Il décrit une solution standard, utilisable dans la conception de différents logiciels.

https://fr.wikipedia.org/wiki/Patron_de_conception

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Design Patterns

Bibliographie

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Patrons de conception ?

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Design Patterns

Bibliographie

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

« Chaque patron décrit un problème qui se manifeste constamment dans notre environnement, et donc décrit le cœur de la solution à ce problème, d'une façon telle que l'on puisse réutiliser cette solution des millions de fois, sans jamais le faire deux fois de la même manière »

Christopher Alexander, 1977.

Patrons de conception ?

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Design Patterns

Bibliographie

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

3 familles de patrons d'après le GoF

- ▶ *créateurs* : ils définissent comment faire linstanciation et la configuration des classes et des objets ;
- ▶ *structuraux* : ils définissent comment organiser les classes d'un programme dans une structure plus large (séparant linterface de limplémentation) ;
- ▶ *comportementaux* : ils définissent comment organiser les objets pour que ceux-ci collaborent (distribution des responsabilités) et expliquent le fonctionnement des algorithmes impliqués ;

Patrons de conception ?

Matthieu Falce

Quelques exemples :

- ▶ factory
- ▶ adapter
- ▶ chain of responsibility
- ▶ decorator
- ▶ facade
- ▶ iterator
- ▶ observer
- ▶ strategy
- ▶ Model View Controller (MVC)

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Design Patterns

Bibliographie

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

Patrons de conception ?

Matthieu Falce

2 principes généraux :

- ▶ Tenir compte de l'interface et pas de l'implémentation.
- ▶ Préférer la composition à l'héritage.

```
class ConteneurComposition():
    def __init__(self):
        self._conteneur = []

    def append(self, valeur):
        print("avant append")
        self._conteneur.append(valeur)
        print("après append")

class ConteneurHeritage(list):
    def append(self, valeur):
        print("avant append")
        super().append(valeur)
        print("après append")

cc = ConteneurComposition()
ch = ConteneurHeritage()

# les 2 objets ont la même interface mais pas le même type (duck typing)
cc.append(1)
ch.append(2)
```

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Design Patterns

Bibliographie

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

Comportementaux – Iterator

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Design Patterns

Bibliographie

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

Inclus de base dans le langage

Comportementaux – Chain of responsibility

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Design Patterns

Bibliographie

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

Comment assurer une série de traitement sur des objets ?

```
class ContentFilter(object):
    def __init__(self, filters=None):
        self._filters = filters or []

    def filter(self, content):
        for filter in self._filters:
            content = filter(content)
        return content

content_filter = ContentFilter(
    [
        lambda c: (e for e in c if e % 2 == 0),
        lambda c: (e for e in c if str(e) == str(e)[::-1]),
    ]
)
content = range(1000)
filtered_content = content_filter.filter(content)
print(list(filtered_content)[10:20])
```

Comportementaux – Observer

Comment distribuer des notifications à plusieurs objets qui doivent être avertis d'une notification ?

```
class Observer():
    observers = []
    def __init__(self):
        self.observers.append(self)
        self._observables = {}
    def observe(self, event_name, callback):
        self._observables[event_name] = callback

class Event():
    def __init__(self, name, data):
        self.name = name
        self.data = data
    def fire(self):
        for observer in Observer.observers:
            if self.name in observer._observables:
                observer._observables[self.name](self.data)

class Salle(Observer):
    def vient_d_arriver(self, who):
        print("nouvel événement : ", who, "est arrivé !")

salle = Salle()
salle.observe('arrive', salle.vient_d_arriver)
Event('arrive', 'Matthieu').fire()
Event('part', 'Matthieu').fire()
```

Matthieu Falce

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Concepts
Association
Modélisation
POO en python
Gestion des exceptions
Classe ou pas ?
Méthodes
Design Patterns
Bibliographie
Bonnes pratiques
Bibliothèque standard
Interface graphiques
Code natif
Python scientifique

Createur – Singleton

Comment s'assurer qu'une seule instance d'une classe ne peut exister à un moment donné et fournir un point d'accès vers cette instance ?

```
class Singleton(object):
    _instances = {}

    def __new__(cls, *args, **kw):
        if not cls in cls._instances:
            instance = super().__new__(cls)
            cls._instances[cls] = instance
        return cls._instances[cls]

class Logger(Singleton):
    pass

class Logger2(Singleton):
    pass

l1 = Logger()
l1_bis = Logger()
print(l1 is l1_bis)

l2 = Logger2()
print(l1 is l2)
```

Matthieu Falce

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Concepts
Association
Modélisation
POO en python
Gestion des exceptions
Classe ou pas ?
Méthodes
Design Patterns
Bibliographie
Bonnes pratiques
Bibliothèque standard
Interface graphiques
Code natif
Python scientifique

Createur – Singleton

Matthieu Falce

Comment s'assurer qu'une seule instance d'une classe ne peut exister à un moment donné et fournir un point d'accès vers cette instance ?

On peut le faire autrement (à la main) :

- ▶ définir dans un module
- ▶ définir dans un fichier de conf chargé une seule fois
- ▶ passer l'instance à chaque objet

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Design Patterns

Bibliographie

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

Structural – Décorateur

Matthieu Falce

Comment ajouter de nouvelles fonctions à un objet dynamiquement lors de l'exécution ?

Inclus de base dans le langage (les fonctions sont des *first class citizen*)

Pas forcément !

Les décos avec `@` sont statiques et pas dynamiques.
Mais même concept d'enveloppement.

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Design Patterns

Bibliographie

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

Structural – Adapter

Matthieu Falce

Comment déguiser une classe en une autre ?

```
from datetime import datetime

def log(message, destination):
    destination.write("{} - {}".format(datetime.now(), message))

class ConsoleDestination:
    def write(self, message):
        print(message)

# le duck typing facilite ce pattern car on n'a pas
# besoin d'avoir le bon type, juste la bonne interface
log("dans un fichier", open("/tmp/log.log", "w"))
log("dans la console", ConsoleDestination())
```

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Design Patterns

Bibliographie

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Conclusion

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Design Patterns

Bibliographie

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

- ▶ faciles à mettre en place en python avec le *duck typing*
- ▶ permettent d'exprimer et de formaliser une approche
- ▶ permettent de structurer des projets en ayant des abstractions connues (changement des équipes, longs développements)
- ▶ permettent de prévoir de bonnes API à ces classes
- ▶ ne pas chercher à en abuser

Bibliographie I

- ▶ 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/>

Matthieu Falce

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Concepts
Association
Modélisation
POO en python
Gestion des exceptions
Classe ou pas ?
Méthodes
Design Patterns
Bibliographie
Bonnes pratiques
Bibliothèque standard
Interface graphiques
Code natif
Python scientifique

Bibliographie II

- ▶ Design patterns :
 - ▶ https://github.com/ActiveState/code/blob/master/recipes/Python/102187_Singleton_as_a_metaclass/recipe-102187.py
 - ▶ <https://github.com/faif/python-patterns>
 - ▶ <https://www.toptal.com/python/python-design-patterns>
 - ▶ <https://www.youtube.com/watch?v=0vJJlVBVTFg>
 - ▶ <http://sametmax.com/objets-proxy-et-pattern-adapter-en-python/>
 - ▶ <http://www.e-naxos.com/Blog/post/Design-Patterns-ou-quand-comment-et-pourquoi-.aspx>
 - ▶ <http://sdz.tdct.org/sdz/le-pattern-decorator-en-python.html>
- ▶ Loi de Demeter :
 - ▶ [https://www2.ccs.neu.edu/research/demeter/demeter-method/LawOfDemeter/paper-boy/demeter.pdf](http://www2.ccs.neu.edu/research/demeter/demeter-method/LawOfDemeter/paper-boy/demeter.pdf)

Matthieu Falce

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Concepts
Association
Modélisation
POO en python
Gestion des exceptions
Classe ou pas ?
Méthodes
Design Patterns
Bibliographie
Bonnes pratiques
Bibliothèque standard
Interface graphiques
Code natif
Python scientifique

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

Interface
graphiques

Code natif

Python scientifique

Bonnes pratiques

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

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 ?

- ▶ 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*)
- ▶ on peut revenir à une version antérieure du projet / savoir qui a fait quoi / quand
 - ▶ utiliser un système de contrôle de version (Git, ...)

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

Interface
graphiques

Code natif

Python scientifique

Avant Propos

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

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

Interface
graphiques

Code natif

Python scientifique

Écosystème

- ▶ Environnement isolé / installation de paquets :
 - ▶ `virtualenv` (+ wrappers comme `pew` ou `virtualenvwrapper`)
 - ▶ `pip`
 - ▶ `pipenv`
 - ▶ `conda`
 - ▶ `easy_install`
 - ▶ `poetry`
- ▶ PyPI
- ▶ wheels
- ▶ eggs
- ▶ ...

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

Interface
graphiques

Code natif

Python scientifique

Comment ça marche !

En pratique, vous voulez :

- ▶ avoir un environnement virtuel pour chaque projet sur lequel vous travaillez
- ▶ avoir la liste des paquets à installer et leurs versions pour les répliquer facilement

Certains IDE (comme pycharm) créent automatiquement un environnement virtuel à chaque nouveau projet.

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

Interface
graphiques

Code natif

Python scientifique

Comment ça marche II

Pour cela, vous pouvez utiliser les outils que nous avons vu :

- ▶ **virtualenv** avec **pip**, le plus simple, inclus dans la distribution standard
- ▶ **poetry** qui gère
 - ▶ l'environnement
 - ▶ les dépendances (primaires et secondaires)
 - ▶ toutes les facettes de votre projet
- ▶ **conda** qui gère
 - ▶ la version de python
 - ▶ l'environnement
 - ▶ les dépendances python déjà compilées (stockées sur leur forge)
 - ▶ mais aussi des logiciels entiers (pas forcément en python)

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

Interface
graphiques

Code natif

Python scientifique

Comment ça marche III

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

Interface
graphiques

Code natif

Python scientifique

Le choix est une question de gouts.

- ▶ personnellement pip et virtualenv m'ont toujours suffit
- ▶ dans la communauté scientifique, conda est préféré, car dédié aux gens peu technique (frontend graphique de l'installateur / gestionnaire d'environnements), installations de logiciels compilés facilement...
- ▶ les adeptes des nouveautés préfèrent poetry

Installer

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

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

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

Interface
graphiques

Code natif

Python scientifique

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]"
```

Commandes classiques

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

Interface
graphiques

Code natif

Python scientifique

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]"
```

Commandes classiques

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

Interface
graphiques

Code natif

Python scientifique

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
```

Commandes classiques

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

Interface
graphiques

Code natif

Python scientifique

Fichier requirements.txt

freeze des dépendances

```
pip freeze > requirements.txt
```

installer depuis un fichier de requirements

```
pip install -r requirements.txt
```

Autres commandes

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

Interface
graphiques

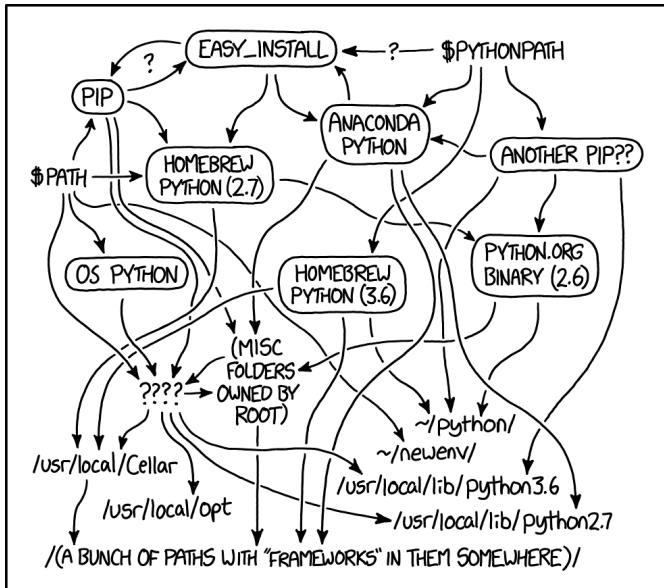
Code natif

Python scientifique

- ▶ 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)

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/>

Matthieu Falce

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

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

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

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

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

Interface
graphiques

Code natif

Python scientifique

virtualenv

Matthieu Falce

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

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

Interface
graphiques

Code natif

Python scientifique

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

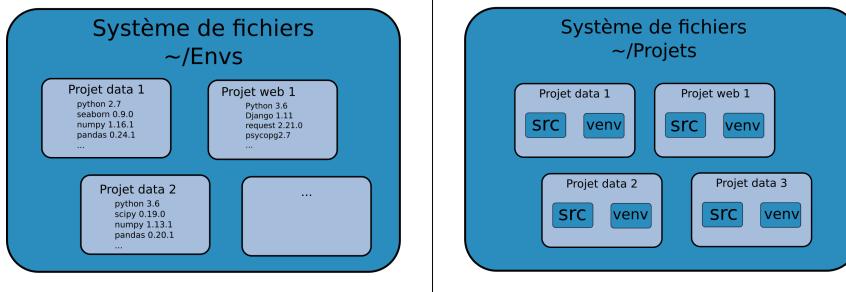
Documentation

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique



Organisation des environnements virtuels

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

Interface
graphiques

Code natif

Python scientifique

- ▶ 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

Outils de débogage

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)
```

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

Interface
graphiques

Code natif

Python scientifique

Qualité du code – pep8 / linters

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
- ▶ ...

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

Interface
graphiques

Code natif

Python scientifique

Qualité du code – pep8 / linters

Matthieu Falce

Il existe des “linters” pour vous assister dans l’écriture.

Ils peuvent lister les erreurs, variables non déclarées, typos, mauvais import...

Ils s’exécutent sans exécuter le code (on parle d’analyse statique)

- ▶ flake8 / pylint
- ▶ mypy / pyright
- ▶ ...

Chacun a ses spécificités (vérification des types, des erreurs de syntaxe...).

Ils peuvent s’intégrer 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

Interface graphiques

Code natif

Python scientifique

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).

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

Ils peuvent s’intégrer 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

Interface graphiques

Code natif

Python scientifique

En pratique

- ▶ faire attention en cas de projet long²⁶ / collaboratif (utiliser les mêmes outils, en même temps) en cas d'utilisation d'un formateur automatique
- ▶ outils
 - ▶ black
 - ▶ isort (mise au propre des imports)
 - ▶ mypy / pylint
- ▶ les intégrer dans des outils (par exemple à chaque sauvegarde d'un fichier)
- ▶ on peut les intégrer dans des pre-commits hook / un mécanisme d'intégration continue

26.https://black.readthedocs.io/en/stable/guides/introducing_black_to_your_project.html

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

Interface
graphiques

Code natif

Python scientifique

Timing et profilage

```
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 : {}s".format(time.time() - tic))

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

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

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

Interface
graphiques

Code natif

Python scientifique

Timing et profilage

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

Interface
graphiques

Code natif

Python scientifique

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}
```

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

Nous allons utiliser la bibliothèque unittest²⁷

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

Tests unitaires – tests verts

```
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()
```

Résultat :

```
python test_unittest.py
```

```
....
```

```
-----
```

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

```
OK
```

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

Interface
graphiques

Code natif

Python scientifique

Tests unitaires – tests rouges

```
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()
```

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
```

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

Interface
graphiques

Code natif

Python scientifique

Tests unitaires – fixtures

```
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()
```

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

Interface
graphiques

Code natif

Python scientifique

Tests unitaires – fixtures

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
```

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

Interface
graphiques

Code natif

Python scientifique

Aller plus loin

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.

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

Interface
graphiques

Code natif

Python scientifique

Aller plus loin

Il existe d'autres modules pour lancer les tests ('testrunners')
28.

- ▶ (doctest²⁹)
- ▶ nose³⁰
- ▶ pytest (allège la syntaxe des tests)³¹

Les tests sont souvent utilisés avec des 'mocks'³² pour modifier le comportement des modules externes.

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

Interface
graphiques

Code natif

Python scientifique

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

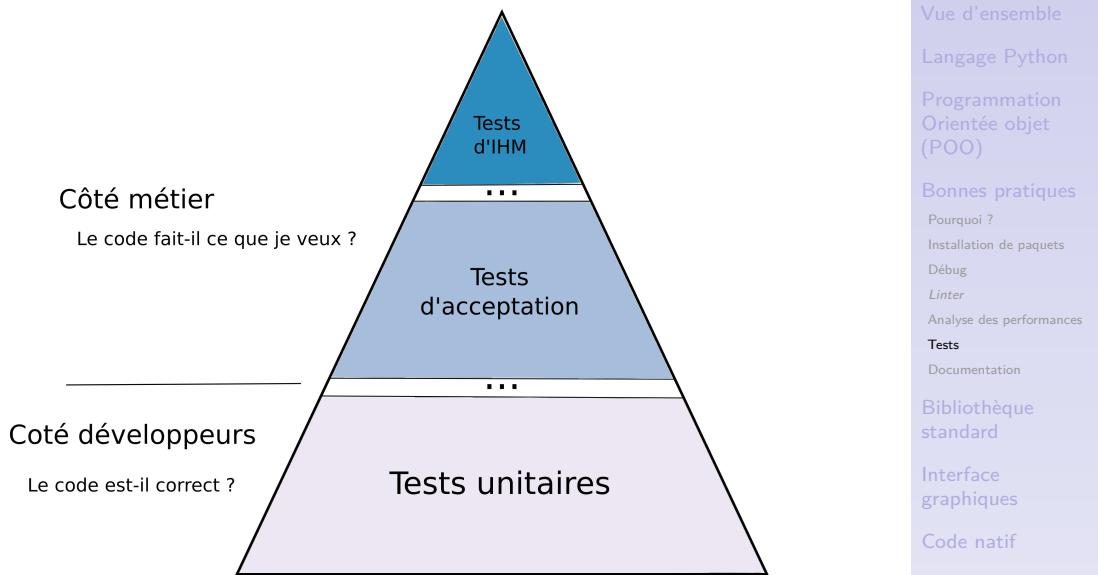
29.<https://docs.python.org/3.6/library/doctest.html>

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

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

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

Aller plus loin – autres types de tests



Inspiration :
<https://www.slideshare.net/RajIndugula/agile-testing-practices-38015016>

Documentation ?

- ▶ 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
```

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

Interface
graphiques

Code natif

Python scientifique

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

Interface
graphiques

Code natif

Python scientifique

Documentation ?

- ▶ 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__)
```

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

Interface
graphiques

Code natif

Python scientifique

Comment écrire sa documentation ?

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

Interface
graphiques

Code natif

Python scientifique

Exemple minimal

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

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://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

Interface
graphiques

Code natif

Python scientifique

Outils d'extraction de documentation

Matthieu Falce

- ▶ sphinx (semi automatique) avec :
 - ▶ autosummary ³³
 - ▶ autodoc ³⁴
- ▶ sphinx (automatique) avec :
 - ▶ autoapi ³⁵
 - ▶ sphinx-autoapi ³⁶
- ▶ pdoc ³⁷
- ▶ pydoc ³⁸
- ▶ doxygen ³⁹

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Débug

Linter

Analyse des performances

Tests

Documentation

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

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

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

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

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

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

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

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

Syntaxe pour extraction automatique

- ▶ PEP 8 : <https://www.python.org/dev/peps/pep-0008/#documentation-strings>
- ▶ PEP 257 :
<https://www.python.org/dev/peps/pep-0257/>
- ▶ pdoc : markdown ⁴⁰
- ▶ doxygen : markdown + syntaxe spécifique ⁴¹
- ▶ sphinx : RestructuredText ⁴²
- ▶ sphinx avec extension Napoleon ⁴³
 - ▶ Google ⁴⁴
 - ▶ Numpy ⁴⁵

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

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

42.https://thomas-cokelaer.info/tutorials/sphinx/rest_syntax.html

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

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

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

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

Interface
graphiques

Code natif

Python scientifique

Formatage des docstrings – Doxygen

```
## @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
```

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

Interface
graphiques

Code natif

Python scientifique

Formatage des docstrings – Doxygen

Python

The screenshot shows a Doxygen-generated Python documentation page. At the top, there's a navigation bar with links to 'Main Page', 'Packages', and 'Classes'. Below the navigation bar, the title 'pyexample Namespace Reference' is displayed. A note says 'Documentation for this module: More...'. Under the 'Classes' section, there's a 'PyClass' entry with a link to its documentation. In the 'Functions' section, there's a 'func()' entry with a link to its documentation. A 'Detailed Description' section follows, containing a note about the module and a link to more details. Below that is a 'Function Documentation' section, which includes a 'func()' entry with a detailed description and a link to more details. The bottom right corner of the page indicates it was generated by 'doxygen 1.8.15'.

Résultat HTML de l'exemple précédent

Formatage des docstrings – reST

```
"""
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
"""
```

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

Interface
graphiques

Code natif

Python scientifique

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

Interface
graphiques

Code natif

Python scientifique

Formatage des docstrings – Google vs Numpy

```
"""
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
"""


```

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

Interface
graphiques

Code natif

Python scientifique

Formatage des docstrings – Google vs Numpy

```
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.

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

Interface
graphiques

Code natif

Python scientifique

Résultat HTML de l'exemple précédent

Bibliographie

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

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

Interface graphiques

Code natif

Python scientifique

Bibliothèque standard

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque standard

Batteries included

Module sys

Module os

Module subprocess

Mathématiques

Expressions régulières

Base de données

XML

JSON

Lecture / écritures fichiers métiers

Interaction réseau

Archivage des fichiers

Aller plus loin

Interface graphiques

Code natif

Python scientifique

“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 / parallèle / IPC / crypto ...
- ▶ multimédia (images, son, IHM)
- ▶ débuggeur, tests unitaires...
- ▶ gestion des dates, traductions...

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

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Batteries included

Module sys

Module os

Module subprocess

Mathématiques

Expressions régulières

Base de données

XML

JSON

Lecture / écritures fichiers
métiers

Interaction réseau

Archivage des fichiers

Aller plus loin

Interface
graphiques

Code natif

Python scientifique

sys

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("Ouiii")
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()
```

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Batteries included

Module sys

Module os

Module subprocess

Mathématiques

Expressions régulières

Base de données

XML

JSON

Lecture / écritures fichiers
métiers

Interaction réseau

Archivage des fichiers

Aller plus loin

Interface
graphiques

Code natif

Python scientifique

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())
```

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Batteries included

Module sys

Module subprocess

Mathématiques

Expressions régulières

Base de données

XML

JSON

Lecture / écritures fichiers
métiers

Interaction réseau

Archivage des fichiers

Aller plus loin

Interface
graphiques

Code natif

Python scientifique

OS

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")
)
```

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Batteries included

Module sys

Module subprocess

Mathématiques

Expressions régulières

Base de données

XML

JSON

Lecture / écritures fichiers
métiers

Interaction réseau

Archivage des fichiers

Aller plus loin

Interface
graphiques

Code natif

Python scientifique

Subprocess

Dédié à lancer des commandes "système" depuis python

- ▶ permet de lancer (*spawn*) des processus
- ▶ permet de se connecter à leur entrées / sortie et d'interagir avec
- ▶ permet un contrôle plus fin que `os.system` et donc privilégier dans les cas complexes

Pour créer les commandes à lancer (il faut une liste de strings) :

- ▶ utiliser le module `shlex` (spécialement conçu pour Unix)
- ▶ utiliser la méthode `split(" ")` des chaînes pour les cas simples

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Batteries included

Module sys

Module os

Module subprocess

Mathématiques

Expressions régulières

Base de données

XML

JSON

Lecture / écritures fichiers
métiers

Interaction réseau

Archivage des fichiers

Aller plus loin

Interface
graphiques

Code natif

Python scientifique

Subprocess

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Batteries included

Module sys

Module os

Module subprocess

Mathématiques

Expressions régulières

Base de données

XML

JSON

Lecture / écritures fichiers
métiers

Interaction réseau

Archivage des fichiers

Aller plus loin

Interface
graphiques

Code natif

Python scientifique

```
# les commandes sont lancées sous linux
import subprocess

# va bloquer jusqu'à la fin du process
# recommandé dans le cas général
subprocess.run(["bash", "-c", "ls /usr/bin | grep ls"], check=True)

# lancement dans un shell ou pas (plus besoin du bash -c)
subprocess.run(["ls /usr/bin | grep ls"], shell=True, check=True)

# capture de l'output
output = subprocess.run(["ls", "/tmp"], capture_output=True)
print(output.stdout)

# pipes
ls_process = subprocess.run(["ls", "/usr/bin"], stdout=subprocess.PIPE)
grep_process = subprocess.run(
    ["grep", "python"], input=ls_process.stdout, stdout=subprocess.PIPE
)
```

Subprocess

Matthieu Falce

On peut utiliser une syntaxe à base de context managers pour fermer les process automatiquement

```
# les commandes sont lancées sous linux
import subprocess

# non bloquant, permet de communiquer
# avec plusieurs process

with subprocess.Popen(
    ["echo", "salut\nje suis matthieu"], stdout=subprocess.PIPE
) as process_echo:
    with subprocess.Popen(
        ["grep", "salut"], stdin=process_echo.stdout, stdout=subprocess.PIPE
    ) as process_grep:
        stdout, stderr = process_grep.communicate()
        print(f"Output from stdout: {stdout}, {stderr}, ")

# Output from stdout: b'salut\n', None,
```

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Batteries included

Module sys

Module os

Module subprocess

Mathématiques

Expressions régulières

Base de données

XML

JSON

Lecture / écritures fichiers
métiers

Interaction réseau

Archivage des fichiers

Aller plus loin

Interface
graphiques

Code natif

Python scientifique

Subprocess

Matthieu Falce

Plus d'informations :

- ▶ <https://realpython.com/python-subprocess/>
- ▶ <https://docs.python.org/3/library/subprocess.html>

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Batteries included

Module sys

Module os

Module subprocess

Mathématiques

Expressions régulières

Base de données

XML

JSON

Lecture / écritures fichiers
métiers

Interaction réseau

Archivage des fichiers

Aller plus loin

Interface
graphiques

Code natif

Python scientifique

Outils mathématiques

Ne pas forcément utiliser ceux là pour les calculs scientifiques.

Ils sont plus lents que ceux de numpy / scipy

```
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
```

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Batteries included

Module sys

Module os

Module subprocess

Mathématiques

Expressions régulières

Base de données

XML

JSON

Lecture / écritures fichiers
métiers

Interaction réseau

Archivage des fichiers

Aller plus loin

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

Module subprocess

Mathématiques

Expressions régulières

Base de données

XML

JSON

Lecture / écritures fichiers
métiers

Interaction réseau

Archivage des fichiers

Aller plus loin

Interface
graphiques

Code natif

Python scientifique

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

Module subprocess

Mathématiques

Expressions régulières

Base de données

XML

JSON

Lecture / écritures fichiers
métiers

Interaction réseau

Archivage des fichiers

Aller plus loin

Interface
graphiques

Code natif

Python scientifique

Exemples

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Batteries included

Module sys

Module os

Module subprocess

Mathématiques

Expressions régulières

Base de données

XML

JSON

Lecture / écritures fichiers
métiers

Interaction réseau

Archivage des fichiers

Aller plus loin

Interface
graphiques

Code natif

Python scientifique

Expression	Chaînes capturées	Chaînes non cap- turées
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

Exemples

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-ako, ex-æquo
^Section .+	Section 1 / Section a / Section a.a/2	"" / Sectionner / voir Section 1
[1234567890]+ (,[1234567890]+)?	2 / 42 / 2,32 / 0.432	3, / ,643 / ""

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Batteries included

Module sys

Module os

Module subprocess

Mathématiques

Expressions régulières

Base de données

XML

JSON

Lecture / écritures fichiers
métiers

Interaction réseau

Archivage des fichiers

Aller plus loin

Interface
graphiques

Code natif

Python scientifique

Cas d'usages

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Batteries included

Module sys

Module os

Module subprocess

Mathématiques

Expressions régulières

Base de données

XML

JSON

Lecture / écritures fichiers
métiers

Interaction réseau

Archivage des fichiers

Aller plus loin

Interface
graphiques

Code natif

Python scientifique

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)

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

Module subprocess

Mathématiques

Expressions régulières

Base de données

XML

JSON

Lecture / écritures fichiers
métiers

Interaction réseau

Archivage des fichiers

Aller plus loin

Interface
graphiques

Code natif

Python scientifique

En python

```
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
```

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Batteries included

Module sys

Module os

Module subprocess

Mathématiques

Expressions régulières

Base de données

XML

JSON

Lecture / écritures fichiers
métiers

Interaction réseau

Archivage des fichiers

Aller plus loin

Interface
graphiques

Code natif

Python scientifique

En python

```
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\.\.]* c'est bien, (?P<mieux>[\w\.\.]* 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')]
```

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Batteries included

Module sys

Module os

Module subprocess

Mathématiques

Expressions régulières

Base de données

XML

JSON

Lecture / écritures fichiers
métiers

Interaction réseau

Archivage des fichiers

Aller plus loin

Interface
graphiques

Code natif

Python scientifique

Accès aux bases de données

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Batteries included

Module sys

Module os

Module subprocess

Mathématiques

Expressions régulières

Base de données

XML

JSON

Lecture / écritures fichiers
métiers

Interaction réseau

Archivage des fichiers

Aller plus loin

Interface
graphiques

Code natif

Python scientifique

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

Présentation DB API

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()
```

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Batteries included

Module sys

Module os

Module subprocess

Mathématiques

Expressions régulières

Base de données

XML

JSON

Lecture / écritures fichiers
métiers

Interaction réseau

Archivage des fichiers

Aller plus loin

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

Module subprocess

Mathématiques

Expressions régulières

Base de données

XML

JSON

Lecture / écritures fichiers
métiers

Interaction réseau

Archivage des fichiers

Aller plus loin

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

Module subprocess

Mathématiques

Expressions régulières

Base de données

XML

JSON

Lecture / écritures fichiers métiers

Interaction réseau

Archivage des fichiers

Aller plus loin

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("----- Sélection -----")
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

Module subprocess

Mathématiques

Expressions régulières

Base de données

XML

JSON

Lecture / écritures fichiers métiers

Interaction réseau

Archivage des fichiers

Aller plus loin

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

Module subprocess

Mathématiques

Expressions régulières

Base de données

XML

JSON

Lecture / écritures fichiers
métiers

Interaction réseau

Archivage des fichiers

Aller plus loin

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

Module subprocess

Mathématiques

Expressions régulières

Base de données

XML

JSON

Lecture / écritures fichiers
métiers

Interaction réseau

Archivage des fichiers

Aller plus loin

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

Module subprocess

Mathématiques

Expressions régulières

Base de données

XML

JSON

Lecture / écritures fichiers
métiers

Interaction réseau

Archivage des fichiers

Aller plus loin

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)
        db.commit()
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

Module subprocess

Mathématiques

Expressions régulières

Base de données

XML

JSON

Lecture / écritures fichiers
métiers

Interaction réseau

Archivage des fichiers

Aller plus loin

Interface
graphiques

Code natif

Python scientifique

Bibliographie / Aller plus loin

- ▶ <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 ⁴⁷ ⇒ abstraire les différences entre moteurs
 - ▶ SQLAlchemy
 - ▶ Pewee
 - ▶ PonyORM
 - ▶ ORM Django

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

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Batteries included

Module sys

Module os

Module subprocess

Mathématiques

Expressions régulières

Base de données

XML

JSON

Lecture / écritures fichiers métiers

Interaction réseau

Archivage des fichiers

Aller plus loin

Interface graphiques

Code natif

Python scientifique

XML ⁴⁸

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Batteries included

Module sys

Module os

Module subprocess

Mathématiques

Expressions régulières

Base de données

XML

JSON

Lecture / écritures fichiers métiers

Interaction réseau

Archivage des fichiers

Aller plus loin

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")
```

48.source : <https://stackoverflow.com/questions/1912434/how-do-i-parse-xml-in-python>
<https://stackoverflow.com/questions/3605680/creating-a-simple-xml-file-using-python>

XML 48

Matthieu Falce

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

48.source : <https://stackoverflow.com/questions/1912434/how-do-i-parse-xml-in-python>
<https://stackoverflow.com/questions/3605680/creating-a-simple-xml-file-using-python>

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Batteries included

Module sys

Module os

Module subprocess

Mathématiques

Expressions régulières

Base de données

XML

JSON

Lecture / écritures fichiers
métiers

Interaction réseau

Archivage des fichiers

Aller plus loin

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

Module subprocess

Mathématiques

Expressions régulières

Base de données

XML

JSON

Lecture / écritures fichiers
métiers

Interaction réseau

Archivage des fichiers

Aller plus loin

Interface
graphiques

Code natif

Python scientifique

JSON

Matthieu Falce



Certaines données ne sont pas JSON sérialisables. Il faut créer son propre serialiseur JSON dans ce cas. ⁴⁹

```
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))
```

49.<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

Module subprocess

Mathématiques

Expressions régulières

Base de données

XML

JSON

Lecture / écritures fichiers

métiers

Interaction réseau

Archivage des fichiers

Aller plus loin

Interface
graphiques

Code natif

Python scientifique

CSV – excel

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Batteries included

Module sys

Module os

Module subprocess

Mathématiques

Expressions régulières

Base de données

XML

JSON

Lecture / écritures fichiers

métiers

Interaction réseau

Archivage des fichiers

Aller plus loin

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

Module subprocess

Mathématiques

Expressions régulières

Base de données

XML

JSON

Lecture / écritures fichiers
métiers

Interaction réseau

Archivage des fichiers

Aller plus loin

Interface
graphiques

Code natif

Python scientifique

CSV – excel

Matthieu Falce

On peut utiliser xlrd, openpyxl ou pandas (qui se base sur
ces dernières) ⁵⁰

```
# 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

Module subprocess

Mathématiques

Expressions régulières

Base de données

XML

JSON

Lecture / écritures fichiers
métiers

Interaction réseau

Archivage des fichiers

Aller plus loin

Interface
graphiques

Code natif

Python scientifique

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

Bonus

Matthieu Falce

1. fichiers matrices Matlab

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Batteries included

Module sys

Module os

Module subprocess

Mathématiques

Expressions régulières

Base de données

XML

JSON

Lecture / écritures fichiers
métiers

Interaction réseau

Archivage des fichiers

Aller plus loin

Interface
graphiques

Code natif

Python scientifique

2. fichiers UFF

Matthieu Falce

3. fichiers OP2 et démo

```
import scipy.io
import pyuff
from pyNastran.op2.op2 import OP2

mat = scipy.io.loadmat('./fichiers_a_lire/exemple_MAT.mat')
print(mat)

model = OP2()
model.read_op2('./fichiers_a_lire/exemple_OP2.op2')
print(model.get_op2_stats())

uff_file = pyuff.UFF('./fichiers_a_lire/exemple_UFF.UFF')
uff_file
uff_file.read_sets()
```

Web – http

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

Module subprocess

Mathématiques

Expressions régulières

Base de données

XML

JSON

Lecture / écritures fichiers
métiers

Interaction réseau

Archivage des fichiers

Aller plus loin

Interface
graphiques

Code natif

Python scientifique

Web – http

Matthieu Falce

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

Module subprocess

Mathématiques

Expressions régulières

Base de données

XML

JSON

Lecture / écritures fichiers
métiers

Interaction réseau

Archivage des fichiers

Aller plus loin

Interface
graphiques

Code natif

Python scientifique

Web – http

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Batteries included

Module sys

Module os

Module subprocess

Mathématiques

Expressions régulières

Base de données

XML

JSON

Lecture / écritures fichiers
métiers

Interaction réseau

Archivage des fichiers

Aller plus loin

Interface
graphiques

Code natif

Python scientifique

On peut aussi lancer un serveur web vite fait sur sa machine :

```
python -m http.server
```

Sockets

```
# 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)
```

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Batteries included

Module sys

Module os

Module subprocess

Mathématiques

Expressions régulières

Base de données

XML

JSON

Lecture / écritures fichiers

métiers

Interaction réseau

Archivage des fichiers

Aller plus loin

Interface
graphiques

Code natif

Python scientifique

Manipulation de fichiers (archivage)

```
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()
```

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque

standard

Batteries included

Module sys

Module os

Module subprocess

Mathématiques

Expressions régulières

Base de données

XML

JSON

Lecture / écritures fichiers

métiers

Interaction réseau

Archivage des fichiers

Aller plus loin

Interface

graphiques

Code natif

Python scientifique

Manipulation de fichiers (archivage)

Matthieu Falce

```
from shutil import make_archive, copy  
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)  
copy(archive_name, "/tmp/my_archive")
```

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Batteries included

Module sys

Module os

Module subprocess

Mathématiques

Expressions régulières

Base de données

XML

JSON

Lecture / écritures fichiers
métiers

Interaction réseau

Archivage des fichiers

Aller plus loin

Interface
graphiques

Code natif

Python scientifique

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

Module subprocess

Mathématiques

Expressions régulières

Base de données

XML

JSON

Lecture / écritures fichiers
métiers

Interaction réseau

Archivage des fichiers

Aller plus loin

Interface
graphiques

Code natif

Python scientifique

Autres modules intéressants II

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

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Batteries included

Module sys

Module os

Module subprocess

Mathématiques

Expressions régulières

Base de données

XML

JSON

Lecture / écritures fichiers métiers

Interaction réseau

Archivage des fichiers

Aller plus loin

Interface
graphiques

Code natif

Python scientifique

Autres modules intéressants III

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Batteries included

Module sys

Module os

Module subprocess

Mathématiques

Expressions régulières

Base de données

XML

JSON

Lecture / écritures fichiers métiers

Interaction réseau

Archivage des fichiers

Aller plus loin

Interface
graphiques

Code natif

Python scientifique

Interface graphiques

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Tkinter

Structure du code

Placement widgets

Événements

QT

Conclusion

Code natif

Python scientifique

Contexte

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Tkinter

Contexte

IHM

Programmation
événementielle

MVC

Conteneurs

Widgets

Variables de contrôle

Menu

Structure du code

Placement widgets

Événements

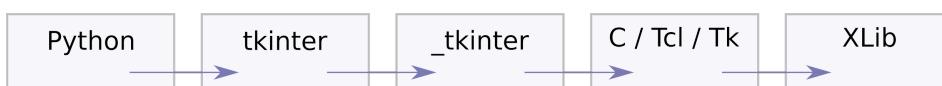
QT

Conclusion

Code natif

Python scientifique

- ▶ Tcl : langage de programmation ⁵¹
- ▶ Tk : toolkit d'IHM de Tcl ⁵²
- ▶ Tkinter : binding python pour Tcl / Tk



Etapes de traduction du code

51.https://fr.wikipedia.org/wiki/Tool_Command_Language

52.[https://fr.wikipedia.org/wiki/Tk_\(informatique\)](https://fr.wikipedia.org/wiki/Tk_(informatique))

Principe de fonctionnement des IHM

Par définition : on interagit avec une interface graphique

Problématiques :

- ▶ organisation de l'information (UX)
 - ▶ non traité ici
- ▶ réaction aux actions de l'utilisateur (informatique)
 - ▶ programmation événementielle
- ▶ rafraîchissement de l'interface (performance / ingénierie)
 - ▶ géré par le framework (normalement...) / optimisation
- ▶ garantir la simplicité du code (informatique / ingénierie)
 - ▶ patron de construction MVC

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Tkinter

Contexte

IHM

Programmation événementielle

MVC

Conteneurs

Widgets

Variables de contrôle

Menu

Structure du code

Placement widgets

Événements

QT

Conclusion

Code natif

Python scientifique

Programmation événementielle

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Tkinter

Contexte

IHM

Programmation événementielle

MVC

Conteneurs

Widgets

Variables de contrôle

Menu

Structure du code

Placement widgets

Événements

QT

Conclusion

Code natif

Python scientifique

En informatique, la programmation événementielle est un paradigme de programmation fondé sur les événements. Elle s'oppose à la programmation séquentielle. Le programme sera principalement défini par ses réactions aux différents événements qui peuvent se produire, c'est-à-dire des changements d'état de variable, par exemple l'incrémentation d'une liste, un mouvement de souris ou de clavier.

https://fr.wikipedia.org/wiki/Programmation_événementielle

Programmation événementielle

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Tkinter

Contexte

IHM

Programmation événementielle

MVC

Conteneurs

Widgets

Variables de contrôle

Menu

Structure du code

Placement widgets

Événements

QT

Conclusion

Code natif

Python scientifique

La programmation événementielle peut également être définie comme une technique d'architecture logicielle où l'application a une boucle principale divisée en deux sections : la première section détecte les événements, la seconde les gère. Elle est particulièrement mise en œuvre dans le domaine des interfaces graphiques.

https://fr.wikipedia.org/wiki/Programmation_évenementielle

Programmation événementielle

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Tkinter

Contexte

IHM

Programmation événementielle

MVC

Conteneurs

Widgets

Variables de contrôle

Menu

Structure du code

Placement widgets

Événements

QT

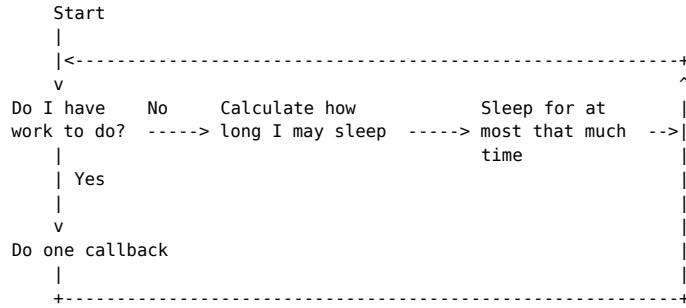
Conclusion

Code natif

Python scientifique

Programmation événementielle

Matthieu Falce



Source: <https://wiki.tcl.tk/1527>

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Tkinter

Contexte

IHM

Programmation
événementielle

MVC

Conteneurs

Widgets

Variables de contrôle

Menu

Structure du code

Placement widgets

Événements

QT

Conclusion

Code natif

Python scientifique

Programmation événementielle

Matthieu Falce

```
Main           Main           More
event ---> Callback ---> update ---> event ---> callbacks
loop           loop           as needed
```

Source: <https://wiki.tcl.tk/1527>

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Tkinter

Contexte

IHM

Programmation
événementielle

MVC

Conteneurs

Widgets

Variables de contrôle

Menu

Structure du code

Placement widgets

Événements

QT

Conclusion

Code natif

Python scientifique

Programmation événementielle

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Tkinter

Contexte

IHM

Programmation
événementielle

MVC

Conteneurs

Widgets

Variables de contrôle

Menu

Structure du code

Placement widgets

Événements

QT

Conclusion

Code natif

Python scientifique



Les callbacks doivent s'exécuter rapidement.
Sinon blocage de la boucle d'événement

Programmation événementielle

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Tkinter

Contexte

IHM

Programmation
événementielle

MVC

Conteneurs

Widgets

Variables de contrôle

Menu

Structure du code

Placement widgets

Événements

QT

Conclusion

Code natif

Python scientifique

Bonus : boucle d'événement minimale (en tkinter)

```
import tkinter

while True:
    tkinter.update_idletasks()
    tkinter.update()

## équivalent à
# tkinter.mainloop()
```

Permet de rajouter sa propre boucle d'événements
(modélisation physique par exemple)

Patron de conception : Modèle - Vue - Contrôleur

Modèle-vue-contrôleur ou MVC est un motif d'architecture logicielle destiné aux interfaces graphiques lancé en 1978 et très populaire pour les applications web. Le motif est composé de trois types de modules ayant trois responsabilités différentes : les modèles, les vues et les contrôleurs.

<https://fr.wikipedia.org/wiki/Modèle-vue-contrôleur>

MVC est également très utilisé pour l'architecture d'interfaces graphiques

Matthieu Falce
Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Bibliothèque standard
Interface graphiques
Tkinter
Contexte
IHM
Programmation événementielle
MVC
Conteneurs
Widgets
Variables de contrôle
Menu
Structure du code
Placement widgets
Événements
QT
Conclusion
Code natif
Python scientifique

Patron de conception : Modèle - Vue - Contrôleur

- ▶ le modèle (Model) : contient les données à afficher
 - ▶ base de données
 - ▶ liste de nom en mémoire
 - ▶ API
- ▶ la vue (View) : contient la présentation de l'interface graphique
 - ▶ tableau
 - ▶ HTML
- ▶ le contrôleur (Controller) contient la logique concernant les actions effectuées par l'utilisateur
 - ▶ supprimer une ligne des données
 - ▶ mettre à jour une information

Matthieu Falce
Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Bibliothèque standard
Interface graphiques
Tkinter
Contexte
IHM
Programmation événementielle
MVC
Conteneurs
Widgets
Variables de contrôle
Menu
Structure du code
Placement widgets
Événements
QT
Conclusion
Code natif
Python scientifique

Patron de conception : Modèle - Vue - Contrôleur

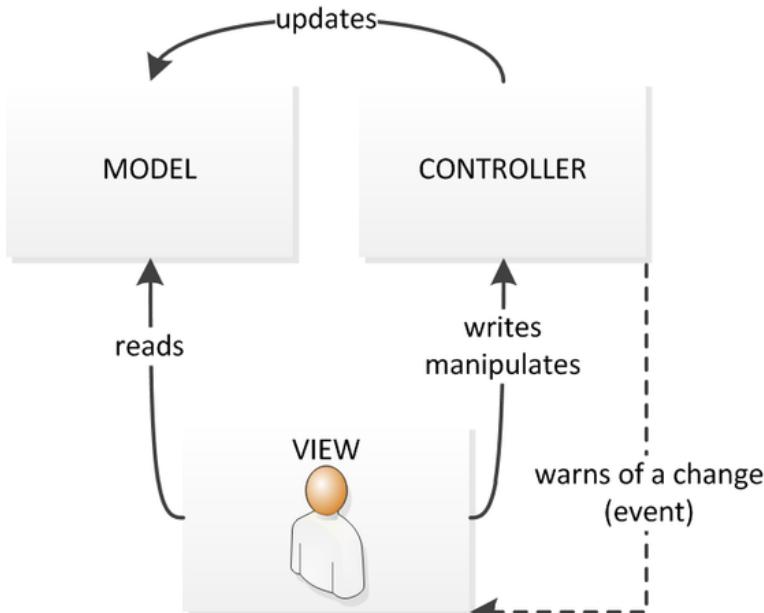


Schéma du modèle MVC

Source: <https://fr.wikipedia.org/wiki/Mod%C3%A8le-vue-contr%C3%B4leur#/media/File:ModeleMVC.png>

Patron de conception : Modèle - Vue - Contrôleur

Et Tcl / Tk dans tout ça ?

In Tkinter, the standard widgets all use tight coupling between the model and the view; the model data is managed by the actual widget instance. Unfortunately, this means that you cannot display data from the same model in two different widgets (for example, two independent views into a text editor buffer). It also means that you have to convert your data to a form suitable for Tk.

<http://effbot.org/zone/model-view-controller.htm>

Inspiration du MVC pour découpler et éviter le code spaghetti.

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Tkinter

Contexte

IHM

Programmation événementielle

MVC

Conteneurs

Widgets

Variables de contrôle

Menu

Structure du code

Placement widgets

Événements

QT

Conclusion

Code natif

Python scientifique

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Tkinter

Contexte

IHM

Programmation événementielle

MVC

Conteneurs

Widgets

Variables de contrôle

Menu

Structure du code

Placement widgets

Événements

QT

Conclusion

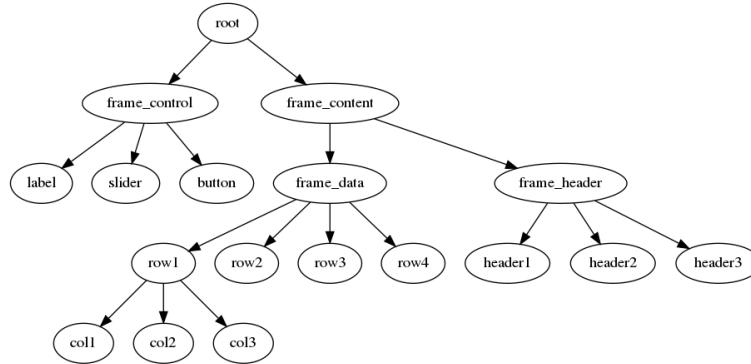
Code natif

Python scientifique

Conteneurs

Le conteneur principal est un cadre (Frame).

- ▶ la fenêtre principale est un cadre
- ▶ chaque cadre possède son propre système de positionnement
- ▶ permet de créer des applications modulaires



Example de hiérarchie de widgets

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Tkinter

Contexte

IHM

Programmation

événementielle

MVC

Conteneurs

Widgets

Variables de contrôle

Menu

Structure du code

Placement widgets

Événements

QT

Conclusion

Code natif

Python scientifique

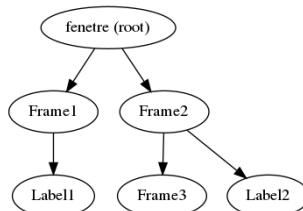
Conteneurs 53

Frames

```
from tkinter import *
fenetre = Tk(); fenetre['bg']='white'

# frame 1
Frame1 = Frame(fenetre, borderwidth=2, relief=GROOVE)
Frame1.pack(side=LEFT, padx=30, pady=30)
# frame 2
Frame2 = Frame(fenetre, borderwidth=2, relief=GROOVE)
Frame2.pack(side=LEFT, padx=10, pady=10)
# frame 3 dans frame 2
Frame3 = Frame(Frame2, bg="white", borderwidth=2, relief=GROOVE)
Frame3.pack(side=RIGHT, padx=5, pady=5)
# Ajout de labels
Label(Frame1, text="Frame 1").pack(padx=10, pady=10)
Label(Frame2, text="Frame 2").pack(padx=10, pady=10)
Label(Frame3, text="Frame 3",bg="white").pack(padx=10, pady=10)

fenetre.mainloop()
```



Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Tkinter

Contexte

IHM

Programmation

événementielle

MVC

Conteneurs

Widgets

Variables de contrôle

Menu

Structure du code

Placement widgets

Événements

QT

Conclusion

Code natif

Python scientifique

53.Exemples inspirés de
<https://apprendre-python.com/page-tkinter-interface-graphique-python-tutoriel>

Conteneurs 53

Matthieu Falce

Frames

```
from tkinter import *
fenetre = Tk(); fenetre['bg']='white'

# frame 1
Frame1 = Frame(fenetre, borderwidth=2, relief=GROOVE)
Frame1.pack(side=LEFT, padx=30, pady=30)
# frame 2
Frame2 = Frame(fenetre, borderwidth=2, relief=GROOVE)
Frame2.pack(side=LEFT, padx=10, pady=10)
# frame 3 dans frame 2
Frame3 = Frame(Frame2, bg="white", borderwidth=2, relief=GROOVE)
Frame3.pack(side=RIGHT, padx=5, pady=5)
# Ajout de labels
Label(Frame1, text="Frame 1").pack(padx=10, pady=10)
Label(Frame2, text="Frame 2").pack(padx=10, pady=10)
Label(Frame3, text="Frame 3",bg="white").pack(padx=10, pady=10)

fenetre.mainloop()
```



53.Exemples inspirés de
<https://apprendre-python.com/page-tkinter-interface-graphique-python-tutoriel>

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Tkinter
Contexte
IHM
Programmation
événementielle
MVC

Conteneurs

Widgets
Variables de contrôle
Menu
Structure du code
Placement widgets
Événements
QT
Conclusion

Code natif

Python scientifique

Conteneurs 53

Matthieu Falce

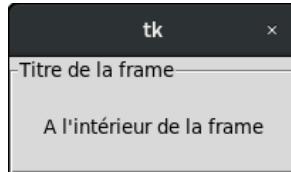
LabelFrame

```
from tkinter import *
fenetre = Tk()

l = LabelFrame(fenetre, text="Titre de la frame", padx=20, pady=20)
l.pack(fill="both", expand="yes")

Label(l, text="A l'intérieur de la frame").pack()

fenetre.mainloop()
```



53.Exemples inspirés de
<https://apprendre-python.com/page-tkinter-interface-graphique-python-tutoriel>

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Tkinter
Contexte
IHM
Programmation
événementielle
MVC

Conteneurs

Widgets
Variables de contrôle
Menu
Structure du code
Placement widgets
Événements
QT
Conclusion

Code natif

Python scientifique

Conteneurs 53

Paned window (peuvent se redimensionner)

```
from tkinter import *
fenetre = Tk()
p = PanedWindow(fenetre, orient=HORIZONTAL)
p.pack(side=TOP, expand=Y, fill=BOTH, pady=2, padx=2)
p.add(Label(p, text='Volet 1', background='blue', anchor=CENTER))
p.add(Label(p, text='Volet 2', background='white', anchor=CENTER) )
p.add(Label(p, text='Volet 3', background='red', anchor=CENTER) )
p.pack()
p2 = PanedWindow(fenetre, orient=VERTICAL)
p2.pack(side=BOTTOM, expand=Y, fill=BOTH, pady=2, padx=2)
p2.add(Label(p2, text='Volet 1', background='blue', anchor=CENTER))
p2.add(Label(p2, text='Volet 2', background='white', anchor=CENTER) )
p2.add(Label(p2, text='Volet 3', background='red', anchor=CENTER) )
p2.pack()
fenetre.mainloop()
```



53.Exemples inspirés de
<https://apprendre-python.com/page-tkinter-interface-graphique-python-tutoriel>

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Tkinter
Contexte
IHM
Programmation
événementielle
MVC

Conteneurs

Widgets

Variables de contrôle

Menu

Structure du code

Placement widgets

Événements

QT

Conclusion

Code natif

Python scientifique

Widgets 55

Composant d'interface graphiques avec lequel on peut interagir⁵⁴

- ▶ Label
- ▶ Button
- ▶ Text
- ▶ RadioButton
- ▶ ListBox
- ▶ Menu
- ▶ ...

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Tkinter
Contexte
IHM
Programmation
événementielle
MVC

Conteneurs

Widgets

Variables de contrôle

Menu

Structure du code

Placement widgets

Événements

QT

Conclusion

Code natif

Python scientifique

54.https://fr.wikipedia.org/wiki/Composant_d'interface_graphique

55.exemples inspirés de

<https://apprendre-python.com/page-tkinter-interface-graphique-python-tutoriel>

Widgets 54

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Tkinter

Contexte

IHM

Programmation
événementielle

MVC

Conteneurs

Widgets

Variables de contrôle

Menu

Structure du code

Placement widgets

Événements

QT

Conclusion

Code natif

Python scientifique

Label

```
from tkinter import *\n\nfenetre = Tk()\n\nlabel = Label(fenetre, text="Du texte ou une image", bg="yellow")\nlabel.pack()\n\nfenetre.mainloop()
```



54.exemples inspirés de
<https://apprendre-python.com/page-tkinter-interface-graphique-python-tutoriel>

Widgets 54

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Tkinter

Contexte

IHM

Programmation
événementielle

MVC

Conteneurs

Widgets

Variables de contrôle

Menu

Structure du code

Placement widgets

Événements

QT

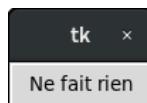
Conclusion

Code natif

Python scientifique

Button

```
from tkinter import *\n\nfenetre = Tk()\n\nbouton = Button(fenetre, text="Ne fait rien")\nbouton.pack()\n\nfenetre.mainloop()
```



54.exemples inspirés de
<https://apprendre-python.com/page-tkinter-interface-graphique-python-tutoriel>

Widgets 54

List

```
from tkinter import *

fenetre = Tk()

liste = Listbox(fenetre)
liste.insert(1, "Python")
liste.insert(2, "PHP")
liste.insert(3, "CSS")
liste.insert(4, "Javascript")

liste.pack()

# pour savoir ce qui est sélectionné
index_selectionnes = liste.curselection()
if index_selectionnes:
    # index est un tuple avec les indexs sélectionnés
    valeur_selectionnee = liste.get(index_selectionnes[0])

fenetre.mainloop()
```



54.exemples inspirés de
<https://apprendre-python.com/page-tkinter-interface-graphique-python-tutoriel>

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Tkinter

Contexte

IHM

Programmation événementielle

MVC

Conteneurs

Widgets

Variables de contrôle

Menu

Structure du code

Placement widgets

Événements

QT

Conclusion

Code natif

Python scientifique

Widgets 54

Canvas

```
from tkinter import *

fenetre = Tk()

canvas = Canvas(fenetre, width=150, height=120, background='yellow')
ligne1 = canvas.create_line(75, 0, 75, 120)
ligne2 = canvas.create_line(0, 60, 150, 60)
txt = canvas.create_text(75, 60, text="Cible", font="Arial 16 italic", fill="blue")
canvas.pack()

fenetre.mainloop()
```



54.exemples inspirés de
<https://apprendre-python.com/page-tkinter-interface-graphique-python-tutoriel>

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Tkinter

Contexte

IHM

Programmation événementielle

MVC

Conteneurs

Widgets

Variables de contrôle

Menu

Structure du code

Placement widgets

Événements

QT

Conclusion

Code natif

Python scientifique

Widgets 54

Matthieu Falce

Scale

```
from tkinter import *

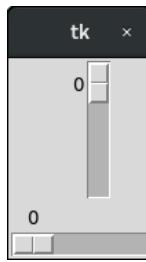
fenetre = Tk()

scale_ver = Scale(fenetre)
scale_ver.pack()

scale_hor = Scale(fenetre, orient="horizontal")
scale_hor.pack()

# TODO : get value

fenetre.mainloop()
```



54.exemples inspirés de
<https://apprendre-python.com/page-tkinter-interface-graphique-python-tutoriel>

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Tkinter

Contexte

IHM

Programmation événementielle

MVC

Conteneurs

Widgets

Variables de contrôle

Menu

Structure du code

Placement widgets

Événements

QT

Conclusion

Code natif

Python scientifique

Widgets 54

Matthieu Falce

Scrollbar

```
from tkinter import *

fenetre = Tk()

scrollbar = Scrollbar(fenetre)
scrollbar.pack(side=RIGHT, fill=Y)

# double connection :
# * on scroll dans le widget => met à jour scrollbar
# * on bouge l'ascenseur => met à jour le widget
listbox = Listbox(fenetre, yscrollcommand=scrollbar.set)
for i in range(1000):
    listbox.insert(END, "ligne : " + str(i))
listbox.pack(side=LEFT, fill=BOTH)

scrollbar.config(command=listbox.yview)

fenetre.mainloop()
```

- ▶ permet d'afficher des widgets plus gros que la fenêtre
- ▶ modifie le scroll en X ou Y
- ▶ s'utilise avec :
 - ▶ ListBox
 - ▶ Text
 - ▶ Canvas

54.exemples inspirés de
<https://apprendre-python.com/page-tkinter-interface-graphique-python-tutoriel>

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Tkinter

Contexte

IHM

Programmation événementielle

MVC

Conteneurs

Widgets

Variables de contrôle

Menu

Structure du code

Placement widgets

Événements

QT

Conclusion

Code natif

Python scientifique

Variables de contrôle

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Tkinter
Contexte

IHM

Programmation
événementielle

MVC

Conteneurs

Widgets

Variables de contrôle

Menu

Structure du code

Placement widgets

Événements

QT

Conclusion

Code natif

Python scientifique

Les variables modifiées en Tk (dans des widgets par exemple) ne sont pas modifiées en Python

Les classes Variables

- ▶ BooleanVar
- ▶ DoubleVar
- ▶ IntVar
- ▶ StringVar

Certains *widgets* en ont besoin pour fonctionner

Variables de contrôle

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Tkinter
Contexte

IHM

Programmation
événementielle

MVC

Conteneurs

Widgets

Variables de contrôle

Menu

Structure du code

Placement widgets

Événements

QT

Conclusion

Code natif

Python scientifique

CheckBox

```
from tkinter import *  
  
fenetre = Tk()  
var = IntVar()  
  
bouton = Checkbutton(fenetre, text="J'accepte les CGU", variable=var)  
bouton.pack()  
  
# récupération de la valeur  
print(var.get())  
  
fenetre.mainloop()
```



Variables de contrôle

Matthieu Falce

RadioButton

```
from tkinter import *

fenetre = Tk()

value = IntVar()
bouton1 = Radiobutton(fenetre, text="H", variable=value, value=1)
bouton2 = Radiobutton(fenetre, text="F", variable=value, value=2)
bouton3 = Radiobutton(fenetre, text="Autre", variable=value, value=3)
bouton1.pack()
bouton2.pack()
bouton3.pack()

valeur = value.get(); print(type(valeur), valeur)

fenetre.mainloop()
```



Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Bibliothèque standard
Interface graphiques
Tkinter
Contexte
IHM
Programmation événementielle
MVC
Conteneurs
Widgets
Variables de contrôle
Menu
Structure du code
Placement widgets
Événements
QT
Conclusion
Code natif
Python scientifique

Variables de contrôle

Matthieu Falce

Scale – la suite

```
from tkinter import *

fenetre = Tk()

value = DoubleVar()
scale = Scale(fenetre, variable=value)
scale.pack()

valeur = value.get()
print(type(valeur), valeur)

fenetre.mainloop()
```

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Bibliothèque standard
Interface graphiques
Tkinter
Contexte
IHM
Programmation événementielle
MVC
Conteneurs
Widgets
Variables de contrôle
Menu
Structure du code
Placement widgets
Événements
QT
Conclusion
Code natif
Python scientifique

Variables de contrôle

Matthieu Falce

Entry

```
from tkinter import *

fenetre = Tk()

value = StringVar()
value.set("Valeur")
entree = Entry(fenetre, textvariable=value, width=30)
entree.pack()

# label est mis à jour tout automatiquement
label = Label(fenetre, textvariable=value)
label.pack()

valeur = value.get()
print(type(valeur), valeur)

fenetre.mainloop()
```

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Bibliothèque standard
Interface graphiques
Tkinter
Contexte
IHM
Programmation événementielle
MVC
Conteneurs
Widgets
Variables de contrôle
Menu
Structure du code
Placement widgets
Événements
QT
Conclusion
Code natif
Python scientifique

Variables de contrôle

Matthieu Falce

Entry – validation

```
# plus de détails ici
# https://stackoverflow.com/questions/4140437/
# ou ici : http://tkinter.fdex.eu/doc/entw.html

from tkinter import *

fenetre = Tk()

def validate(valeur_dans_entry):
    print("passée:", valeur_dans_entry)
    if valeur_dans_entry == "a":
        return True
    fenetre.bell()
    return False

# validation désactivée avec les StringVar
# on peut enregistrer la valeur dans une globale
# ou utiliser les callbacks pour la modification de la Variable sinon...
# key : appelle la validation à chaque appui de touche
# %P : la valeur que l'on aurait eue si c'était valide
tcl_function_validate = (fenetre.register(validate), "%P")
entree = Entry(
    fenetre, width=30, validate="key",
    validatecommand=tcl_function_validate
)
entree.pack()

fenetre.mainloop()
```

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Bibliothèque standard
Interface graphiques
Tkinter
Contexte
IHM
Programmation événementielle
MVC
Conteneurs
Widgets
Variables de contrôle
Menu
Structure du code
Placement widgets
Événements
QT
Conclusion
Code natif
Python scientifique

Widget quizz

Matthieu Falce

Quels types de widgets pour quelle interaction ?

- ▶ entrer un numéro de téléphone ⁵⁵
- ▶ sélectionner un volume ⁵⁶
- ▶ créer un mot de passe ⁵⁷
- ▶ choisir dans une liste d'actions ⁵⁸
- ▶ choisir un login / mot de passe ⁵⁹

55.<https://qz.com/679782/programmers-imagine-the-most-ridiculous-ways-to-input-a-phone-number/>
56.<https://uxdesign.cc/the-worst-volume-control-ui-in-the-world-60713dc86950>
57.https://www.reddit.com/r/ProgrammerHumor/comments/904mko/password_input_with_extra_security/
58.<https://www.extremetech.com/extreme/262166-hawaiis-missile-scare-driven-terrible-ui-fc-c-launches-investigation>
59.https://www.reddit.com/r/ProgrammerHumor/comments/8r9xua/so_ive_heard_we_are_now_makin_g_logins_right/

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Tkinter

Contexte

IHM

Programmation événementielle

MVC

Conteneurs

Widgets

Variables de contrôle

Menu

Structure du code

Placement widgets

Événements

QT

Conclusion

Code natif

Python scientifique

Barre de menu

Matthieu Falce

```
from tkinter import *

def ma_fonction():
    print('coucou', bv.get(), rv.get())

fenetre = Tk()
menubar = Menu(fenetre)

bv = BooleanVar(fenetre)
rv = StringVar(fenetre)

menu1 = Menu(menubar, tearoff=0)
menu1.add_command(label="Nouveau", command=ma_fonction)
menu1.add_checkbutton(
    label="Autosave", variable=bv, command=ma_fonction)
menubar.add_cascade(label="Fichier", menu=menu1)

menu2 = Menu(menubar, tearoff=0)
menu2.add_radiobutton(label='rouge', variable=rv, value="(1, 0, 0)")
menu2.add_radiobutton(label='vert', variable=rv, value="(0, 1, 0)")
menubar.add_cascade(label="Couleurs", menu=menu2)
menu1.add_cascade(label="Couleurs", menu=menu2) # sous menu

fenetre.config(menu=menubar)
fenetre.mainloop()
```

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Tkinter

Contexte

IHM

Programmation événementielle

MVC

Conteneurs

Widgets

Variables de contrôle

Menu

Structure du code

Placement widgets

Événements

QT

Conclusion

Code natif

Python scientifique

Barre de menu

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Tkinter

Contexte

IHM

Programmation
événementielle

MVC

Conteneurs

Widgets

Variables de contrôle

Menu

Structure du code

Placement widgets

Événements

QT

Conclusion

Code natif

Python scientifique

Structure du code⁶²

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Tkinter

Structure du code

Placement widgets

Événements

QT

Conclusion

Code natif

Python scientifique

► gros codes → encapsulation dans des classes⁶⁰

► soit classe normale / soit widget custom

► pour une classe normale on passe le widget parent

► si on hérite de Tk.frame / de Tk on crée un widget⁶¹

► permet une réutilisation facile dans d'autres projets

60.<https://softwareengineering.stackexchange.com/questions/213935/why-use-classes-when-programming-a-tkinter-gui-in-python>

61.<https://stackoverflow.com/questions/7300072/inheriting-from-frame-or-not-in-a-tkinter-application>

62.<https://stackoverflow.com/questions/17466561/best-way-to-structure-a-tkinter-application/17470842>

Approche orientée objet

```
# source
# https://www.pythontutorial.net/tkinter/tkinter-object-oriented-window/

import tkinter as tk
from tkinter import ttk
from tkinter.messagebox import showinfo

class App(tk.Tk):
    def __init__(self):
        super().__init__()

        # configure the root window
        self.title("My Awesome App")
        self.geometry("300x50")

        # label
        self.label = ttk.Label(self, text="Hello, Tkinter!")
        self.label.pack()

        # button
        self.button = ttk.Button(self, text="Click Me")
        self.button["command"] = self.button_clicked
        self.button.pack()

    def button_clicked(self):
        showinfo(title="Information", message="Hello, Tkinter!")

if __name__ == "__main__":
    app = App()
    app.mainloop()
```

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Tkinter

Structure du code

Placement widgets

Événements

QT

Conclusion

Code natif

Python scientifique

Approche orientée objet

```
# source
# https://stackoverflow.com/questions/17466561/

import tkinter as tk

class MainApplication(tk.Frame):
    def __init__(self, parent, *args, **kwargs):
        tk.Frame.__init__(self, parent, *args, **kwargs)
        self.parent = parent

        <create the rest of your GUI here>

if __name__ == "__main__":
    root = tk.Tk()
    MainApplication(root).pack(side="top", fill="both", expand=True)
    root.mainloop()
```

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Tkinter

Structure du code

Placement widgets

Événements

QT

Conclusion

Code natif

Python scientifique

Layout Managers

Matthieu Falce

2 algorithmes de layout :

- ▶ pack
 - ▶ placement des éléments en fonction des autres
 - ▶ le plus simple
- ▶ grid
 - ▶ placement des éléments sur une grille
 - ▶ le plus puissant

Options :

- ▶ expand
- ▶ fill
- ▶ padding : ipadx / ipady / padx / pady

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Tkinter

Structure du code

Placement widgets

Événements

QT

Conclusion

Code natif

Python scientifique

Packing

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Tkinter

Structure du code

Placement widgets

Événements

QT

Conclusion

Code natif

Python scientifique

```
"""
Placement du widget Listbox utilisant toute la fenêtre.

"""

from tkinter import *
root = Tk()

listbox = Listbox(root)
listbox.pack(fill=BOTH, expand=1)

for i in range(20):
    listbox.insert(END, str(i))

mainloop()
```

Packing

Matthieu Falce

```
"""
Les widgets sont placés les uns sous les autres
et occupent toute la largeur (en X).
"""

from tkinter import *
root = Tk()

w = Label(root, text="Red", bg="red", fg="white")
w.pack(fill=X)
w = Label(root, text="Green", bg="green", fg="black")
w.pack(fill=X)
w = Label(root, text="Blue", bg="blue", fg="white")
w.pack(fill=X)

mainloop()
```

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Tkinter

Structure du code

Placement widgets

Événements

QT

Conclusion

Code natif

Python scientifique

Packing

Matthieu Falce

```
"""
Placement des widgets les uns à la gauche des autres
"""

from tkinter import *
root = Tk()

w = Label(root, text="Bleu", bg="blue", fg="white")
w.pack(side=LEFT)
w = Label(root, text="Blanc", bg="white", fg="black")
w.pack(side=LEFT)
w = Label(root, text="Rouge", bg="red", fg="white")
w.pack(side=LEFT)

mainloop()
```

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Tkinter

Structure du code

Placement widgets

Événements

QT

Conclusion

Code natif

Python scientifique

Grid layout

```
""" Utilisation du grid layout pour construire une
interface plus complexe.
"""

from tkinter import *
fen1 = Tk()

# création de widgets 'Label' et 'Entry' :
txt1 = Label(fen1, text="Premier champ :")
txt2 = Label(fen1, text="Second :")
txt3 = Label(fen1, text="Troisième :")
entr1 = Entry(fen1)
entr2 = Entry(fen1)
entr3 = Entry(fen1)

# création d'un widget 'Canvas' contenant une image bitmap :
can1 = Canvas(fen1, width=160, height=160, bg="white")
photo = PhotoImage(file="ptichat.png")
item = can1.create_image(80, 80, image=photo)

# Mise en page à l'aide de la méthode 'grid' :
txt1.grid(row=1, sticky=E)
txt2.grid(row=2, sticky=E)
txt3.grid(row=3, sticky=E)
entr1.grid(row=1, column=2)
entr2.grid(row=2, column=2)
entr3.grid(row=3, column=2)
can1.grid(row=1, column=3, rowspan=3, padx=10, pady=5)

# démarrage :
fen1.mainloop()
```

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Tkinter

Structure du code

Placement widgets

Événements

QT

Conclusion

Code natif

Python scientifique

Gestion des événements

Plusieurs façons de réagir aux événements

- ▶ **command** : appelle un fonction quand on clic / interagit sur un widget
- ▶ **bind** : relie une fonction à un événement particulier
- ▶ **trace** : appelle une fonction quand on change une *Var
- ▶ **after** : exécute une fonction après N millisecondes

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Tkinter

Structure du code

Placement widgets

Événements

Gestionnaire de fenêtre

Multifenêtre

QT

Conclusion

Code natif

Python scientifique

Gestion des événements

Matthieu Falce

command

La plupart des widgets ont une méthode command

```
from tkinter import *

def on_click():
    print("clic")

fenetre = Tk()

bouton = Button(fenetre, text="clic", command=on_click)
bouton.pack()

fenetre.mainloop()
```

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Tkinter

Structure du code

Placement widgets

Événements

Gestionnaire de fenêtre

Multifenêtre

QT

Conclusion

Code natif

Python scientifique

Gestion des événements

Matthieu Falce

command

Comment passer des paramètres à la fonction ?

```
from tkinter import *

def on_click(bouton_id):
    print("clic", bouton_id)

fenetre = Tk()

bouton1 = Button(fenetre, text="clic", command=lambda: on_click(1))
bouton1.pack()

bouton2 = Button(fenetre, text="clic 2", command=lambda: on_click(2))
bouton2.pack()

fenetre.mainloop()
```

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Tkinter

Structure du code

Placement widgets

Événements

Gestionnaire de fenêtre

Multifenêtre

QT

Conclusion

Code natif

Python scientifique

Gestion des événements

Matthieu Falce

```
bind

from tkinter import *
fenetre = Tk()

def clavier(event):
    touche = event.keysym
    print(touche)

def mouvement(event):
    pos = event.x, event.y
    print(pos, event.widget)

canvas = Canvas(fenetre, width=500, height=500)
label = Label(fenetre, text="Survolez moi", height=10)

canvas.bind("<B1-Motion>", mouvement)
label.bind("<Motion>", mouvement)
fenetre.bind("<Key>", clavier)

canvas.pack()
label.pack()

fenetre.mainloop()
```

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Tkinter
Structure du code
Placement widgets

Événements

Gestionnaire de fenêtre
Multifenêtre

QT
Conclusion

Code natif

Python scientifique

Gestion des événements

Matthieu Falce

bind

L'objet event⁶³

- ▶ passé aux fonctions bindées
- ▶ toujours les même champs, quelque soit l'événement
- ▶ contient les informations sur l'événement
 - ▶ le widget d'appel
 - ▶ la position de l'événement
 - ▶ la touche pressée
 - ▶ ...

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Tkinter
Structure du code
Placement widgets

Événements

Gestionnaire de fenêtre
Multifenêtre

QT
Conclusion

Code natif

Python scientifique

63.<http://tkinter.fdex.eu/doc/event.html>

Gestion des événements

Matthieu Falce

bind

Liste des événements que l'on peut binder :

- ▶ <Button-1> : Click gauche
- ▶ <Button-2> : Click milieu
- ▶ <Button-3> : Click droit
- ▶ <Double-Button-1> : Double click droit
- ▶ <Double-Button-2> : Double click gauche
- ▶ <KeyPress> : Pression sur une touche
- ▶ <KeyPress-a> : Pression sur la touche A (minuscule)
- ▶ <Return> : Pression sur la touche entrée
- ▶ <Escape> : Touche Echap

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Tkinter

Structure du code

Placement widgets

Événements

Gestionnaire de fenêtre

Multifenêtre

QT

Conclusion

Code natif

Python scientifique

Gestion des événements

Matthieu Falce

bind

- ▶ <Up> : Pression sur la flèche directionnelle haut
- ▶ <Down> : Pression sur la flèche directionnelle bas
- ▶ <ButtonRelease> : Lorsque qu'on relâche le click
- ▶ <Motion> : Mouvement de la souris
- ▶ <B1-Motion> : Mouvement de la souris avec click gauche
- ▶ <Enter> : Entrée du curseur dans un widget
- ▶ <Leave> : Sortie du curseur dans un widget
- ▶ <Configure> : Redimensionnement de la fenêtre
- ▶ <Map> <Unmap> : Ouverture et iconification de la fenêtre
- ▶ <MouseWheel> : Utilisation de la roulette

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Tkinter

Structure du code

Placement widgets

Événements

Gestionnaire de fenêtre

Multifenêtre

QT

Conclusion

Code natif

Python scientifique

Gestion des événements

Matthieu Falce

trace

```
from tkinter import *

def mise_a_jour_valeur(*args):
    print(value.get())

fenetre = Tk()

value = StringVar()
value.set("Valeur")
entree = Entry(fenetre, textvariable=value)
entree.pack()

# on peut choisir d'avoir des infos
# quand la variable est lue ("r") / écrite ("w")
value.trace("w", mise_a_jour_valeur)

fenetre.mainloop()
```

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Tkinter

Structure du code

Placement widgets

Événements

Gestionnaire de fenêtre

Multifenêtre

QT

Conclusion

Code natif

Python scientifique

wm 63

Matthieu Falce

Permet de modifier le comportement et l'apparence de la fenêtre. Dépend du gestionnaire de fenêtre (Window Manager) de l'OS ⇒ options non multiplateforme

```
# source : https://stackoverflow.com/questions/33286544/
from tkinter import *
frame = Tk()

# Remove shadow & drag bar. Note: Must be used before
# wm calls otherwise these will be removed.
frame.overrideredirect(1)

# Always keep window on top of others
# appel aux attributs en Tk
frame.call("wm", "attributes", ".", "-topmost", "true")
# appel à l'attribut objet
frame.topmost = True

# Set offset from top-left corner of screen as well as size
frame.geometry("100x100+500+500")

# Fullscreen mode
frame.call("wm", "attributes", ".", "-fullscreen", "true")

# Window Opacity 0.0-1.0
frame.call("wm", "attributes", ".", "-alpha", "0.9")

frame.mainloop()
```

63.<https://wiki.tcl.tk/9457>

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Tkinter

Structure du code

Placement widgets

Événements

Gestionnaire de fenêtre

Multifenêtre

QT

Conclusion

Code natif

Python scientifique

Applications Multifenêtre

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Tkinter

Structure du code

Placement widgets

Événements

Gestionnaire de fenêtre

Multifenêtre

QT

Conclusion

Code natif

Python scientifique

- ▶ choix d'un fichier / dossier
- ▶ réponse à une question
- ▶ formulaire supplémentaire pour finir une action
- ▶ "simplifier" la présentation

Message / dialogues / popup

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Tkinter

Structure du code

Placement widgets

Événements

Gestionnaire de fenêtre

Multifenêtre

QT

Conclusion

Code natif

Python scientifique

Interaction ponctuelle avec l'utilisateur.
Poser une question / informer...

- ▶ showinfo, showwarning, showerror
- ▶ askquestion, askokcancel, askyesno
- ▶ askretrycancel

```
from tkinter import messagebox

# la fenêtre principale Tk est créée
# automatiquement si elle n'existe
# pas déjà

val = messagebox.askokcancel(
    "Thanos",
    "Supprimer 50% du disque dur ?"
)
print(val)
```



Message / dialogues / popup

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Tkinter

Structure du code

Placement widgets

Événements

Gestionnaire de fenêtre

Multifenêtre

QT

Conclusion

Code natif

Python scientifique

Interaction ponctuelle avec l'utilisateur.

Choisir d'un fichier / dossier⁶⁴

- ▶ askopenfilename et askopenfilenames
- ▶ asksaveasfile et asksaveasfilename
- ▶ askopenfile et askopenfiles
- ▶ askdirectory

[64.<http://tkinter.fdex.eu/doc/popdial.html>](http://tkinter.fdex.eu/doc/popdial.html)

Message / dialogues / popup

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Tkinter

Structure du code

Placement widgets

Événements

Gestionnaire de fenêtre

Multifenêtre

QT

Conclusion

Code natif

Python scientifique

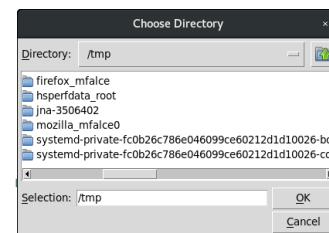
Interaction ponctuelle avec l'utilisateur.

Choisir d'un fichier / dossier⁶⁴

- ▶ askopenfilename et askopenfilenames
- ▶ asksaveasfile et asksaveasfilename
- ▶ askopenfile et askopenfiles
- ▶ askdirectory

```
from tkinter import filedialog

val = filedialog.askdirectory()
print(type(val), val) # <class
```



[64.<http://tkinter.fdex.eu/doc/popdial.html>](http://tkinter.fdex.eu/doc/popdial.html)

Message / dialogues / popup

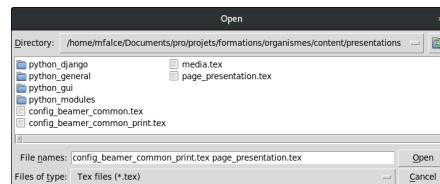
Interaction ponctuelle avec l'utilisateur.

Choisir d'un fichier / dossier⁶⁴

- ▶ askopenfilename et askopenfilenames
- ▶ asksaveasfile et asksaveasfilename
- ▶ askopenfile et askopenfiles
- ▶ askdirectory

```
from tkinter import filedialog

val = filedialog.askopenfiles(
    filetypes=[("Tex files", "*.tex"),
               ("png files", "*.png"),
               ("All files", "*")]
)
print(type(val), val)
# <class 'list'>
# [
#     <_io.TextIOWrapper name='.../config.tex' mode='r' encoding='UTF-8'>,
#     <_io.TextIOWrapper name='.../pres.tex' mode='r' encoding='UTF-8'>
# ]
```



[64.http://tkinter.fdex.eu/doc/popdial.html](http://tkinter.fdex.eu/doc/popdial.html)

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Tkinter

Structure du code

Placement widgets

Événements

Gestionnaire de fenêtre

Multifenêtre

QT

Conclusion

Code natif

Python scientifique

Fenetres secondaires

On utilise TopLevel⁶⁵ :

```
from tkinter import *

top_levels = []
def on_click():
    n = Toplevel(fenetre)
    t = str(len(top_levels))
    Button(
        master=n, text=t)
    .pack()
    top_levels.append(n)

fenetre = Tk()

bouton = Button(
    fenetre,
    command=on_click,
    text="Ouvre une fenêtre",
)
bouton.pack()

fenetre.mainloop()
```



Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Tkinter

Structure du code

Placement widgets

Événements

Gestionnaire de fenêtre

Multifenêtre

QT

Conclusion

Code natif

Python scientifique

[65.http://effbot.org/tkinterbook/toplevel.htm](http://effbot.org/tkinterbook/toplevel.htm)

Style

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Tkinter

Structure du code

Placement widgets

Événements

Gestionnaire de fenêtre

Multifenêtre

QT

Conclusion

Code natif

Python scientifique

TTK (themed Tk) : des widgets avec des styles pour ressembler à des applications natives

Bibliographie / Aller plus loin I

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Tkinter

Structure du code

Placement widgets

Événements

Gestionnaire de fenêtre

Multifenêtre

QT

Conclusion

Code natif

Python scientifique

Méthodes communes aux widgets :

<http://tkinter.fdex.eu/doc/uwm.html>

Event loop :

- ▶ <https://wiki.tcl.tk/17363>
- ▶ <https://stackoverflow.com/questions/29158220/tkinter-understanding-mainloop/29158947>

MVC :

- ▶ Article fondateur (smalltalk)
<http://www.math.sfedu.ru/smalltalk/gui/mvc.pdf>
- ▶ <https://fr.wikipedia.org/wiki/Modèle-vue-contrôleur>
- ▶ tutoriels MVC en Qt
 - ▶ <https://doc.qt.io/archives/qt-4.8/model-view-programming.html>

Bibliographie / Aller plus loin II

- ▶ <https://openclassrooms.com/fr/courses/1894236-programmez-avec-le-langage-c/1902176-larchitecture-mvc-avec-les-widgets-complexes>
- ▶ <https://www.codeguru.com/cpp/cpp/implementing-an-mvc-model-with-the-qt-c-framework.html>
- ▶ MVC en Tkinter <https://codereview.stackexchange.com/questions/163342/applying-model-view-controller-to-tkinter-matplotlib-application>

RAD : <https://github.com/alejandroautalan/pygubu>
Organisation d'un code Tkinter :

- ▶ https://python-textbok.readthedocs.io/en/1.0/Introduction_to_GUI_Programming.html

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Tkinter

Structure du code

Placement widgets

Événements

Gestionnaire de fenêtre

Multifenêtre

QT

Conclusion

Code natif

Python scientifique

Contexte

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Tkinter

Structure du code

Placement widgets

Événements

QT

Contexte

Exemples

Conclusion

Code natif

Python scientifique

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>

Contexte

- ▶ 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

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Tkinter

Structure du code

Placement widgets

Événements

QT

Contexte

Exemples

Conclusion

Code natif

Python scientifique

Qt5 / Qt6

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Tkinter

Structure du code

Placement widgets

Événements

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/.](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

Interface graphiques

Tkinter

Structure du code

Placement widgets

Événements

QT

Contexte

Exemples

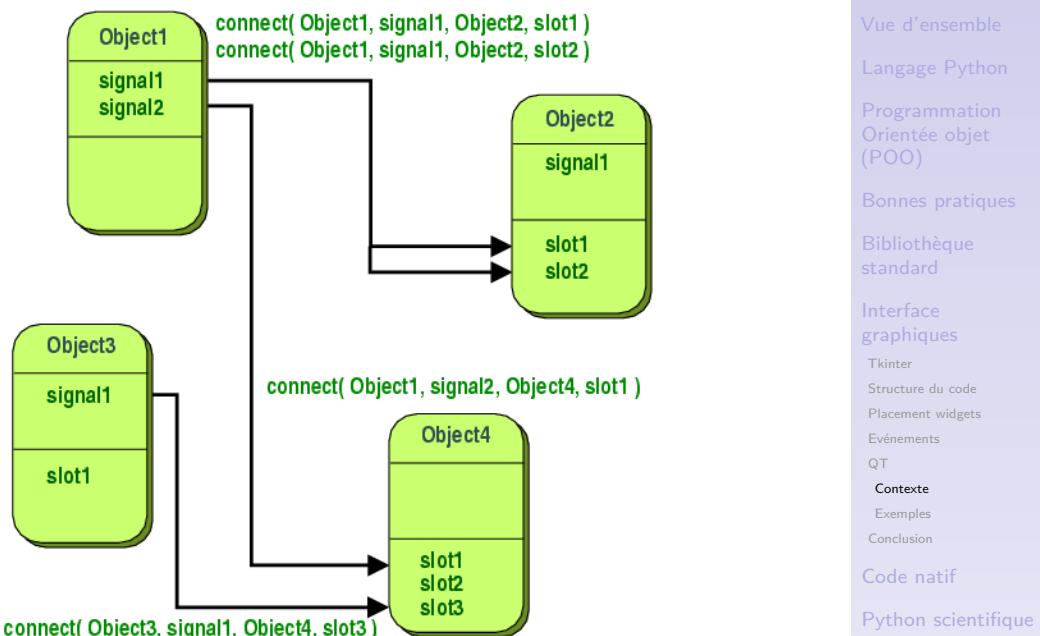
Conclusion

Code natif

Python scientifique

Signaux et slots

Matthieu Falce



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

Exemples de code

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Tkinter

Structure du code

Placement widgets

Événements

QT

Contexte

Exemples

Conclusion

Code natif

Python scientifique

Des ressources peuvent se trouver ici :

- ▶ <https://github.com/pyqt/examples>
- ▶ <https://www.pythonguis.com/tutorials/pyqt-signals-slots-events/>

Exemples de code

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Tkinter

Structure du code

Placement widgets

Événements

QT

Contexte

Exemples

Conclusion

Code natif

Python scientifique

```
# 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()
```

Exemples de code

```
# 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()
```

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Tkinter
Structure du code
Placement widgets
Événements
QT

Contexte
Exemples
Conclusion

Code natif

Python scientifique

Elements à considérer

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Tkinter
Structure du code
Placement widgets
Événements
QT
Conclusion
Choix du framework
Autres bibliothèques

Code natif

Python scientifique

Comparaison

	Qt	Tkinter
Avantages	<ul style="list-style-type: none">* 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	<ul style="list-style-type: none">* Disponible de base en python sans rien installer* Facile à prendre en main
Inconvénients	<ul style="list-style-type: none">* Complex (POQ, il faut chercher la documentation pour le C++)* Mécanisme de licence compliqué quand on ne fait pas de l'open source* Doit être installé	<ul style="list-style-type: none">* Pas de widgets avancés (un tableau par exemple)* Intégration au style de l'OS compliquée* Gestion de la complexité compliquée

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

Listing

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)

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Tkinter

Structure du code

Placement widgets

Événements

QT

Conclusion

Choix du framework

Autres bibliothèques

Code natif

Python scientifique

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Tkinter

Structure du code

Placement widgets

Événements

QT

Conclusion

Choix du framework

Autres bibliothèques

Code natif

Python scientifique

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Ctypes

Cython

Embarquer du code Python
dans du C

Bibliographie

Python scientifique

Code natif

Ctypes

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){
    sprintf(res, size-1, "Hello %s !\n", name);
}
```

test2.c

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

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Ctypes

Cython

Embarquer du code Python
dans du C

Bibliographie

Python scientifique

Ctypes

Matthieu Falce

Makefile :

```
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

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

Interface
graphiques

Code natif

Ctypes

Cython

Embarquer du code Python
dans du C

Bibliographie

Python scientifique

Ctypes

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;
}
```

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Ctypes

Cython

Embarquer du code Python
dans du C

Bibliographie

Python scientifique

Ctypes

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()
```

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

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

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.

Cython

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

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

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;
}
```

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Ctypes

Cython

Embarquer du code Python
dans du C

Bibliographie

Python scientifique

Cython

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)
```

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Ctypes

Cython

Embarquer du code Python
dans du C

Bibliographie

Python scientifique

Cython

Matthieu Falce

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

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

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:     http://docs.cython.org/en/latest/src/tutorial/strings.html
+12:
+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:     )
```

Explications

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

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

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

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

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



- ▶ 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`
 - ▶ LDFLAGS : 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

Interface graphiques

Code natif

Ctypes

Cython

Embarquer du code Python dans du C

Bibliographie

Python scientifique

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://sametmax.com/appeler-du-code-c-depuis-python-avec-ctypes/>
 - ▶ 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

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>
- ▶ <https://realpython.com/python-bindings-overview/>
- ▶ <https://realpython.com/build-python-c-extension-module/>
- ▶ 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

Interface graphiques

Code natif

Ctypes

Cython

Embarquer du code Python dans du C

Bibliographie

Python scientifique

Python scientifique

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Performances

Dask

Matériel de mesure

Bibliographie

Écosystème

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

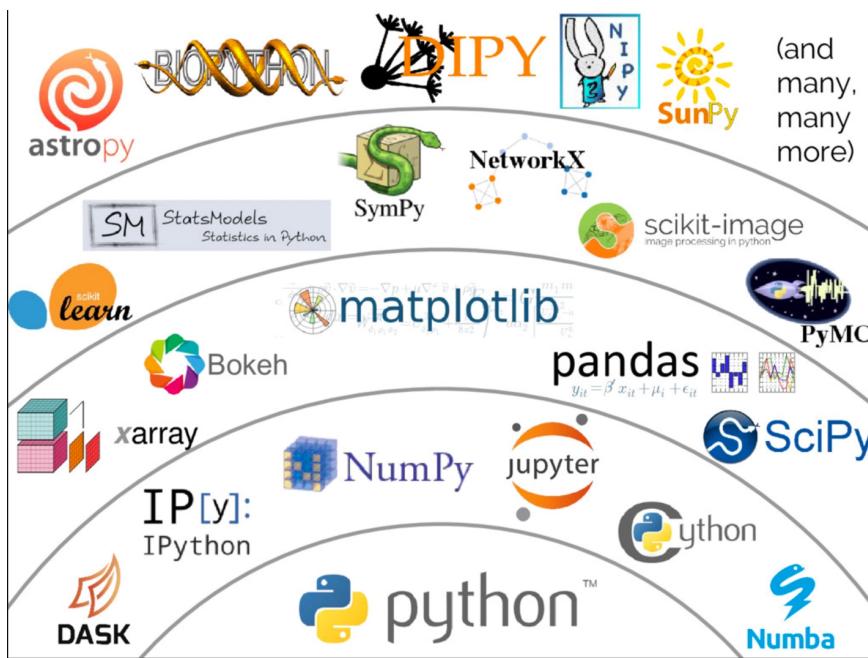
Performances

Dask

Matériel de mesure

Bibliographie

Écosystème



Source : <https://www.datacamp.com/community/blog/python-scientific-computing-case>

Matthieu Falce

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Bibliothèque standard
Interface graphiques
Code natif
Python scientifique
Écosystème scientifique
Jupyter
NumPy
Scipy
Pandas
Sympy
Analyse de réseaux
Matlab
Machine learning / statistiques
Graphiques
Performances
Dask
Matériel de mesure
Bibliographie

Écosystème

Calculs :

- ▶ numpy⁶⁶
- ▶ scipy⁶⁷
- ▶ pandas⁶⁸
- ▶ ...

Plotting :

- ▶ matplotlib⁶⁹
- ▶ seaborn⁷⁰
- ▶ bokeh⁷¹
- ▶ ...

66.<http://www.numpy.org/>

67.<https://www.scipy.org/>

68.<https://pandas.pydata.org/>

69.<https://matplotlib.org/>

70.<https://seaborn.pydata.org/>

71.<https://bokeh.pydata.org/en/latest/>

Matthieu Falce

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Bibliothèque standard
Interface graphiques
Code natif
Python scientifique
Écosystème scientifique
Jupyter
NumPy
Scipy
Pandas
Sympy
Analyse de réseaux
Matlab
Machine learning / statistiques
Graphiques
Performances
Dask
Matériel de mesure
Bibliographie

Python scientifique

```
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))
```

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Performances

Dask

Matériel de mesure

Bibliographie

IPython – Jupyter

- ▶ 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
- ▶ ...

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Performances

Dask

Matériel de mesure

Bibliographie



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, ...)

Utilise des bibliothèques Fortran ou C/C++ → bonnes performances

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

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

Analyse de réseaux

Matlab

Tableaux numpy

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

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

Analyse de réseaux

Matlab

Même manipulation que Matlab

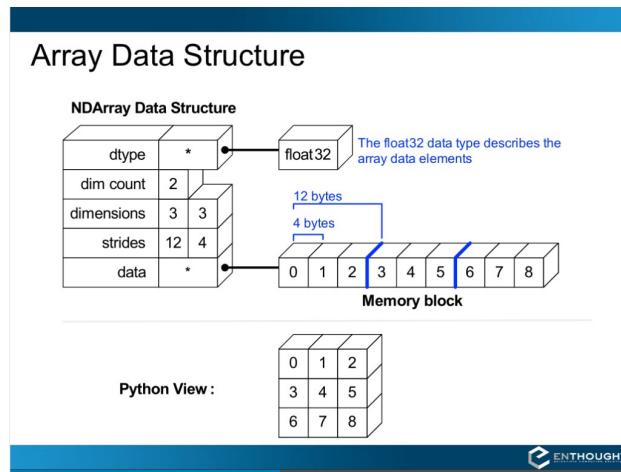
```
import numpy as np
```

```
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))
```

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

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

Analyse de réseaux

Matlab

Structure

- ▶ 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>

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

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

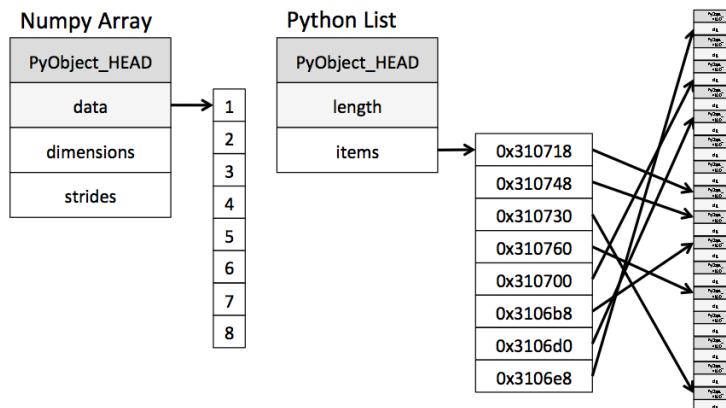
Sympy

Analyse de réseaux

Matlab

Structure

- ▶ 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>

Création des tableaux

```
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)
```

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

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

Analyse de réseaux

Matlab

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

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

Analyse de réseaux

Matlab

Création des tableaux

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

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

Analyse de réseaux

Matlab

Creation

Code

```
Z = zeros(9)
```

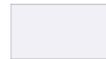
Result



Code

```
Z = zeros((5,9))
```

Result



Code

```
Z = ones(9)
```



Code

```
Z = ones((5,9))
```



Code

```
Z = array([0,0,0,0,0,0,0,0,0])
```



Code

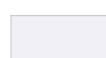
```
Z = array([0,0,0,0,0,0,0,0,0],
```

```
[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]])
```



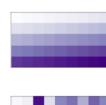
Code

```
Z = arange(9)
```



Code

```
Z = arange(5*9).reshape(5,9)
```



Code

```
Z = random.uniform(0,1,9)
```



Code

```
Z = random.uniform(0,1,(5,9))
```



<http://www.labri.fr/perso/nrougier/teaching/numpy/numarray.html>

Changements de forme / dimensions

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Présentation

Tableaux

Structure des tableaux

Création des tableaux

Manipulations usuelles

Broadcasting

Fonctions universelles

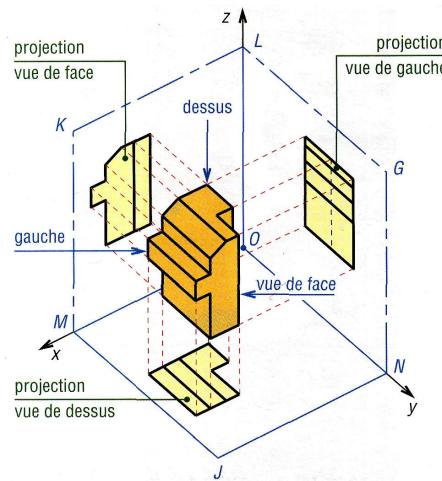
Scipy

Pandas

Sympy

Analyse de réseaux

Matlab



http://www.zpag.net/Tecnologies_Industrielles/projections_orthogonales_normalis.htm

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

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

Analyse de réseaux

Matlab

Changements de forme / dimensions

Matthieu Falce

Reshaping

Code

```
| z[2,2] = 1
```



```
| z = z.reshape(4,3)
```



```
| z = z.reshape(6,2)
```



```
| z = z.reshape(2,6)
```



Code

```
| z = z.reshape(1,12)
```



```
| z = z.reshape(12,1)
```



<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 réassigntion 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

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

Analyse de réseaux

Matlab

Indexing

Matthieu Falce

Slicing

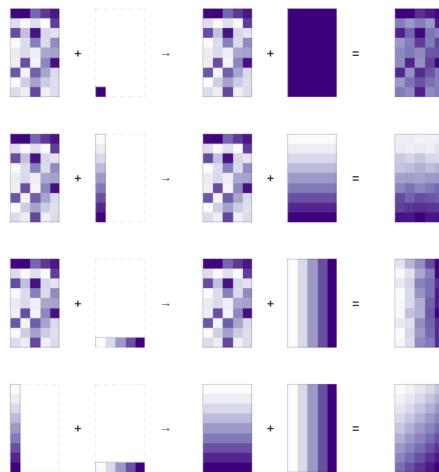
Code	Result	Code	Result
z		z[...]=1	
z[1,1]=1		z[:,0]=1	
z[0,:]=1		z[2:,2:]=1	
z[:,::2]=1		z[::2,:]=1	
z[:-2,:,:2]=1		z[2:4,2:4]=1	
z[:,2,:,:2]=1		z[3::2,3::2]=1	

<http://www.labri.fr/perso/nrougier/teaching/numpy/numpy.html>

Broadcasting

Le broadcasting permet de manipuler entre eux des tableaux de tailles différentes.

Broadcasting



<http://www.labri.fr/perso/nrougier/teaching/numpy/numpy.html>

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

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

Analyse de réseaux

Matlab

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

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

Analyse de réseaux

Matlab

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

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

Analyse de réseaux

Matlab

Présentation

Matthieu Falce

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 Scipy⁷²

Scipy utilise LAPACK, Numpy pas forcément⁷³

72.<https://docs.scipy.org/doc/numpy/reference/routines.dual.html>

73.<https://www.scipy.org/scipylib/faq.html#what-is-the-difference-between-numpy-and-scipy>

Scikits

Matthieu Falce

SciPy Toolkits → extensions pour Scipy indépendantes

Liste non exhaustive 74

- ▶ scikit-learn
- ▶ scikit-image
- ▶ scikit-bio
- ▶ ...

74.liste complète : <https://scikits.appspot.com/scikits>

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Présentation

Scikits

Fonctionnalités

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Performances

Dask

FFT 75

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Présentation

Scikits

Fonctionnalités

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Performances

Dask

```
import numpy as np
import matplotlib.pyplot as plt

t = np.arange(256)
sig = np.sin(t)

# sp est un tableau de complexes
sp1 = np.fft.fft(sig)
freq1 = np.fft.fftfreq(t.shape[-1])
plt.plot(freq1, sp1.real, freq1, sp1.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, sp1.real, label="F1")
plt.legend()
plt.show()
```

75.<https://docs.scipy.org/doc/numpy/reference/generated/numpy.fft.fft.html#numpy.fft.fft>

Optimisation 76

```
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])
```

76.<https://docs.scipy.org/doc/scipy/reference/tutorial/optimize.html>

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Présentation

Scikits

Fonctionnalités

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Performances

Dask

Optimisation 76

```
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.

76.<https://docs.scipy.org/doc/scipy/reference/tutorial/optimize.html>

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Présentation

Scikits

Fonctionnalités

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Performances

Dask

Interpolation 77

Matthieu Falce

- ▶ 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, 'o', xnew, f(xnew), '--', xnew, f2(xnew), '---')
plt.legend(['data', 'linear', 'cubic'], loc='best')
plt.show()
```

77.<https://docs.scipy.org/doc/scipy/reference/tutorial/interpolate.html>

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Présentation

Scikits

Fonctionnalités

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Performances

Dask

Intégration – équations différentielles 78 79

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Présentation

Scikits

Fonctionnalités

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Performances

Dask

78.<https://docs.scipy.org/doc/scipy/reference/tutorial/integrate.html>

79.<https://docs.scipy.org/doc/scipy/reference/integrate.html>

Intégration – équations différentielles 78 79

```
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)
```

78.<https://docs.scipy.org/doc/scipy/reference/tutorial/integrate.html>
79.<https://docs.scipy.org/doc/scipy/reference/integrate.html>

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Présentation

Scikits

Fonctionnalités

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Performances

Dask

Intégration – équations différentielles 78 79

```
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()
```

78.<https://docs.scipy.org/doc/scipy/reference/tutorial/integrate.html>
79.<https://docs.scipy.org/doc/scipy/reference/integrate.html>

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Présentation

Scikits

Fonctionnalités

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Performances

Dask

Traitement signal⁸¹

“Tout” pour le traitement du signal :⁸⁰

- ▶ convolutions
- ▶ conceptions / analyses de filtres (FIR, FII) / analyse réponse
- ▶ analyses de spectres
- ▶ détections de pics

80.<https://docs.scipy.org/doc/scipy/reference/signal.html#module-scipy.signal>

81.<https://docs.scipy.org/doc/scipy/reference/tutorial/signal.html>

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Présentation

Scikits

Fonctionnalités

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Performances

Dask

Algèbre linéaire⁸²

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Présentation

Scikits

Fonctionnalités

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Performances

Dask



Fonctions potentiellement légèrement différentes de celles de
Numpy
Compilées avec BLAS et LAPACK

82.<https://docs.scipy.org/doc/scipy/reference/tutorial/linalg.html>

Matrices creuses 83 84

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...)

83.<https://docs.scipy.org/doc/scipy-0.14.0/reference/sparse.html>

84.<https://docs.scipy.org/doc/scipy/reference/tutorial/arspack.html>

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Présentation

Scikits

Fonctionnalités

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Performances

Dask

Lois statistiques 85 86

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Présentation

Scikits

Fonctionnalités

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Performances

Dask

85.<https://docs.scipy.org/doc/scipy/reference/tutorial/linalg.html>

86.<https://docs.scipy.org/doc/scipy/reference/stats.html#module-scipy.stats>

Manipulation d'images

Les images (rasterisées ou pixelisées) sont des tableaux *numpy* classiques.

Voici les principales bibliothèques pour en faire :

- ▶ *Pillow*⁸⁷ : plutôt utilisée pour les manipulations classiques (redimensionnement, rotation, changement de format, ...)
- ▶ *numpy / scipy* classique⁸⁸ : permet de manipuler les tableaux de données de pixels directement
- ▶ *scikit image*^{89 90} : Propose de nombreux algorithmes pour faire du traitement d'images
- ▶ *openCV*⁹¹ : le port de la Bibliothèque d'analyse d'image openCV. Permet de travailler en temps réel (à partir d'une caméra par exemple)

87.<https://pillow.readthedocs.io/en/stable/>

88.Plus d'infos ici : http://scipy-lectures.org/advanced/image_processing/

89.<https://scikit-image.org/>

90.<https://scipy-lectures.org/packages/scikit-image/index.html>

91.https://docs.opencv.org/master/d6/d00/tutorial_py_root.html

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Présentation

Scikits

Fonctionnalités

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Performances

Dask

Manipulation d'images

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Présentation

Scikits

Fonctionnalités

Pandas

Sympy

Analyse de réseaux

Matlab

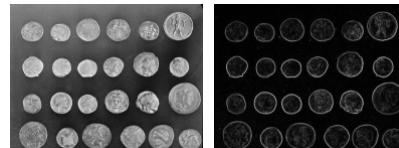
Machine learning /
statistiques

Graphiques

Performances

Dask

```
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/>

Manipulation d'images

```
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/>

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique
Jupyter
NumPy
Scipy
Présentation
SciKits
Fonctionnalités
Pandas
Sympy
Analyse de réseaux
Matlab
Machine learning /
statistiques
Graphiques
Performances
Dask

Présentation

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique
Jupyter
NumPy
Scipy
Pandas
Présentation
Dataframes
Manipulations de
dataframes
Sympy
Analyse de réseaux
Matlab
Machine learning /
statistiques
Graphiques
Performances

Bibliothèque essentielle à l'analyse de données.

Données structurées (lignes / colonnes à la SQL) et séries
temporelles.

Dataframes et séries

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Présentation

Dataframes

Manipulations de
dataframes

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Performances

- ▶ **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)

Manipulations de dataframes

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Présentation

Dataframes

Manipulations de
dataframes

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Performances

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()
```

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

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Présentation

Dataframes

Manipulations de
dataframes

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Performances

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

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Présentation

Dataframes

Manipulations de
dataframes

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Performances

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

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Présentation

Dataframes

Manipulations de
dataframes

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Performances

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

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Présentation

Dataframes

Manipulations de
dataframes

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Performances

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

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Présentation

Dataframes

Manipulations de
dataframes

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Performances

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#strptime-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

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Présentation

Dataframes

Manipulations de
dataframes

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Performances

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>

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/>

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Présentation

Dataframes

Manipulations de dataframes

Sympy

Analyse de réseaux

Matlab

Machine learning / statistiques

Graphiques

Performances

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>

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

© Matt Harasymczuk, 2020, CC-BY-SA-4.0

Source : <https://python.astrotech.io/pandas/dataframe/loc.html>

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Présentation

Dataframes

Manipulations de dataframes

Sympy

Analyse de réseaux

Matlab

Machine learning / statistiques

Graphiques

Performances

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>

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']

© Matt Harasymczuk, 2020, CC-BY-SA-4.0

Source : <https://python.astrotech.io/pandas/dataframe/loc.html>

Présentation

Sympy⁸⁷ est une bibliothèque permettant le calcul symbolique :

- ▶ simplification d'expression
- ▶ analyse (dérivation, intégration)
- ▶ affichage des sorties en LATEX

87. <http://www.sympy.org>

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Présentation

Dataframes

Manipulations de dataframes

Sympy

Analyse de réseaux

Matlab

Machine learning / statistiques

Graphiques

Performances

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning / statistiques

Graphiques

Performances

Dask

Matériel de mesure

Bibliographie

Calcul symbolique

```
from sympy import *
# permet l'affichage des formules
init_session()

# on déclare des variables
x, a, b = symbols("x a b")

# on défini 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)
```

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Performances

Dask

Matériel de mesure

Bibliographie

Présentation

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Performances

Dask

Matériel de mesure

Bibliographie

Il existe plusieurs bibliothèques pour manipuler des graphes (réseaux) en python :

- ▶ [igraph](http://igraph.org)⁸⁸
- ▶ [networkx](http://networkx.github.io)⁸⁹
- ▶ [graph-tool](http://graph-tool.skewed.de)⁹⁰

La plus connue est networkX mais peut être lente (codée en pur python), les autres sont potentiellement plus rapide.

88.<http://igraph.org>

89.<http://networkx.github.io>

90.<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

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Performances

Dask

Matériel de mesure

Bibliographie

Concept

Matthieu Falce

Matlab est un outil dédié à l'analyse mathématique / calcul numérique / simulation.

De nos jours, l'utilisation de cet outil est concurrencée par celle de python :

- ▶ outil open source et gratuit (sans licence à renouveler)
- ▶ familiarité des équipes avec python
- ▶ "réel" langage de programmation (aller au delà des maths)
- ▶ concurrence de la pile scientifique de python qui "clone" matlab

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Matlab depuis Python

Python depuis Matlab

Echange de données

Machine learning /
statistiques

Graphiques

Performances

Python ⇔ Matlab

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Matlab

Matlab depuis Python

Python depuis Matlab

Echange de données

Machine learning /
statistiques

Graphiques

Performances

Il est possible de communiquer entre python et matlab.

Plusieurs mécanismes existent :

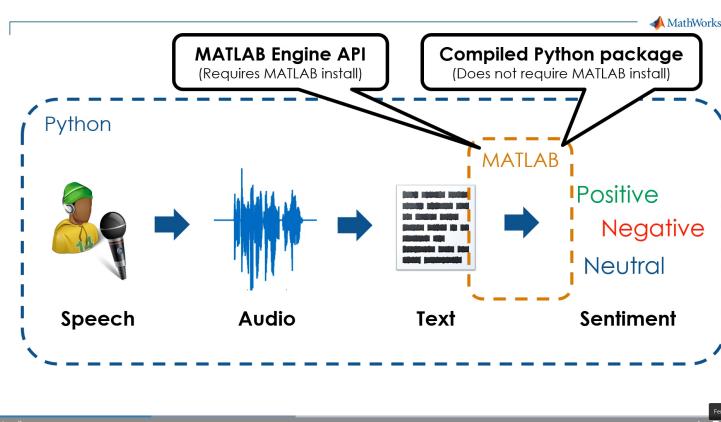
- ▶ appeler du code matlab depuis python
 - ▶ partager un moteur matlab en python (matlab engine api)
 - ▶ créer un module standalone matlab
- ▶ appeler du code python depuis matlab
- ▶ échanger les données entre les deux (commencer un traitement dans un langage et le finir dans l'autre)

Ils sont décrits ici : <https://fr.mathworks.com/products/matlab/matlab-and-python.html>

Appeler du code matlab depuis python

Matthieu Falce

Vous pouvez appeler des fonctions matlab depuis python directement :



Source: [https://fr.mathworks.com/support/search.html/videos/how-to-call-matlab-from-python-1571136879916.html?fq\[\]=%20asset_type_name%3Avideo&fq\[\]=%20category%3Amatlab/matlab-engine-for-python](https://fr.mathworks.com/support/search.html/videos/how-to-call-matlab-from-python-1571136879916.html?fq[]=%20asset_type_name%3Avideo&fq[]=%20category%3Amatlab/matlab-engine-for-python)

Appeler du code matlab depuis python

Vous pouvez appeler des fonctions matlab depuis python directement :

Why Call MATLAB from Python?

Already working in Python, and

- Want to reuse existing MATLAB code
- Need functionality that is only available in MATLAB
- Want to share MATLAB functionality with Python users who do not have MATLAB installed

Source: [https://fr.mathworks.com/support/search.html/videos/how-to-call-matlab-from-python-1571136879916.html?fq\[\]=%asset_type_name%video&fq\[\]=%category%matlab/matlab-engine-for-python](https://fr.mathworks.com/support/search.html/videos/how-to-call-matlab-from-python-1571136879916.html?fq[]=%asset_type_name%video&fq[]=%category%matlab/matlab-engine-for-python)

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Matlab

Matlab depuis Python

Python depuis Matlab

Echange de données

Machine learning / statistiques

Graphiques

Performances

Appeler du code matlab depuis python – Matlab Engine

- ▶ avoir une installation de matlab valide
- ▶ installer l'engine (au choix) :
https://fr.mathworks.com/help/matlab/matlab_external/install-the-matlab-engine-for-python.html
 - ▶ pip install matlabengine (télécharge depuis internet)
 - ▶ cd "matlabroot/extern/engines/python" puis python -m pip install . (utilise la version locale à matlab)
- ▶ gestion des tableaux de nombres :
https://fr.mathworks.com/help/matlab/matlab_external/matlab-arrays-as-python-variables.html

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Matlab

Matlab depuis Python

Python depuis Matlab

Echange de données

Machine learning / statistiques

Graphiques

Performances

Appeler du code matlab depuis python – Matlab Engine

```
# il faut installer le module matlabengine
import matlab.engine
import numpy

% source :
% https://fr.mathworks.com/
% help/matlab/ref/function.html

% une fonction qui retourne 2 résultats
function [m, s] = matlab_stat_exemple(x)
    n = length(x)
    m = sum(x)
    s = sqrt(sum(x-m).^2/n)
end

# en utilisant les fonctions de numpy
moy, std = eng.matlab_stat_exemple(
    numpy.cos(numpy.arange(3)), nargout=2
)

# en utilisant les fonctions de matlab
moy2, std2 = eng.matlab_stat_exemple(
    eng.cos(matlab.double([0, 1, 2])), nargout=2
)

assert moy == moy2
assert std == std2
```

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Matlab

Matlab depuis Python

Python depuis Matlab

Echange de données

Machine learning /
statistiques

Graphiques

Performances

Appeler du code matlab depuis python – Module compilé

- ▶ il faut avoir installé matlab compiler sdk (ce qui n'est pas mon cas)
- ▶ le compiler va créer un paquet python que l'on peut importer et utiliser comme n'importe quel paquet
- ▶ j'imagine que l'on peut aussi créer une .dll en C que l'on peut appeler en python
- ▶ intérêt : ne nécessite pas d'avoir une licence matlab sur les ordinateurs qui utilisent le module

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Matlab

Matlab depuis Python

Python depuis Matlab

Echange de données

Machine learning /
statistiques

Graphiques

Performances

Appeler du code python depuis matlab

Matthieu Falce

- ▶ il est possible d'appeler des fonctions, objets ou modules python depuis matlab
- ▶ cela permet de travailler intégralement dans matlab
- ▶ cela permet d'étendre les capacités de matlab
- ▶ fonctionne depuis la version R2019b (<https://fr.mathworks.com/help/matlab/ref/pyenv.html>)
- ▶ il est possible de choisir l'interpréteur (ou environnement virtuel) que l'on veut : `pyenv('Version','executable')` (https://fr.mathworks.com/help/matlab/matlab_external/install-supported-python-implementation.html)

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Matlab

Matlab depuis Python

Python depuis Matlab

Echange de données

Machine learning /
statistiques

Graphiques

Performances

Appeler du code python depuis matlab

Matthieu Falce

Il faut faire attention à :

- ▶ bien savoir où sont les variables (dans python ou dans matlab)
- ▶ bien gérer le PYTHONPATH pour les imports
- ▶ bien convertir les variables dans matlab pour les réutiliser
 - ▶ matlab vers python :
https://fr.mathworks.com/help/matlab/matlab_external/passing-data-to-python.html
 - ▶ matlab depuis python :
https://fr.mathworks.com/help/matlab/matlab_external/pass-data-to-matlab-from-python.html

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Matlab

Matlab depuis Python

Python depuis Matlab

Echange de données

Machine learning /
statistiques

Graphiques

Performances

Appeler du code python depuis matlab

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Matlab

Matlab depuis Python

Python depuis Matlab

Echange de données

Machine learning /
statistiques

Graphiques

Performances

On peut appeler des fonctions python en les préfixant par py.

```
% on peut appeler une fonction en préfixant avec py
py.sum([1, 2])

% on peut convertir les variables complexes pour les utiliser en matlab
a = py.numpy.arange(10)
cos(double(a))

% on peut executer du code python et dire quelles sont les variables
min = 10
max = 20
pyrun("import random; a = random.randint(min, max)", "a", min=min, max=max)
```

Appeler du code python depuis matlab

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Matlab

Matlab depuis Python

Python depuis Matlab

Echange de données

Machine learning /
statistiques

Graphiques

Performances

```
# fichier exemple.py
# importé dans matlab
import math

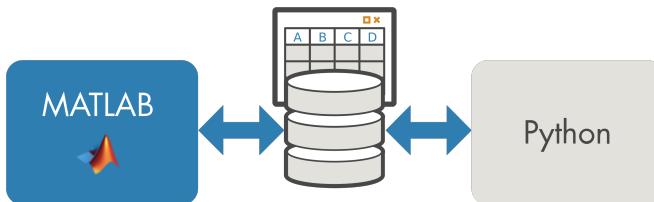
print(math.cos(2))

def f(a: list):
    return [math.cos(i) for i in a]
```

Echanger des données

Matthieu Falce

Vous pouvez échanger des données entre python et matlab :



Source: <https://fr.mathworks.com/products/matlab/matlab-and-python.html>

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Matlab

Matlab depuis Python

Python depuis Matlab

Echange de données

Machine learning / statistiques

Graphiques

Performances

Echanger des données

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Matlab

Matlab depuis Python

Python depuis Matlab

Echange de données

Machine learning / statistiques

Graphiques

Performances

- ▶ il est possible d'ouvrir les .mat en python
 - ▶ utiliser la fonction `scipy.io.loadmat`⁹¹
 - ▶ le format mat n'est pas totalement documenté, donc il peut y avoir des surprises
 - ▶ aucune action particulière à faire en matlab
- ▶ utiliser des formats d'échanges classiques
 - ▶ CSV
 - ▶ parquet / arrow (format de données tabulaires optimisé pour les grandes quantités de données)

91.<https://docs.scipy.org/doc/scipy/reference/generated/scipy.io.loadmat.html>

Echanger des données

Lecture / écriture de fichiers parquet en python (on utilise pyarrow)

```
# écrire un objet parquet à partir d'un dataframe
import numpy as np
import pandas as pd
import pyarrow as pa
import pyarrow.parquet as pq

df = pd.DataFrame({
    "one": [-1, np.nan, 2.5],
    "two": ["foo", "bar", "baz"],
    "three": [True, False, True],
}, index=list("abc"))
table = pa.Table.from_pandas(df)

# créer un fichier parquet à partir d'un objet parquet
pq.write_table(table, "example.parquet")

# lire un fichier parquet dans un dataframe pandas
table2 = pq.read_table("example.parquet")
table2.to_pandas()
```

Source:

<https://arrow.apache.org/docs/python/parquet.html>

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Matlab

Matlab depuis Python

Python depuis Matlab

Echange de données

Machine learning /
statistiques

Graphiques

Performances

Echanger des données

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Matlab

Matlab depuis Python

Python depuis Matlab

Echange de données

Machine learning /
statistiques

Graphiques

Performances

Sources:

- ▶ <https://fr.mathworks.com/help/matlab/ref/parquetwrite.html>
- ▶ <https://fr.mathworks.com/help/matlab/ref/parquetread.html>

Etat des lieux

Matthieu Falce

Il existe plusieurs bibliothèques dédiées apprentissage :

- ▶ modèles statistiques (régressions, tests statistiques, méthodes sur séries temporelles) : `statsmodels`⁹¹
- ▶ algorithmes de *machine learning* : `scikit-learn`⁹²
- ▶ *curve feating* : `prophet`⁹³ (analyse de séries temporelles)
- ▶ *deep learning* : `tensorflow`⁹⁴, `keras`⁹⁵

91.<https://www.statsmodels.org/stable/index.html>

92.<https://scikit-learn.org/stable/>

93.<https://facebook.github.io/prophet/>

94.<https://www.tensorflow.org/>

95.<https://keras.io/>

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Performances

Dask

Matériel de mesure

Bibliographie

Graphiques

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Matplotlib

Personnalisation

Seaborn

Concurrents 2D

Graphiques 3D

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

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Matplotlib

Personnalisation

Seaborn

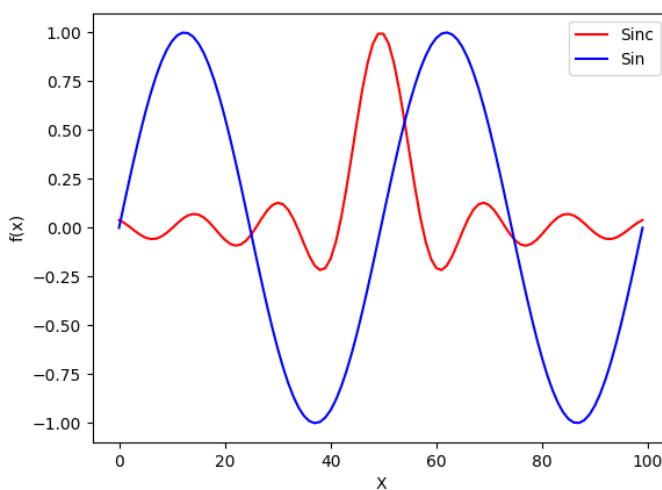
Concurrents 2D

Graphiques 3D

Matplotlib

Bibliothèques de plot la plus célèbre

API inspirée de Matlab
Le style aussi



Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Matplotlib

Personnalisation

Seaborn

Concurrents 2D

Graphiques 3D

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

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Matplotlib

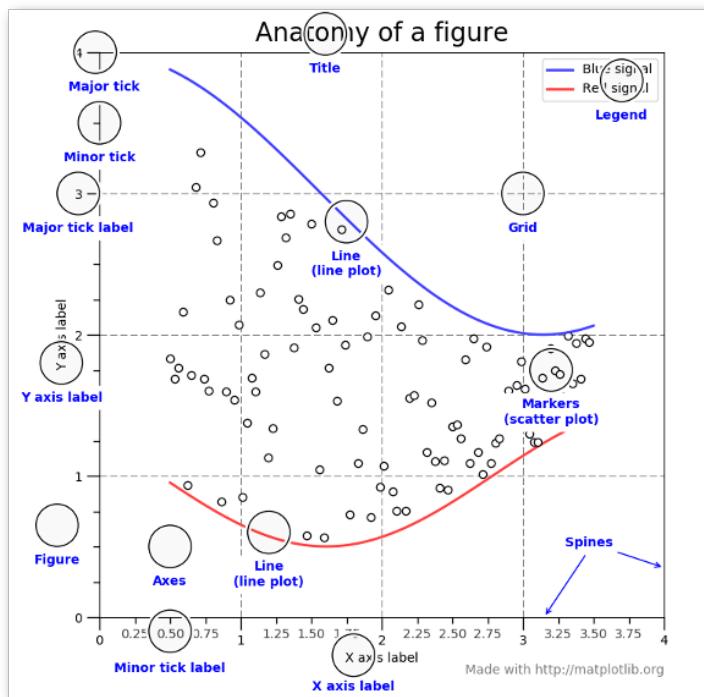
Personnalisation

Seaborn

Concurrents 2D

Graphiques 3D

Anatomie d'une figure



Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Matplotlib

Personnalisation

Seaborn

Concurrents 2D

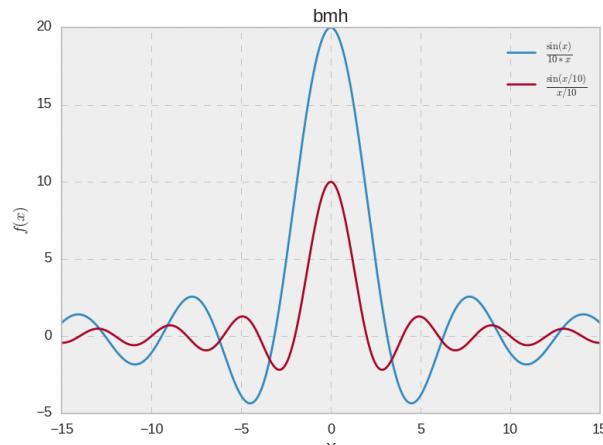
Graphiques 3D

https://matplotlib.org/faq/usage_faq.html

Personnalisation

Vous pouvez surcharger la configuration de Matplotlib :

- ▶ passer des paramètres à la fonction de dessin
- ▶ changer de style
- ▶ modifier le `pyplot.rcParams`



Thème : `bmh`

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Matplotlib

Personnalisation

Seaborn

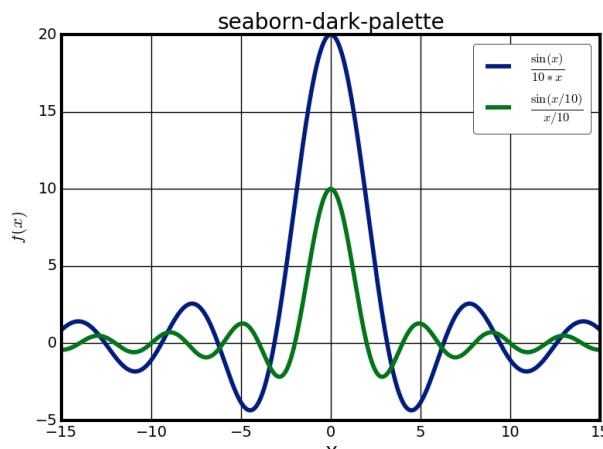
Concurrents 2D

Graphiques 3D

Personnalisation

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`

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Matplotlib

Personnalisation

Seaborn

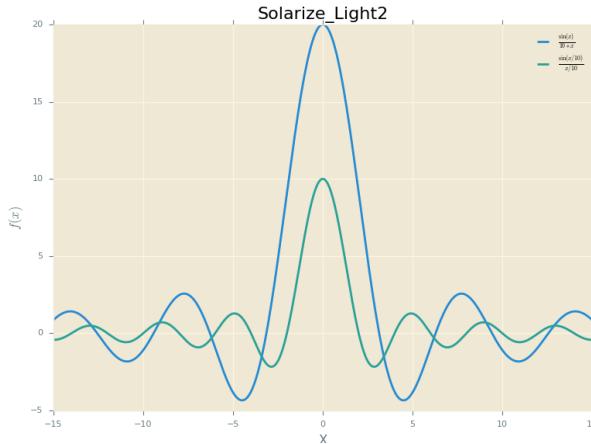
Concurrents 2D

Graphiques 3D

Personnalisation

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

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Matplotlib

Personnalisation

Seaborn

Concurrents 2D

Graphiques 3D

Personnalisation

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'] = 8
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()
```

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Matplotlib

Personnalisation

Seaborn

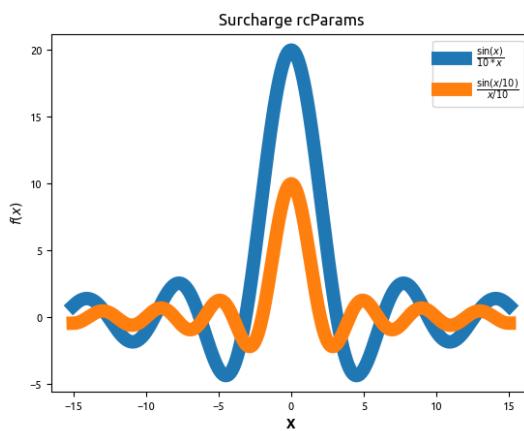
Concurrents 2D

Graphiques 3D

Personnalisation

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`

Seaborn

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")
```

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning / statistiques

Graphiques

Matplotlib

Personnalisation

Seaborn

Concurrents 2D

Graphiques 3D

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning / statistiques

Graphiques

Matplotlib

Personnalisation

Seaborn

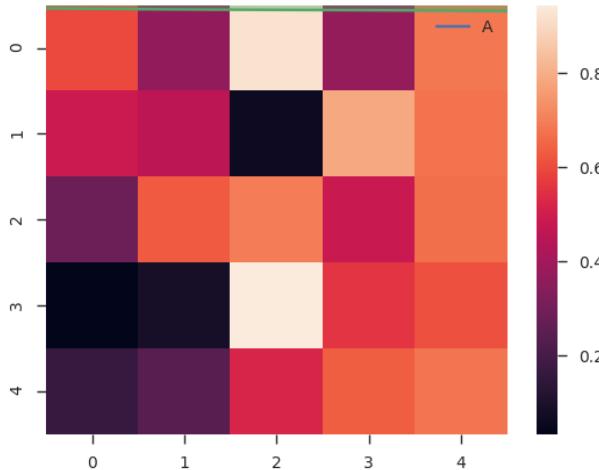
Concurrents 2D

Graphiques 3D

Seaborn

Seaborn se base sur matplotlib.

Il rajoute des styles et des fonctionnalités (surtout utile en statistiques).



Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Matplotlib

Personnalisation

Seaborn

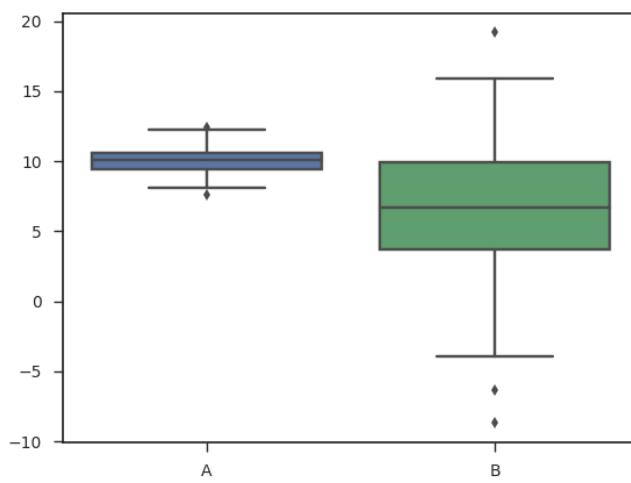
Concurrents 2D

Graphiques 3D

Seaborn

Seaborn se base sur matplotlib.

Il rajoute des styles et des fonctionnalités (surtout utile en statistiques).



Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Matplotlib

Personnalisation

Seaborn

Concurrents 2D

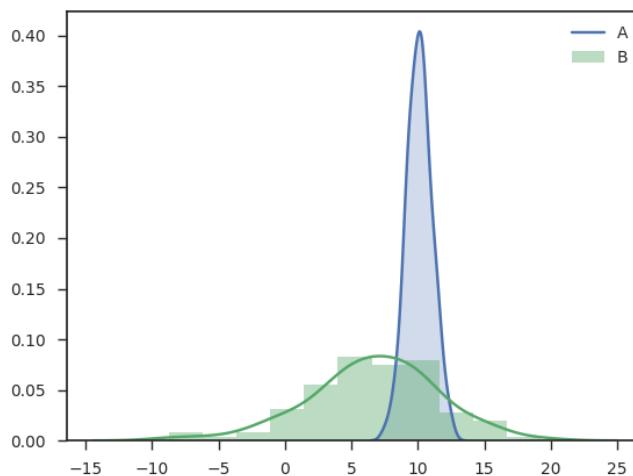
Graphiques 3D

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

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Matplotlib

Personnalisation

Seaborn

Concurrents 2D

Graphiques 3D

Concurrents 2D

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Matplotlib

Personnalisation

Seaborn

Concurrents 2D

Graphiques 3D

Concurrent : *plotly*

- ▶ 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()
```

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Matplotlib

Personnalisation

Seaborn

Concurrents 2D

Graphiques 3D

Concurrent : *plotly*

```
import plotly.graph_objects as go

fig = go.Figure(
    data=go.Scatter(
        x=[1, 2, 3, 4],
        y=[10, 11, 12, 13],
        mode="markers",
        marker=dict(
            size=[40, 60, 80, 100],
            color=[0, 1, 2, 3]),
    )
)

fig.show()
```

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Matplotlib

Personnalisation

Seaborn

Concurrents 2D

Graphiques 3D

Concurrent : *plotly*

```
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)
```

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Matplotlib

Personnalisation

Seaborn

Concurrents 2D

Graphiques 3D

Dashboards

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Matplotlib

Personnalisation

Seaborn

Concurrents 2D

Graphiques 3D

On peut utiliser Dash⁹⁶

- ▶ 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/>

⁹⁶<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

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Matplotlib

Personnalisation

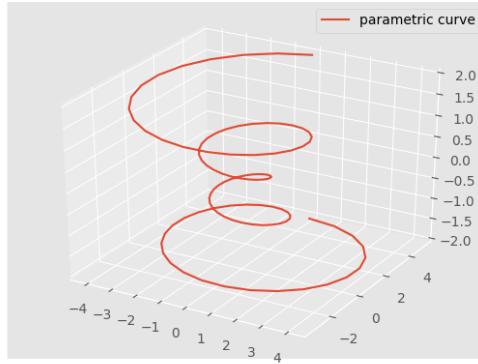
Seaborn

Concurrents 2D

Graphiques 3D

Graphiques 3D

Matthieu Falce



Résultat graphique 3D



Ce n'est pas de la "vraie" 3D... (pas de notion de volumes)

Graphiques 3D

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Matplotlib

Personnalisation

Seaborn

Concurrents 2D

Graphiques 3D

Pour faire de la vraie 3d :

- ▶ mayavi
- ▶ <https://lorensen.github.io/VTKExamples/site/Python/>
- ▶ (ParaView)
- ▶ moteurs de jeu 3D

Avant-propos

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Performances

Numexpr

Dask

Matériel de mesure

Bibliographie



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
- ▶ ...

Numexpr 97

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)
```

97.<https://github.com/pydata/numexpr>

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning / statistiques

Graphiques

Performances

Numexpr

Dask

Matériel de mesure

Bibliographie

Dask 98

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

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning / statistiques

Graphiques

Performances

Dask

Autres

Matériel de mesure

Bibliographie

98.<https://dask.org/>

Dask

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

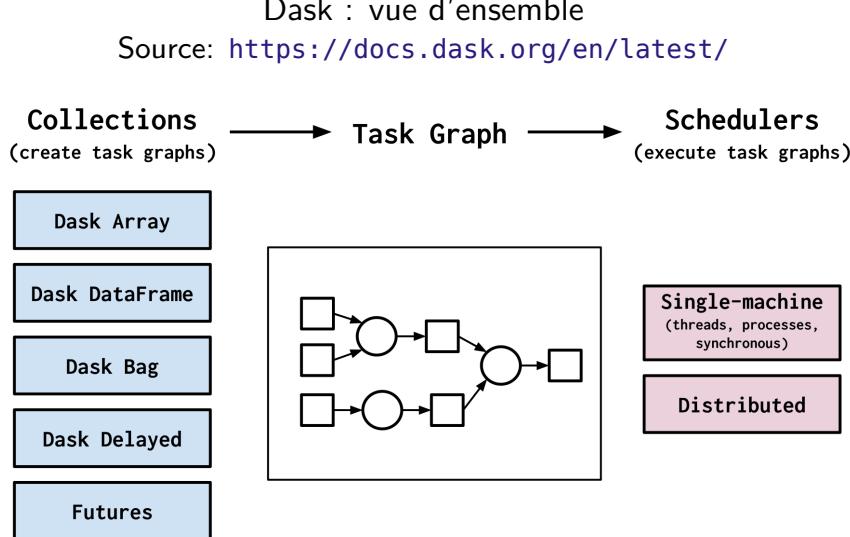
Performances

Dask

Autres

Matériel de mesure

Bibliographie



Dask

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Performances

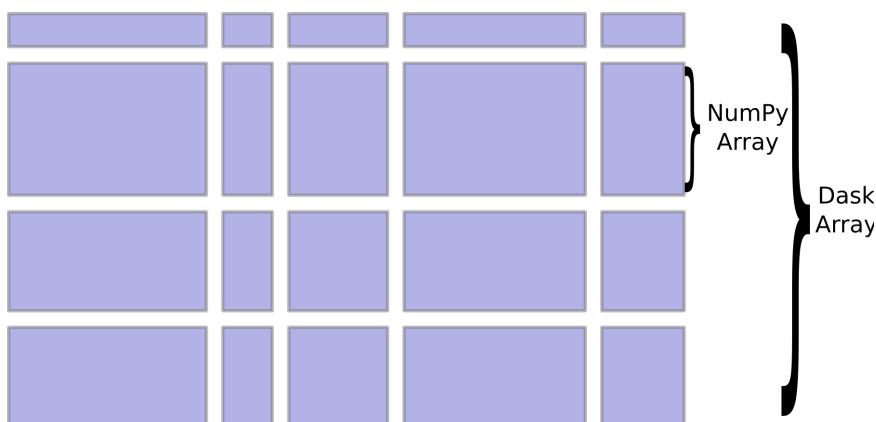
Dask

Autres

Matériel de mesure

Bibliographie

Dask : structure d'un *dask array*
Source: <https://docs.dask.org/en/latest/array.html>



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

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Performances

Dask

Autres

Matériel de mesure

Bibliographie

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

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Performances

Dask

Autres

Matériel de mesure

Bibliographie

Autres techniques

D'autres techniques, plus ou moins matures permettent d'améliorer les temps de calculs également :

- ▶ Numba
- ▶ Pythran
- ▶ (Theano)
- ▶ distribution python par Intel

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Performances

Dask

Autres

Matériel de mesure

Bibliographie

Concept

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Performances

Dask

Matériel de mesure

Concepts

Protocoles

Les matériels de mesure en électronique (oscilloscope / générateurs / ...) peuvent être contrôlés à distance.

Il existe plusieurs connectiques pour faire cela.

- ▶ GPIB
- ▶ USB
- ▶ RS232
- ▶ ethernet

Pour toute cette partie, la ressource principale utilisée est

[https:](https://goughlui.com/2021/03/28/tutorial-introduction-to-scp-i-automation-of-test-equipment-with-pyvisa/)

[//goughlui.com/2021/03/28/tutorial-introduction-to-scp-i-automation-of-test-equipment-with-pyvisa/](https://goughlui.com/2021/03/28/tutorial-introduction-to-scp-i-automation-of-test-equipment-with-pyvisa/)

Protocoles

Différents protocoles permettent de communiquer avec les appareils :

- ▶ SCPI (Standard Control of Programmable Instruments)
 - ▶ commandes ASCII envoyées aux instruments
- ▶ VISA (Virtual Instrument Software Architecture)
 - ▶ API permettant de communiquer entre un ordinateur et un instrument
 - ▶ peut être spécifique à une marque / modèle
 - ▶ son but est de faire le lien entre l'application et l'instrument en "effaçant" l'OS et les pilotes

VISA permet de s'abstraire du matériel et des conversions des commandes. Le code sera quasiment identique que l'appareil soit branché en USB ou Ethernet.

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning / statistiques

Graphiques

Performances

Dask

Matériel de mesure

Concepts

Protocoles

SCPI

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning / statistiques

Graphiques

Performances

Dask

Matériel de mesure

Concepts

Protocoles

SCPI

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Performances

Dask

Matériel de mesure

Concepts

Protocoles

Exemples :

- ▶ set command (toutes sont équivalentes)

- ▶ SOURCE:VOLTAGE 10
- ▶ SOURCE:VOLT 10
- ▶ SOUR:VOLTAGE 10
- ▶ SOUR:VOLT 10
- ▶ VOLT 10

- ▶ query command

- ▶ MEAS:VOLT?

En python

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Performances

Dask

Matériel de mesure

Concepts

Protocoles

Installer :

- ▶ pip install pyvisa

- ▶ il s'agit d'un wrapper pour VISA en python, il doit donc être installé également
- ▶ par exemple National Instruments NI-VISA (windows)
- ▶ pour linux pip install pyvisa-py pyusb pyserial

En python

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Performances

Dask

Matériel de mesure

Concepts

Protocoles

Récupérer la liste des ressources

```
import pyvisa

rm = pyvisa.ResourceManager()
print(rm.list_resources())
```

En python

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Performances

Dask

Matériel de mesure

Concepts

Protocoles

Automatiser un réglage / une lecture

```
# source : https://goughlui.com/2021/03/28/
#           tutorial-introduction-to-scp-i-automation-
#           of-test-equipment-with-pyvisa/
import pyvisa
import time

resource_manager = pyvisa.ResourceManager()
# You can change the variable name and resource name
ins_ngm202 = resource_manager.open_resource("TCPIP0::192.168.80.18::inst0::INSTR")

# On va régler la sortie à 10V / 1A
ins_ngm202.write("SOUR:VOLT 10")
ins_ngm202.write("SOUR:CURR 1")
ins_ngm202.write("OUTP 1")

# On attend 5s
time.sleep(5)

# On lit le voltage / courant
voltage, = ins_ngm202.query_ascii_values("MEAS:VOLT?")
current, = ins_ngm202.query_ascii_values("MEAS:CURR?")

# On éteint la sortie
ins_ngm202.write("OUTP 0")
print(f"Voltage was {voltage}. Current was {current}.")
ins_ngm202.close()
```

Autres ressources

- ▶ IVI : <https://github.com/python-ivi/python-ivi> / <http://alexforencich.com/wiki/en/python-ivi/readme>
- ▶ bibliothèque spécialement pour vos matériels : <https://pypi.org/project/msox3000/>
- ▶ une approche objet de SCPI : <https://github.com/bicarlsen/easy-scpi> (évite de construire les chaînes à la main)
- ▶ utilisation sans PYVISA : <https://magna-power.com/learn/kb/instrumentation-programming-with-python>

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Performances

Dask

Matériel de mesure

Concepts

Protocoles

Bibliographie I

- ▶ 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
 - ▶ <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
 - ▶ <https://docs.scipy.org/doc/numpy/reference/internal.html>

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Performances

Dask

Matériel de mesure

Bibliographie

Bibliographie II

- ▶ <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-lecosysteme-python-scientifique>

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning / statistiques

Graphiques

Performances

Dask

Matériel de mesure

Bibliographie

Bibliographie III

- ▶ **pandas**
 - ▶ la feuille de triche pandas officielle : 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
 - ▶ 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_monaco_parking
 - ▶ https://pandas.pydata.org/pandas-docs/stable/user_guide/cookbook.html#cookbook-resample

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning / statistiques

Graphiques

Performances

Dask

Matériel de mesure

Bibliographie

Bibliographie IV

► 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

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Performances

Dask

Matériel de mesure

Bibliographie

Bibliographie V

- ▶ les exemples sur scipy-lectures : <https://scipy-lectures.org/packages/scikit-image/index.html> et
http://scipy-lectures.org/advanced/image_processing/

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Bibliothèque
standard

Interface
graphiques

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Matlab

Machine learning /
statistiques

Graphiques

Performances

Dask

Matériel de mesure

Bibliographie