

Formation Python, perfectionnement (session de rattrapage)

Matthieu Falce

Décembre 2021

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Au programme I

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Au programme II

Python scientifique

Succès du langage

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

A propos de moi – Qui suis-je ?

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

- ▶ 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

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

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

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Matthieu Falce

Vue d'ensemble

Historique
Philosophie
Python, CPython, ...
Cas d'utilisations de python
Installation
Environnement de développement

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Vue d'ensemble

Un vieux langage ?

Matthieu Falce

Vue d'ensemble

Historique
Philosophie
Python, CPython, ...
Cas d'utilisations de python
Installation
Environnement de développement

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

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

Un vieux langage ?

Matthieu Falce

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



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

Vue d'ensemble

Historique
Philosophie
Python, CPython, ...
Cas d'utilisations de python
Installation
Environnement de développement

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

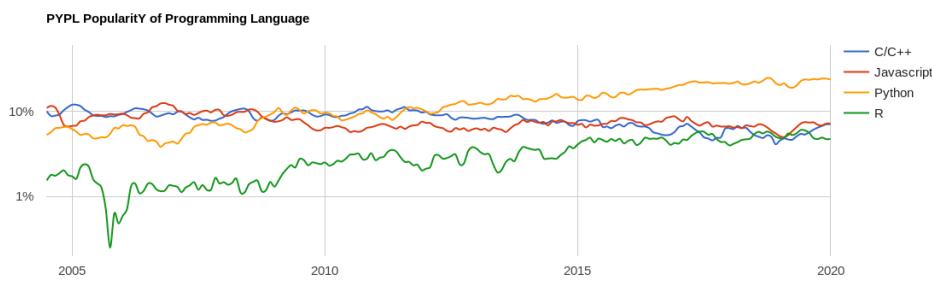
Python scientifique

Succès du langage

Un vieux langage ?

Matthieu Falce

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



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

Vue d'ensemble

Historique
Philosophie
Python, CPython, ...
Cas d'utilisations de python
Installation
Environnement de développement

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

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
Python, CPython, ...
Cas d'utilisations de python
Installation
Environnement de développement

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Origine du nom

- ▶ 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())
```

Matthieu Falce

Vue d'ensemble

Historique
Philosophie
Python, CPython, ...
Cas d'utilisations de python
Installation
Environnement de développement

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Rétrocompatibilité

Matthieu Falce

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

Un exemple de code de démo de la version 1.0.0



Vue d'ensemble

Historique
Philosophie
Python, CPython, ...
Cas d'utilisations de python
Installation
Environnement de développement

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

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)

    ...

    # Other methods like __str__, __add__, etc.
```

Vue d'ensemble

Historique
Philosophie
Python, CPython, ...
Cas d'utilisations de python
Installation
Environnement de développement

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

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
Python, CPython, ...
Cas d'utilisations de python
Installation
Environnement de développement

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

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
Python, CPython, ...
Cas d'utilisations de python
Installation
Environnement de développement

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Python 2 End Of Life

Fin du support de Python le 1er janvier 2020



Matthieu Falce

Vue d'ensemble

Historique
Philosophie
Python, CPython, ...
Cas d'utilisations de python
Installation
Environnement de développement

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

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
Python, CPython, ...
Cas d'utilisations de python
Installation
Environnement de développement

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Zen of Python

Matthieu Falce

Vue d'ensemble

Historique

Philosophie

Python, CPython, ...

Cas d'utilisations de python

Installation

Environnement de développement

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

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

Python, CPython, ...

Cas d'utilisations de python

Installation

Environnement de développement

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

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

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

C'est quoi python au final ?

Matthieu Falce

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

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

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

Vue d'ensemble

Historique

Philosophie

Python, CPython, ...

Cas d'utilisations de python

Installation

Environnement de développement

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Interpréteur embarqué dans des logiciels

Matthieu Falce

Python sert de langage de script dans de nombreux logiciels :

- ▶ blender ³
- ▶ qgis ⁴
- ▶ autodesk ⁵
- ▶ Vim ⁶
- ▶ Minecraft ⁷
- ▶ ...

Vue d'ensemble

Historique

Philosophie

Python, CPython, ...

Cas d'utilisations de python

Scripting

Exemples personnels

Installation

Environnement de développement

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

3.<https://blender.org>

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

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

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

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

Exemples personnels

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

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

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

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

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

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

Matthieu Falce

Vue d'ensemble

Historique

Philosophie

Python, CPython, ...

Cas d'utilisations de python

Scripting

Exemples personnels

Installation

Environnement de développement

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Distributions ¹³

Matthieu Falce

Vue d'ensemble

Historique

Philosophie

Python, CPython, ...

Cas d'utilisations de python

Installation

Environnement de développement

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Il existe plusieurs distributions python.

Les plus connues :

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

Pour commencer et sous Windows, je conseille anaconda

13.<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
 - ▶ sublime text 3 + anaconda (plugin ST, pas la distribution du dessus...)
 - ▶ ViM / Emacs + plugins
- ▶ IDE
 - ▶ eclipse + mode python
 - ▶ PyCharm

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

Matthieu Falce

Vue d'ensemble

Historique
Philosophie
Python, CPython, ...
Cas d'utilisations de python
Installation
Environnement de développement

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Matthieu Falce

Vue d'ensemble

Langage Python

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

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Langage Python

Passage par référence

Matthieu Falce

Vue d'ensemble

Langage Python

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage



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

Vue d'ensemble

Langage Python

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

```
# 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
```

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

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Mutabilité

Matthieu Falce

Vue d'ensemble

Langage Python

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Construction des conteneurs

Matthieu Falce

Vue d'ensemble

Langage Python

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

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
...	...

Pour les classes

Matthieu Falce

Vue d'ensemble

Langage Python

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

```
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
```

Cycle de vie

Matthieu Falce

Vue d'ensemble

Langage Python

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Copier une variable

```
import copy

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

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

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

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

Cycle de vie

Matthieu Falce

Vue d'ensemble

Langage Python

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

```
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

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

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.deque`

Vue d'ensemble

Langage Python

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

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

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

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': {'Canidae': {'Canis': {'dog': 'waf'}}, 'Felidae': {'Felis': {'cat': 'miaou'}}}}}}
```

Matthieu Falce

Vue d'ensemble

Langage Python

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

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

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Le *duck typing* ?

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>

Matthieu Falce

Vue d'ensemble

Langage Python

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Le *duck typing* ?

Matthieu Falce

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

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

Vue d'ensemble

Langage Python

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Exemple

Matthieu Falce

```
def prend_premier(conteneur):  
    return conteneur[0]  
  
def prend_premier_2(iterable):  
    for element in iterable:  
        return element  
  
print(prend_premier([1, 2]))  
print(prend_premier((1, 2)))  
print(prend_premier(open("/etc/hosts")))) # TypeError  
  
print(prend_premier_2([1, 2]))  
print(prend_premier_2((1, 2)))  
print(prend_premier_2(open("/etc/hosts"))))
```

Vue d'ensemble

Langage Python

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Les *able

Matthieu Falce

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

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

Vue d'ensemble

Langage Python

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

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Slicing

Matthieu Falce

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

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

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

Vue d'ensemble

Langage Python

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

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Slicing

Matthieu Falce

Vue d'ensemble

Langage Python

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Explication des slices

Slicing avec un seul bord

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

Slicing avec index négatif

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

Slicing avec pas

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

Objet slice

Matthieu Falce

Vue d'ensemble

Langage Python

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

```
a = [x for x in range(10)]  
  
dix_premiers = slice(0, 10)  
print(type(dix_premiers))  
  
print(a[dix_premiers])  
print(dix_premiers.start)  
print(dix_premiers.stop)  
print(dix_premiers.step)  
  
index_pairs = slice(0,None,2)  
print(a[index_pairs])  
  
slice_inverse = slice(None, None, -1)  
print(a[slice_inverse])
```

Utile pour conserver les valeurs de début, fin et pas dans un
objet précis.

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

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

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

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

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

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

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())
```

Vue d'ensemble

Langage Python

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

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

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

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation

Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

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

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation

Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Boucles – contrôles

Matthieu Falce

Vue d'ensemble

Langage Python

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

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

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

```
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

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation

Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage



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

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation

Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

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

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

Programmation Orientée objet (POO)

Bonnes pratiques
Création de modules
Programmation concurrente
Code natif
Python scientifique
Succès du langage

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

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

Programmation Orientée objet (POO)

Bonnes pratiques
Création de modules
Programmation concurrente
Code natif
Python scientifique
Succès du langage

Tests et conditions – booléens

Matthieu Falce

[Vue d'ensemble](#)

[Langage Python](#)

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

[Programmation](#)

Orientée objet
(POO)

[Bonnes pratiques](#)

[Création de modules](#)

[Programmation concurrente](#)

[Code natif](#)

[Python scientifique](#)

[Succès du langage](#)

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](#)

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

[Programmation](#)

Orientée objet
(POO)

[Bonnes pratiques](#)

[Création de modules](#)

[Programmation concurrente](#)

[Code natif](#)

[Python scientifique](#)

[Succès du langage](#)

```
# 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
```

Gestionnaires de contexte

Matthieu Falce

Vue d'ensemble

Langage Python

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

Programmation
Orientée objet
(POO)

Bonnes pratiques
Création de modules
Programmation concurrente
Code natif
Python scientifique
Succès du langage

Le besoin

```
def f1():
    foo = open("/tmp/foo", "w")
    try:
        foo.write('Salut !')
    finally:
        foo.close()

#####
import threading

def f2():
    lock = threading.Lock()
    lock.acquire()
    my_list = []
    try:
        my_list.append(1)
    finally:
        lock.release()
```

Gestionnaires de contexte

Matthieu Falce

Vue d'ensemble

Langage Python

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

Programmation
Orientée objet
(POO)

Bonnes pratiques
Création de modules
Programmation concurrente
Code natif
Python scientifique
Succès du langage

Le besoin

```
def f1():
    with open("/tmp/foo", "w") as f:
        f.write("Salut !")

#####
import threading

def f2():
    lock = threading.Lock()
    my_list = []
    with lock:
        my_list.append(1)
```

Gestionnaires de contexte

Matthieu Falce

Vue d'ensemble

Langage Python

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

La solution

```
class MonGestionnaire():
    def __enter__(self):
        print("entrée dans le bloc")

    def __exit__(self, type, value, traceback):
        print("sortie du bloc")

with MonGestionnaire():
    print("dans le block")
```

Gestionnaires de contexte

Matthieu Falce

Vue d'ensemble

Langage Python

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Le mot clé as

```
class ListeVide():
    def __enter__(self):
        self.ma_liste = []
        return self.ma_liste

    def __exit__(self, type, value, traceback):
        print("Nb éléments : {}".format(len(self.ma_liste)))

with ListeVide() as l:
    l.append(1)
    l.append(2)
```

Gestionnaires de contexte

Matthieu Falce

Vue d'ensemble

Langage Python

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation

Orientée objet
(POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Utiliser plusieurs gestionnaires en même temps

```
with open("/tmp/t1.txt", "w") as f, open("/tmp/t2.txt", "w") as g:  
    f.write("f - t1")  
    g.write("g - t2")
```

Déclaration d'une fonction

Matthieu Falce

Vue d'ensemble

Langage Python

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Tout est objet

Gestion des arguments

Gotchas

Higher order functions

Closures

Décorateurs

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de modules

Programmation concurrente

```
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)
```

Déclaration d'une fonction

Matthieu Falce

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

exemple_defaults() # 1, None
exemple_defaults(1, 2) # 1, 2
exemple_defaults(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

Gestion des variables
Structures de données
Duck typing
Slicing
Gestion des fichiers
Encodage des caractères
Contrôle de flux
Fonctions
Tout est objet
Gestion des arguments
Gotchas
Higher order functions
Closures
Décorateurs
Générateurs / Itérateurs
Exceptions
Introspection
Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

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

Gestion des variables
Structures de données
Duck typing
Slicing
Gestion des fichiers
Encodage des caractères
Contrôle de flux
Fonctions
Tout est objet
Gestion des arguments
Gotchas
Higher order functions
Closures
Décorateurs
Générateurs / Itérateurs
Exceptions
Introspection
Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Attributs de fonctions

Matthieu Falce

```
def une_fonction():
    counter = getattr(une_fonction, "counter", 0)
    une_fonction.counter = counter + 1

    print("je suis appelée")

une_fonction()
une_fonction()
une_fonction()
print(une_fonction.counter)
```

- ▶ on part du principe qu'une fonction est un objet
- ▶ permet de "donner une mémoire" à la fonction

Vue d'ensemble

Langage Python

Gestion des variables
Structures de données
Duck typing
Slicing
Gestion des fichiers
Encodage des caractères
Contrôle de flux
Fonctions
Tout est objet
Gestion des arguments
Gotchas
Higher order functions
Closures
Décorateurs
Générateurs / Itérateurs
Exceptions
Introspection
Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

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

Gestion des variables
Structures de données
Duck typing
Slicing
Gestion des fichiers
Encodage des caractères
Contrôle de flux
Fonctions
Tout est objet
Gestion des arguments
Gotchas
Higher order functions
Closures
Décorateurs
Générateurs / Itérateurs
Exceptions
Introspection
Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

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

Gestion des variables
Structures de données
Duck typing
Slicing
Gestion des fichiers
Encodage des caractères
Contrôle de flux
Fonctions
Tout est objet
Gestion des arguments
Gotchas
Higher order functions
Closures
Décorateurs
Générateurs / Itérateurs
Exceptions
Introspection
Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Arguments des fonctions

Matthieu Falce

Vue d'ensemble

Langage Python

Gestion des variables
Structures de données
Duck typing
Slicing
Gestion des fichiers
Encodage des caractères
Contrôle de flux
Fonctions
Tout est objet
Gestion des arguments
Gotchas
Higher order functions
Closures
Décorateurs
Générateurs / Itérateurs
Exceptions
Introspection
Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

```
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

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Tout est objet

Gestion des arguments

Gotchas

Higher order functions

Closures

Décorateurs

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

A votre avis, que donnent les fonctions suivantes ?

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

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

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

Arguments des fonctions

Matthieu Falce

Vue d'ensemble

Langage Python

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Tout est objet

Gestion des arguments

Gotchas

Higher order functions

Closures

Décorateurs

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

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 })
```

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, b=2)
f(1, c=2)
```

Vue d'ensemble

Langage Python

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Tout est objet

Gestion des arguments

Gotchas

Higher order functions

Closures

Décorateurs

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Liens avec le unpacking

Matthieu Falce

Vue d'ensemble

Langage Python

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Tout est objet

Gestion des arguments

Gotchas

Higher order functions

Closures

Décorateurs

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

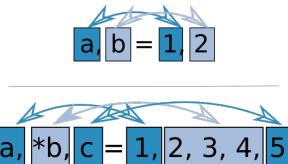
Bonnes pratiques

Création de
modules

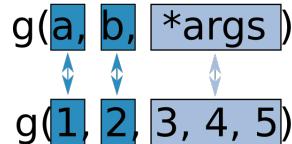
Programmation
concurrente

Unpacking

Pour les variables



Pour les arguments



Arguments des fonctions – résumé

Matthieu Falce

Vue d'ensemble

Langage Python

Gestion des variables
Structures de données
Duck typing
Slicing
Gestion des fichiers
Encodage des caractères
Contrôle de flux
Fonctions
Tout est objet
Gestion des arguments
Gotchas
Higher order functions
Closures
Décorateurs
Générateurs / Itérateurs
Exceptions
Introspection
Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Intérêts / limites

Matthieu Falce

Vue d'ensemble

Langage Python

Gestion des variables
Structures de données
Duck typing
Slicing
Gestion des fichiers
Encodage des caractères
Contrôle de flux
Fonctions
Tout est objet
Gestion des arguments
Gotchas
Higher order functions
Closures
Décorateurs
Générateurs / Itérateurs
Exceptions
Introspection
Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

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

Problèmes classiques – éléments mutables 14 15

```
# 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)
```

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

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

Matthieu Falce

Vue d'ensemble

Langage Python

Gestion des variables
Structures de données
Duck typing
Slicing
Gestion des fichiers
Encodage des caractères
Contrôle de flux
Fonctions
Tout est objet
Gestion des arguments
Gotchas
Higher order functions
Closures
Décorateurs
Générateurs / Itérateurs
Exceptions
Introspection
Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

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("après 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

Gestion des variables
Structures de données
Duck typing
Slicing
Gestion des fichiers
Encodage des caractères
Contrôle de flux
Fonctions
Tout est objet
Gestion des arguments
Gotchas
Higher order functions
Closures
Décorateurs
Générateurs / Itérateurs
Exceptions
Introspection
Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Problèmes classiques – portée des variables

Matthieu Falce

Vue d'ensemble

Langage Python

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Tout est objet

Gestion des arguments

Gotchas

Higher order functions

Closures

Décorateurs

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

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

Vue d'ensemble

Langage Python

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Tout est objet

Gestion des arguments

Gotchas

Higher order functions

Closures

Décorateurs

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

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])
```

Fonctions comme variables

```
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)
```



Matthieu Falce

Vue d'ensemble

Langage Python

- Gestion des variables
- Structures de données
- Duck typing
- Slicing
- Gestion des fichiers
- Encodage des caractères
- Contrôle de flux
- Fonctions
- Tout est objet
- Gestion des arguments
- Gotchas

Higher order functions

- Closures
- Décorateurs
- Générateurs / Itérateurs
- Exceptions
- Introspection
- Bibliographie

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Closures / Fermeture

Matthieu Falce

Vue d'ensemble

Langage Python

- Gestion des variables
- Structures de données
- Duck typing
- Slicing
- Gestion des fichiers
- Encodage des caractères
- Contrôle de flux
- Fonctions
- Tout est objet
- Gestion des arguments
- Gotchas

Higher order functions

- Closures
- Décorateurs
- Générateurs / Itérateurs
- Exceptions
- Introspection
- Bibliographie

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

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

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

Closures – Exemples

Matthieu Falce

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

def parler():

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

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

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

Vue d'ensemble

Langage Python

Gestion des variables
Structures de données
Duck typing
Slicing
Gestion des fichiers
Encodage des caractères
Contrôle de flux
Fonctions
Tout est objet
Gestion des arguments
Gotchas
Higher order functions

Closures

Décorateurs
Générateurs / Itérateurs
Exceptions
Introspection
Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Closures – Exemples

Matthieu Falce

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

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

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

Vue d'ensemble

Langage Python

Gestion des variables
Structures de données
Duck typing
Slicing
Gestion des fichiers
Encodage des caractères
Contrôle de flux
Fonctions
Tout est objet
Gestion des arguments
Gotchas
Higher order functions

Closures

Décorateurs
Générateurs / Itérateurs
Exceptions
Introspection
Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Décorateurs – Syntaxe

Matthieu Falce

[Vue d'ensemble](#)

[Langage Python](#)

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Tout est objet

Gestion des arguments

Gotchas

Higher order functions

Closures

Décorateurs

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

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

[Bonnes pratiques](#)

[Création de modules](#)

[Programmation concurrente](#)

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

Décorateurs – Syntaxe

Matthieu Falce

[Vue d'ensemble](#)

[Langage Python](#)

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Tout est objet

Gestion des arguments

Gotchas

Higher order functions

Closures

Décorateurs

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

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

[Bonnes pratiques](#)

[Création de modules](#)

[Programmation concurrente](#)

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

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

Décorateurs – Syntaxe

Matthieu Falce

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

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

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

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

print(test_deco_syntaxe())
```

Vue d'ensemble

Langage Python

Gestion des variables
Structures de données
Duck typing
Slicing
Gestion des fichiers
Encodage des caractères
Contrôle de flux
Fonctions
Tout est objet
Gestion des arguments
Gotchas
Higher order functions
Closures
Décorateurs
Générateurs / Itérateurs
Exceptions
Introspection
Bibliographie
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente

Décorateurs – Syntaxe

Matthieu Falce

```
# comment accepter des paramètres

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

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

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

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

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

Vue d'ensemble

Langage Python

Gestion des variables
Structures de données
Duck typing
Slicing
Gestion des fichiers
Encodage des caractères
Contrôle de flux
Fonctions
Tout est objet
Gestion des arguments
Gotchas
Higher order functions
Closures
Décorateurs
Générateurs / Itérateurs
Exceptions
Introspection
Bibliographie
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente

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

Gestion des variables
Structures de données
Duck typing
Slicing
Gestion des fichiers
Encodage des caractères
Contrôle de flux
Fonctions
Tout est objet
Gestion des arguments
Gotchas
Higher order functions
Closures
Décorateurs
Générateurs / Itérateurs
Exceptions
Introspection
Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Décorateurs paramétrés

Matthieu Falce

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

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

Vue d'ensemble

Langage Python

Gestion des variables
Structures de données
Duck typing
Slicing
Gestion des fichiers
Encodage des caractères
Contrôle de flux
Fonctions
Tout est objet
Gestion des arguments
Gotchas
Higher order functions
Closures
Décorateurs
Générateurs / Itérateurs
Exceptions
Introspection
Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Introspection ?

Matthieu Falce

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

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

Vue d'ensemble

Langage Python

Gestion des variables
Structures de données
Duck typing
Slicing
Gestion des fichiers
Encodage des caractères
Contrôle de flux
Fonctions
Tout est objet
Gestion des arguments
Gotchas
Higher order functions
Closures
Décorateurs
Générateurs / Itérateurs
Exceptions
Introspection
Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Introspection ?

Matthieu Falce

```
from functools import wraps

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

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

Gestion des variables
Structures de données
Duck typing
Slicing
Gestion des fichiers
Encodage des caractères
Contrôle de flux
Fonctions
Tout est objet
Gestion des arguments
Gotchas
Higher order functions
Closures
Décorateurs
Générateurs / Itérateurs
Exceptions
Introspection
Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Introspection ?

Matthieu Falce

implémentée à travers des variables magiques

- ▶ `__qualname__` : nom qualifié (chemin depuis le module)¹⁶
- ▶ `__name__` : nom de la fonction (pas de la variable qui la contient)
- ▶ `__doc__` : docstring de la fonction

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

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

Vue d'ensemble

Langage Python

Gestion des variables
Structures de données
Duck typing
Slicing
Gestion des fichiers
Encodage des caractères
Contrôle de flux
Fonctions
Tout est objet
Gestion des arguments
Gotchas
Higher order functions
Closures
Décorateurs
Générateurs / Itérateurs
Exceptions
Introspection
Bibliographie

Programmation Orientée objet (POO)

Bonnes pratiques
Création de modules
Programmation concurrente

Cas d'usage

Matthieu Falce

Vue d'ensemble

Langage Python

Gestion des variables
Structures de données
Duck typing
Slicing
Gestion des fichiers
Encodage des caractères
Contrôle de flux
Fonctions
Tout est objet
Gestion des arguments
Gotchas
Higher order functions
Closures
Décorateurs
Générateurs / Itérateurs
Exceptions
Introspection
Bibliographie

Programmation Orientée objet (POO)

Bonnes pratiques
Création de modules
Programmation concurrente

Protocole d'itération

Matthieu Falce

```
a = [1, 2, 3]
print(a, type(a)) # [1, 2, 3] <class 'list'>

it = iter(a)
print(it) # <list_iterator object at 0x7fa8359057b8>
print(type(it)) # <class 'list_iterator'>

print(next(it))
print(next(it))
print(next(it))
print(next(it))
```

Vue d'ensemble

Langage Python

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

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Les différents types

Matthieu Falce

```
nombres_pairs = (i for i in range(100_000_000) if i % 2 == 0)

for pair in nombres_pairs:
    print(pair)
    if pair > 10:
        break

print("fin du premier for")

# a partir de quel nombre est-ce que ça reprend ?
for pair in nombres_pairs:
    print(pair)
    if pair > 20:
        break
```

Vue d'ensemble

Langage Python

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

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Les différents types

Matthieu Falce

```
def generateur(nb_elements):
    print("début générateur")
    for i in range(nb_elements):
        yield i * 50
    print("fin générateur")

def generateur_2(nb_elements):
    """On peut utiliser de la délégation de générateurs avec yield from."""
    yield from (i*10 for i in range(nb_elements))

gen = generateur(4)

# première lecture
for element in gen:
    print(element)

print("-----")

# deuxième lecture
for element in gen: # ne rentre pas
    print(element)

next(gen) # raise StopIteration
```

Vue d'ensemble

Langage Python

Gestion des variables
Structures de données
Duck typing
Slicing
Gestion des fichiers
Encodage des caractères
Contrôle de flux
Fonctions
Générateurs / Itérateurs
Exceptions
Introspection
Bibliographie
Programmation
Orientée objet
(POO)
Bonnes pratiques
Création de
modules
Programmation
concurrente
Code natif
Python scientifique
Succès du langage

Les différents types

Matthieu Falce

```
class Iterateur():
    def __init__(self, nb_elements):
        self.nb_elements = nb_elements
        self.counter = 0

    def __iter__(self):
        print("dans iter")
        return self

    def __next__(self):
        print("dans next")
        if self.counter > self.nb_elements:
            raise StopIteration

        self.counter += 1
        return self.counter

it = Iterateur(5)
for i in it:
    print(i)

next(it) # StopIteration
```

Vue d'ensemble

Langage Python

Gestion des variables
Structures de données
Duck typing
Slicing
Gestion des fichiers
Encodage des caractères
Contrôle de flux
Fonctions
Générateurs / Itérateurs
Exceptions
Introspection
Bibliographie
Programmation
Orientée objet
(POO)
Bonnes pratiques
Création de
modules
Programmation
concurrente
Code natif
Python scientifique
Succès du langage

A quoi ça sert ?

Matthieu Falce

Vue d'ensemble

Langage Python

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

- ▶ évaluation paresseuse
 - ▶ flux de données infini
 - ▶ meilleure gestion de la mémoire
- ▶ traitement asynchrone sans callbacks
 - ▶ le yield bloque l'exécution jusqu'au prochain next

Pipeline

Matthieu Falce

Vue d'ensemble

Langage Python

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

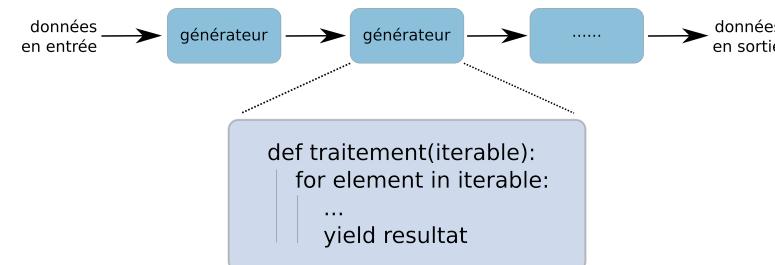
Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage



Pipeline

Matthieu Falce

```
def is_palindrome(nb):
    return str(nb) == str(nb)[::-1]

nombres_pairs = (i for i in range(100_000_000) if i % 2 == 0)
palindromes = (i for i in nombres_pairs if is_palindrome(i))
fois_cinquante = (i * 50 for i in palindromes)
trente_premiers = (i for i, j in zip(fois_cinquante, range(30)))

print(sum(trente_premiers))
```

Vue d'ensemble

Langage Python

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

Programmation
Orientée objet
(POO)

Bonnes pratiques
Création de modules
Programmation concurrente
Code natif
Python scientifique
Succès du langage

Pipeline

Matthieu Falce

```
def palindromes(iterable):
    for element in iterable:
        s_element = str(element)
        if s_element == s_element[::-1]:
            yield element

def nombres_pairs(iterable):
    for element in iterable:
        if element % 2 == 0:
            yield element

def fois_cinquante(iterable):
    for element in iterable:
        yield element * 50

def trente_premiers(iterable):
    for index, element in enumerate(iterable):
        if index > 30:
            break
        yield element

elements = range(100_000_000)
s = sum(trente_premiers(fois_cinquante(palindromes(nombres_pairs(elements)))))
```

Vue d'ensemble

Langage Python

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

Programmation
Orientée objet
(POO)

Bonnes pratiques
Création de modules
Programmation concurrente
Code natif
Python scientifique
Succès du langage

Itertools¹⁷

Matthieu Falce

Un module de la bibliothèque standard pour manipuler des itérables.

- ▶ générateurs infinis (count, cycle, repeat)
- ▶ chaînage d'itérateurs
- ▶ prendre un certain nombre d'éléments (dropwhile, takeWhile)
- ▶ prendre certains éléments (compress, filterfalse)
- ▶ ...

17.<https://docs.python.org/3/library/itertools.html>

Vue d'ensemble

Langage Python

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Consommateurs / Coroutines

Matthieu Falce

Vue d'ensemble

Langage Python

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation Orientée objet (POO)

Bonnes pratiques

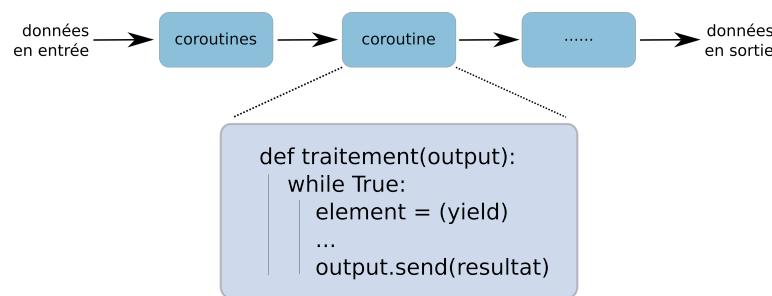
Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage



Consommateurs / Coroutines

```
def coroutine(func):
    """Permet d'appeler next la première fois automatiquement."""
    def wrapper(*arg, **kwargs):
        generator = func(*arg, **kwargs)
        next(generator)
        return generator
    return wrapper

@coroutine
def nombres_pairs(output):
    while True:
        element = (yield)
        if element % 2 == 0:
            output.send(element)

@coroutine
def fois_cinquante(output):
    while True:
        element = (yield)
        output.send(element * 50)

@coroutine
def afficher():
    while True:
        x = (yield)
        print(x)

aff = afficher()
multiplier = fois_cinquante(aff)
nb_pairs = nombres_pairs(multiplier)

for nombre in range(100):
    nb_pairs.send(nombre)
```

Matthieu Falce

Vue d'ensemble

Langage Python

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation

Orientée objet

(POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Intérêts

Matthieu Falce

Vue d'ensemble

Langage Python

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation

Orientée objet

(POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Exceptions

Matthieu Falce

Vue d'ensemble

Langage Python

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Toujours utiliser une exception précise et bien logguer les erreurs.

Sinon des erreurs peuvent en cacher d'autres.

Exceptions

Matthieu Falce

Vue d'ensemble

Langage Python

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

```
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'enlève pas lui-même")  
finally:  
    print("appelé quel que soit le parcours d'exception")
```

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

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Exceptions

Matthieu Falce

Vue d'ensemble

Langage Python

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Demander pardon plutôt que la permission

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

Matthieu Falce

Vue d'ensemble

Langage Python

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation

Orientée objet
(POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Introspection ?

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>

Matthieu Falce

Vue d'ensemble

Langage Python

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation

Orientée objet
(POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Introspection ?

Matthieu Falce

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

Vue d'ensemble

Langage Python

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Comment faire ?

Matthieu Falce

Vue d'ensemble

Langage Python

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

- ▶ 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
- ▶ ...

Comment faire ?

Matthieu Falce

Vue d'ensemble

Langage Python

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation

Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

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

A quoi ça sert ?

Matthieu Falce

Vue d'ensemble

Langage Python

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation

Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Bibliographie I

Matthieu Falce

► Générateurs

- ▶ <http://sametmax.com/parcourir-un-iterable-par-morceaux-en-python/>
- ▶ <https://www.pythonsheets.com/notes/python-generator.html>
- ▶ <https://brett.is/writing/about/generator-pipelines-in-python/>
- ▶ <http://xion.io/post/code/python-generator-args.html>
- ▶ <https://stackoverflow.com/questions/19302530/python-generator-send-function-purpose>
- ▶ <http://sametmax.com/quest-ce-quune-coroutine-en-python-et-a-quoi-ca-sert/>
- ▶ <http://treyhunner.com/2018/06/how-to-make-an-iterator-in-python/>
- ▶ <https://docs.python.org/3/library/itertools.html#itertools.cycle>

Vue d'ensemble

Langage Python

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

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Bibliographie II

Matthieu Falce

- ▶ <http://www.dabeaz.com/coroutines/Coroutines.pdf>
- ▶ <http://www.dabeaz.com/finalgenerator/FinalGenerator.pdf>
- ▶ 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 :

Vue d'ensemble

Langage Python

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

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Bibliographie III

- ▶ <https://docs.python.org/fr/3/tutorial/datastructures.html>
- ▶ http://python-prepa.github.io/information_theory.html
- ▶ <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/frdiveintopython/power_of_introspection/index.php
 - ▶ <https://docs.python.org/3/library/inspect.html>

Matthieu Falce

Vue d'ensemble

Langage Python

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Bibliographie IV

- ▶ <http://sametmax.com/comprendre-les-decorateur-python-pas-a-pas-partie-2/>
- ▶ Variables :
 - ▶ <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 :

Matthieu Falce

Vue d'ensemble

Langage Python

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Bibliographie V

- ▶ <http://sametmax.com/gestion-des-erreurs-en-python/>
- ▶ <http://sametmax.com/comment-recruter-un-developpeur-python/>
- ▶ <http://sametmax.com/pourquoi-utiliser-un-mechanisme-d-exceptions/>
- ▶ *Context managers*
 - ▶ <http://sametmax.com/les-context-managers-et-le-mot-cle-with-en-python/>
 - ▶ <https://alysivji.github.io/managing-resources-with-context-managers-pythonic.html>
 - ▶ <http://eigenhombre.com/introduction-to-context-managers-in-python.html>
- ▶ *Duck Typing*
 - ▶ <https://stackoverflow.com/questions/4205130/what-is-duck-typing>

Matthieu Falce

Vue d'ensemble

Langage Python

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Bibliographie VI

- ▶ <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/>
- ▶ <http://sametmax.com/les-trucmucheables-en-python/>
- ▶ <https://stackoverflow.com/questions/1952464/in-python-how-do-i-determine-if-an-object-is-iterable>
- ▶ <https://stackoverflow.com/questions/6589967/how-to-handle-duck-typing-in-python>

Matthieu Falce

Vue d'ensemble

Langage Python

Gestion des variables

Structures de données

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Générateurs / Itérateurs

Exceptions

Introspection

Bibliographie

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

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 ?

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

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

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 ?

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

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

https://fr.wikipedia.org/wiki/Programmation_orient%C3%A9e_objet

Constitution d'une classe

Matthieu Falce

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

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

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

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

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

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Vocabulaire

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Association entre classes

Matthieu Falce

2 grandes techniques pour associer des classes entre elles :

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

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

UML

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

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 ?

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de
modules

Programmation
concurrente

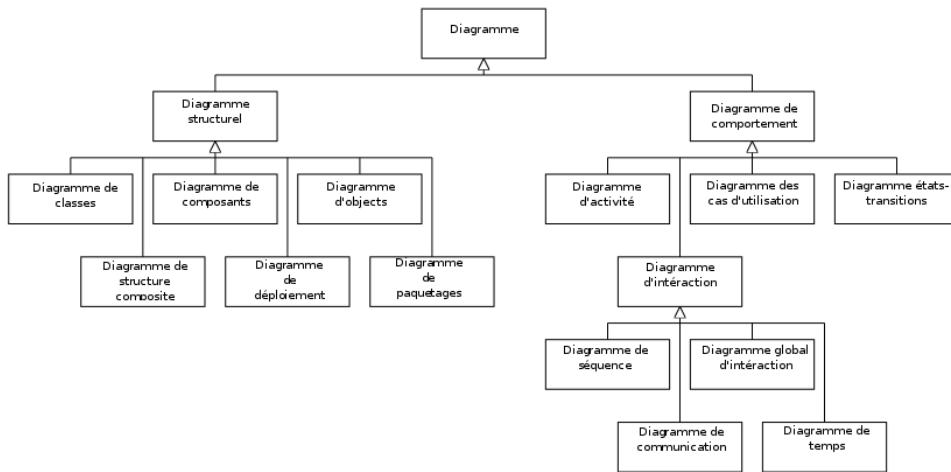
Code natif

Python scientifique

Succès du langage

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)

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de modules

Programmation concurrente

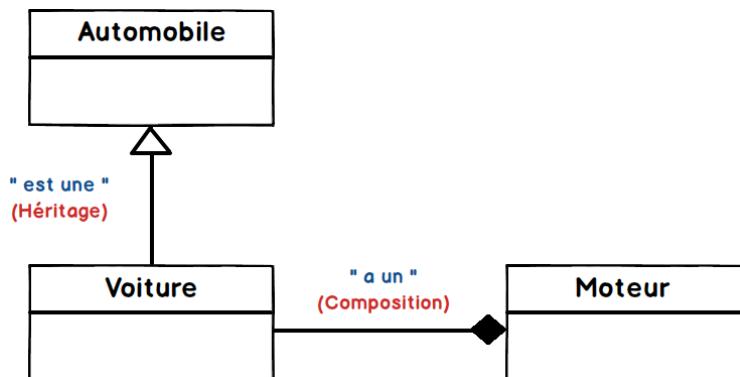
Code natif

Python scientifique

Succès du langage

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 ?

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

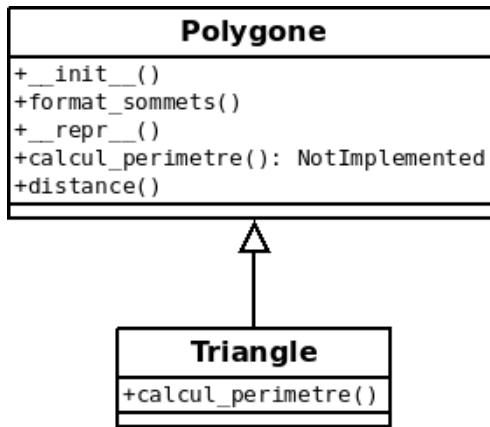
Python scientifique

Succès du langage

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 ?

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

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 ?

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

```
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 ?

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

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 ?

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

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 ?

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

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 ?

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

```
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 ?

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

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 ?

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

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 ?

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

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 ?

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

```
# 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 ?
Accès attributs
Méthodes
Classes abstraites
Héritage multiple
Design Patterns
Métaclasses
Bibliographie

Bonnes pratiques
Création de modules
Programmation concurrente
Code natif
Python scientifique
Succès du langage

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 ?
Accès attributs
Méthodes
Classes abstraites
Héritage multiple
Design Patterns
Métaclasses
Bibliographie

Bonnes pratiques
Création de modules
Programmation concurrente
Code natif
Python scientifique
Succès du langage

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 ?

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

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 ?

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

- ▶ 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

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

```
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
)
```

Dataclasses

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage



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
```

Le problème

Matthieu Falce

- ▶ tous les attributs sont publics par défaut
- ▶ on ne peut pas les encapsuler facilement
- ▶ les getters et setters sont souvent redondants

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Concepts
Association
Modélisation
POO en python
Gestion des exceptions
Classe ou pas ?
Accès attributs
Méthodes
Classes abstraites
Héritage multiple
Design Patterns
Métaclasses
Bibliographie
Bonnes pratiques
Création de modules
Programmation concurrente
Code natif
Python scientifique
Succès du langage

Le problème

Matthieu Falce

```
class TemperatureConverter:  
    def __init__(self, kelvin):  
        self.kelvin = kelvin  
        self.celsius = self.kelvin - 273.15  
  
    def get_celsius(self):  
        return self.kelvin - 273.15  
  
    def set_celsius(self, value_celsius):  
        self.celsius = value_celsius  
        self.kelvin = value_celsius + 273.15  
  
tc = TemperatureConverter(0)  
print(tc.get_celsius())  
  
tc.set_celsius(0)  
print(tc.kelvin)  
print(tc.get_celsius())  
  
tc.celsius = 45 # mouahah  
print(tc.kelvin)  
print(tc.get_celsius())
```

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Concepts
Association
Modélisation
POO en python
Gestion des exceptions
Classe ou pas ?
Accès attributs
Méthodes
Classes abstraites
Héritage multiple
Design Patterns
Métaclasses
Bibliographie
Bonnes pratiques
Création de modules
Programmation concurrente
Code natif
Python scientifique
Succès du langage

__getattr__ / __getattribute__

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage



__getattr__ / __getattribute__

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

```
class Inutile():
    def __init__(self):
        self.a = 1

    def __getattr__(self, attr_name):
        print("getattr", attr_name)
        return 2

i = Inutile()
print(i.a)
print(i.b)
```

__getattr__ / __getattribute__

```
class Inutile():
    def __init__(self):
        self.a = 1

    def __getattribute__(self, attr_name):
        print("getattribute", attr_name)
        return 3

i = Inutile()
print(i.a)
print(i.b)
```

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

__getattr__ / __getattribute__

```
class Inutile():
    def __init__(self):
        self.a = 1

    def __getattribute__(self, attr_name):
        print("getattribute", attr_name)
        if hasattr(self, attr_name):
            print("deja présent")
        return 3

i = Inutile()
print(i.a) # RecursionError
```

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Descriptors

Matthieu Falce

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Concepts
Association
Modélisation
POO en python
Gestion des exceptions
Classe ou pas ?
Accès attributs
Méthodes
Classes abstraites
Héritage multiple
Design Patterns
Métaclasses
Bibliographie
Bonnes pratiques
Création de modules
Programmation concurrente
Code natif
Python scientifique
Succès du langage

- ▶ utilisés pour les accès aux attributs
- ▶ définissent un protocole de lecture / écriture / suppression
- ▶ accès supplémentaire rajouté par Python

Descriptors

Matthieu Falce

```
class Celsius:  
    def __init__(self, value=0.0):  
        self.value = value  
  
    def __get__(self, instance, owner):  
        return instance.kelvin - 273.15  
  
    def __set__(self, instance, value):  
        self.value = value  
        instance.kelvin = value + 273.15  
  
class TemperatureConverter:  
    # un descripteur modifie le comportement d'une classe  
    # PAS d'une instance.  
    celsius = Celsius()  
  
    def __init__(self, kelvin):  
        self.kelvin = kelvin  
  
    tc = TemperatureConverter()  
    print(tc.celsius)  
    print(tc.kelvin)  
  
    tc.kelvin = 0  
    print(tc.kelvin)  
    print(tc.celsius)  
  
    tc.celsius = 0  
    print(tc.kelvin)  
    print(tc.celsius)
```

Descriptors

Matthieu Falce

```
# pourquoi ça marche ?  
  
# en gros, dans le code de `objet`  
def __getattribute__(self, attr):  
    obj = object.__getattribute__(self, attr)  
    if hasattr(obj, "__get__"):  
        return obj.__get__(self, type(self)) # là où la magie opère  
    return obj
```

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Concepts
Association
Modélisation
POO en python
Gestion des exceptions
Classe ou pas ?
Accès attributs
Méthodes
Classes abstraites
Héritage multiple
Design Patterns
Métaclasses
Bibliographie
Bonnes pratiques
Création de modules
Programmation concurrente
Code natif
Python scientifique
Succès du langage

Properties

Matthieu Falce

- ▶ gestion des accès en lecture / écriture / suppression
- ▶ décorateur avec une syntaxe un peu spéciale
- ▶ avantage : on ne sait pas que l'on ne modifie pas l'attribut directement
- ▶ les properties sont une sorte de descripteur

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Concepts
Association
Modélisation
POO en python
Gestion des exceptions
Classe ou pas ?
Accès attributs
Méthodes
Classes abstraites
Héritage multiple
Design Patterns
Métaclasses
Bibliographie
Bonnes pratiques
Création de modules
Programmation concurrente
Code natif
Python scientifique
Succès du langage

Properties

Matthieu Falce

```
class TemperatureConverter:  
    def __init__(self, kelvin):  
        self.kelvin = kelvin  
  
    @property  
    def celsius(self):  
        return self.kelvin - 273.15  
  
    @celsius.setter  
    def celsius(self, value_celsius):  
        self.kelvin = value_celsius + 273.15  
  
tc = TemperatureConverter(0)  
print(tc.celsius)  
print(tc.kelvin)  
  
tc.kelvin = 0  
print(tc.kelvin)  
print(tc.celsius)  
  
tc.celsius = 0  
print(tc.kelvin)  
print(tc.celsius)
```

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Conclusion

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Essayer par priorité :

- ▶ attribut classique
- ▶ property
- ▶ descriptor
- ▶ __getattr__
- ▶ __getattribute__

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 ?

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

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 ?

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

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 ?

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

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 ?

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Résumé

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

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())
```

Classe abstraite ?

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Exemple

Matthieu Falce

```
class FausseClasseAbstraite():
    def fausse_methode_abstraite(self):
        raise NotImplementedError

class ClasseFille(FausseClasseAbstraite):
    def fausse_methode_abstraite(self):
        return "surchargée"

fca = FausseClasseAbstraite()
fca.fausse_methode_abstraite() # NotImplementedError

cf = ClasseFille()
cf.fausse_methode_abstraite()
```

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Concepts
Association
Modélisation
POO en python
Gestion des exceptions
Classe ou pas ?
Accès attributs
Méthodes
Classes abstraites
Héritage multiple
Design Patterns
Métaclasses
Bibliographie
Bonnes pratiques
Création de modules
Programmation concurrente
Code natif
Python scientifique
Succès du langage

Exemple

Matthieu Falce

```
from abc import ABC, abstractmethod

class Abstraite(ABC):
    @abstractmethod
    def methode_abstraite(self):
        pass

class Fille(Abstraite):
    def methode_abstraite(self):
        print("methode abstraite de Fille")

a = Abstraite()
# TypeError: Can't instantiate abstract class Abstraite
# with abstract methods methode_abstraite

b = Fille()
b.methode_abstraite()
```

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Concepts
Association
Modélisation
POO en python
Gestion des exceptions
Classe ou pas ?
Accès attributs
Méthodes
Classes abstraites
Héritage multiple
Design Patterns
Métaclasses
Bibliographie
Bonnes pratiques
Création de modules
Programmation concurrente
Code natif
Python scientifique
Succès du langage

A quoi ça sert

```
from abc import ABC, abstractmethod
import sys

class Button(ABC):
    @abstractmethod
    def paint(self):
        print("Code multiplateforme commun")

class LinuxButton(Button):
    def paint(self):
        super().paint()
        print("code pour X11")

class WindowsButton(Button):
    def paint(self):
        super().paint()
        print("code pour dwm")

if "linux" in sys.platform.lower():
    button = LinuxButton()
elif "windows" in sys.platform.lower():
    button = WindowsButton()

button.paint()
```

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

A quoi ça sert

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Syntaxe

```
class A:  
    pass  
  
class B:  
    pass  
  
class C(A, B):  
    pass
```

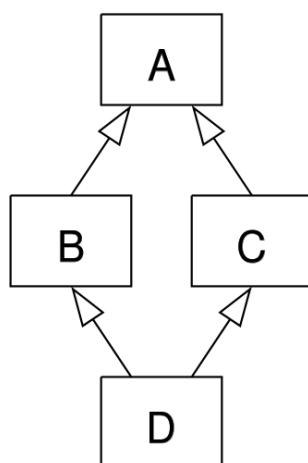
Matthieu Falce

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Concepts
Association
Modélisation
POO en python
Gestion des exceptions
Classe ou pas ?
Accès attributs
Méthodes
Classes abstraites
Héritage multiple
Design Patterns
Métaclasses
Bibliographie
Bonne pratiques
Création de modules
Programmation concurrente
Code natif
Python scientifique
Succès du langage

Problème

Héritage en diamant

Quelle méthode va être appelée ?



Matthieu Falce

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Concepts
Association
Modélisation
POO en python
Gestion des exceptions
Classe ou pas ?
Accès attributs
Méthodes
Classes abstraites
Héritage multiple
Design Patterns
Métaclasses
Bibliographie
Bonne pratique
Création de modules
Programmation concurrente
Code natif
Python scientifique
Succès du langage

Method Résolution Order (MRO)

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

19.https://en.wikipedia.org/wiki/C3_linearization

Method Résolution Order (MRO)

Exemple de résolution

```
class A1: pass
class A2: pass

class B(A1): pass
class C(B): pass

class D1(A1, A2): pass
class D2(A2, A1): pass

print('mro A', A1.__mro__) # A1 / objet
print('mro B', B.__mro__) # B / A1 / objet
print('mro C', C.__mro__) # C / B / A1 / objet

print('mro D1', D1.__mro__) # D1 / A1 / A2 / objet
print('mro D2', D2.__mro__) # D2 / A2 / A1 / objet
```

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Method Résolution Order (MRO)

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Héritage en diamant

```
class A: pass
class B(A): pass
class C(A): pass
class D(B, C): pass

print("MRO diamant : ", D.__mro__)
```

Method Résolution Order (MRO)

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

L'algorithme de linéarisation a cependant des limites...

```
class A: pass
class B(A): pass

class C(B, A): pass
class D(A, B): pass
# TypeError: Cannot create a consistent method resolution
# order (MRO) for bases A, B
```

Mixins

Matthieu Falce

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Concepts
Association
Modélisation
POO en python
Gestion des exceptions
Classe ou pas ?
Accès attributs
Méthodes
Classes abstraites
Héritage multiple
Design Patterns
Métaclasses
Bibliographie

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

- ▶ classe destinée à être composée par héritage
- ▶ service que l'on peut rajouter aux classes filles
- ▶ pas destinée à être utilisée seule

Mixins

Matthieu Falce

```
class Bonjour:  
    def __init__(self, name):  
        self.name = name  
  
    def parler(self):  
        print("bonjour {}".format(self.name))  
  
  
class DisEnCouleur:  
    def parler(self):  
        print("\033[1;31m", end="") # code terminal pour le rouge  
        print("bonjour {}".format(self.name), end="")  
        print("\033[0;0m") # code terminal pour remettre à zéro  
  
  
class DisPas:  
    def parler(self):  
        print("*** censuré ***")  
  
  
class BonjourEnCouleur(DisEnCouleur, Bonjour): pass  
class DisPasBonjour(DisPas, Bonjour): pass  
  
  
b = Bonjour("Matthieu")  
bec = BonjourEnCouleur("Matthieu")  
dpb = DisPasBonjour("Matthieu")  
  
b.parler()  
bec.parler()  
dpb.parler()
```

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Concepts
Association
Modélisation
POO en python
Gestion des exceptions
Classe ou pas ?
Accès attributs
Méthodes
Classes abstraites
Héritage multiple
Design Patterns
Métaclasses
Bibliographie

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

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 ?

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Patrons de conception ?

« 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.

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

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 ?

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

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 ?

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Patrons de conception ?

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)
```

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

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 ?

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

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 ?

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

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

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

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()
```

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 ?
Accès attributs
Méthodes
Classes abstraites
Héritage multiple
Design Patterns
Métaclasses
Bibliographie

Bonnes pratiques
Création de modules
Programmation concurrente
Code natif
Python scientifique
Succès du langage

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 ?

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

Matthieu Falce

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Concepts
Association
Modélisation
POO en python
Gestion des exceptions
Classe ou pas ?
Accès attributs
Méthodes
Classes abstraites
Héritage multiple
Design Patterns
Métaclasses
Bibliographie

Bonnes pratiques
Création de modules
Programmation concurrente
Code natif
Python scientifique
Succès du langage

Structural – Décorateur

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

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

Pas forcément !

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

Structural – Adapter

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

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())
```

Conclusion

Matthieu Falce

- ▶ 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

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Concepts
Association
Modélisation
POO en python
Gestion des exceptions
Classe ou pas ?
Accès attributs
Méthodes
Classes abstraites
Héritage multiple
Design Patterns
Métaclasses
Bibliographie

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Métaclasses – disclaimer

Matthieu Falce

Metaclasses are deeper magic that 99% of users should never worry about. If you wonder whether you need them, you don't (the people who actually need them know with certainty that they need them, and don't need an explanation about why).

Tim Peters (Python Guru)

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Concepts
Association
Modélisation
POO en python
Gestion des exceptions
Classe ou pas ?
Accès attributs
Méthodes
Classes abstraites
Héritage multiple
Design Patterns
Métaclasses
Bibliographie

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Métaclasses – début

Matthieu Falce

- ▶ Tout est objet en python
- ▶ Même les classes
- ▶ MÊME LES CLASSES



Tout ce temps, on m'a menti.

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Concepts
Association
Modélisation
POO en python
Gestion des exceptions
Classe ou pas ?
Accès attributs
Méthodes
Classes abstraites
Héritage multiple
Design Patterns
Métaclasses
Bibliographie
Bonnes pratiques
Création de modules
Programmation concurrente
Code natif
Python scientifique
Succès du langage

Métaclasses – création de classes

Matthieu Falce

```
> help(type)
```

```
Help on class type in module __builtin__:
```

```
class type(object)
| type(object) -> the object's type
| type(name, bases, dict) -> a new type
...
...
```



Tout ce temps, on m'a menti.

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Concepts
Association
Modélisation
POO en python
Gestion des exceptions
Classe ou pas ?
Accès attributs
Méthodes
Classes abstraites
Héritage multiple
Design Patterns
Métaclasses
Bibliographie
Bonnes pratiques
Création de modules
Programmation concurrente
Code natif
Python scientifique
Succès du langage

Métaclasses – Création d'une classe¹⁹

Matthieu Falce

```
# création classique
# d'une classe

class Foo(object):
    bar = True
```

```
# création de la même classe
# en utilisant type

Foo = type('Foo', (), {'bar':True})
```

```
>>> print(Foo)
<class '__main__.Foo'>
>>> print(Foo.bar)
True
>>> f = Foo()
>>> print(f)
<__main__.Foo object at 0x8a9b84c>
>>> print(f.bar)
True
```

19.<https://stackoverflow.com/questions/100003/what-are-metaclasses-in-python>

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Métaclasses – Héritage

Matthieu Falce

```
# FooChild hérite
# de foo.
# Construction classique.

class FooChild(Foo):
    pass
```

```
# FooChild hérite de foo.
# On passe le tuple des
# classes mères

FooChild = type(
    'FooChild', (Foo,), {})
```

```
>>> FooChild = type('FooChild', (Foo,), {})
>>> print(FooChild)
<class '__main__.FooChild'>
>>> print(FooChild.bar) # bar is inherited from Foo
True
```

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Métaclasses – Méthodes

Matthieu Falce

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Concepts
Association
Modélisation
POO en python
Gestion des exceptions
Classe ou pas ?
Accès attributs
Méthodes
Classes abstraites
Héritage multiple
Design Patterns
Métaclasses
Bibliographie
Bonnes pratiques
Création de modules
Programmation concurrente
Code natif
Python scientifique
Succès du langage

```
def echo_bar(self):
    print(self.bar)

FooChild = type('FooChild', (Foo,), {'echo_bar': echo_bar})

>>> hasattr(Foo, 'echo_bar')
False
>>> hasattr(FooChild, 'echo_bar')
True
>>> my_foo = FooChild()
>>> my_foo.echo_bar()
True
```

Métaclasses – Conclusion

Matthieu Falce

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Concepts
Association
Modélisation
POO en python
Gestion des exceptions
Classe ou pas ?
Accès attributs
Méthodes
Classes abstraites
Héritage multiple
Design Patterns
Métaclasses
Bibliographie
Bonnes pratiques
Création de modules
Programmation concurrente
Code natif
Python scientifique
Succès du langage

OK. Mais tout ça pour faire quoi ?

```
class UpperAttrMetaClass(type):
    def __new__(cls, clsname, bases, dct):
        uppercase_attr = {}
        for name, val in dct.items():
            if not name.startswith('__'):
                uppercase_attr[name.upper()] = val
            else:
                uppercase_attr[name] = val

        return super(UpperAttrMetaClass, cls).__new__(
            cls, clsname, bases, uppercase_attr
        )

# implémentation 1 (la plus classique)
class Test(metaclasse=UpperAttrMetaClass):
    text_exemple = "Un texte"

# implémentation 2
Test2 = UpperAttrMetaClass(
    'TestMinuscule', (), {"text_exemple": "Un texte"}
)
```

Métaclasses – Conclusion

Matthieu Falce

OK. Mais tout ça pour faire quoi ?

```
In [15]: Test.TEXT_EXEMPLE  
Out[15]: 'Un texte'
```

```
In [16]: Test.text_exemple
```

AttributeError

```
Traceback (most recent call last)  
<ipython-input-16-fb2ca099ba4e> in <module>()  
----> 1 Test.text_exemple
```

```
AttributeError: type object 'TestMinuscule'  
has no attribute 'text_exemple'
```

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Métaclasses – Conclusion

Matthieu Falce

OK. Mais tout ça pour faire quoi ?

Les seuls cas d'utilisation que je connais sont pour la création d'API facilement manipulables.

Exemple avec django

```
class Person(models.Model):  
    name = models.CharField(max_length=30)  
    age = models.IntegerField()  
  
guy = Person(name='bob', age='35')  
print(guy.age)  
# 35 et non pas models.IntegerField  
# Fait les requêtes nécessaires en base  
# pour récupérer les données si besoin
```

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

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 ?
Accès attributs
Méthodes
Classes abstraites
Héritage multiple
Design Patterns
Métaclasses
Bibliographie
Bonnes pratiques
Création de modules
Programmation concurrente
Code natif
Python scientifique
Succès du langage

Bibliographie II

- ▶ Descriptors :
 - ▶ <https://docs.python.org/3/howto/descriptor.html>
 - ▶ <https://docs.python.org/3/reference/datamodel.html#customizing-attribute-access>
 - ▶ <https://eev.ee/blog/2012/05/23/python-faq-descriptors/>
 - ▶ <https://stackoverflow.com/questions/22616559/use-cases-for-property-vs-descriptor-vs-get-attribute>
 - ▶ <https://www.smallsurething.com/python-descriptors-made-simple/>
 - ▶ <https://stackoverflow.com/questions/12846116/python-descriptor-vs-property>
 - ▶ <https://stackoverflow.com/questions/17330160/how-does-the-property-decorator-work>
 - ▶ <http://sametmax.com/les-descripteurs-en-python/>

Matthieu Falce

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Concepts
Association
Modélisation
POO en python
Gestion des exceptions
Classe ou pas ?
Accès attributs
Méthodes
Classes abstraites
Héritage multiple
Design Patterns
Métaclasses
Bibliographie
Bonnes pratiques
Création de modules
Programmation concurrente
Code natif
Python scientifique
Succès du langage

Bibliographie III

- ▶ <https://stackoverflow.com/questions/3798835/understanding-get-and-set-and-python-descriptors>
- ▶ <https://stackoverflow.com/questions/23309698/why-is-the-descriptor-not-getting-called-when-defined-as-instance-attribute>
- ▶ <https://stackoverflow.com/questions/4877290/what-is-the-dict-dict-attribute-of-a-python-class#4877655>
- ▶ 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>

Matthieu Falce

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Concepts
Association
Modélisation
POO en python
Gestion des exceptions
Classe ou pas ?
Accès attributs
Méthodes
Classes abstraites
Héritage multiple
Design Patterns
Métaclasses
Bibliographie

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Bibliographie IV

- ▶ <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)
- ▶ Héritage multiple / MRO :
 - ▶ https://en.wikipedia.org/wiki/Multiple_inheritance
 - ▶ https://en.wikipedia.org/wiki/C3_linearization

Matthieu Falce

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Concepts
Association
Modélisation
POO en python
Gestion des exceptions
Classe ou pas ?
Accès attributs
Méthodes
Classes abstraites
Héritage multiple
Design Patterns
Métaclasses
Bibliographie

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Bibliographie V

- ▶ <https://stackoverflow.com/questions/3277367/how-does-pythons-super-work-with-multiple-inheritance>
 - ▶ <https://easyaspython.com/mixins-for-fun-and-profit-cb9962760556>
 - ▶ <https://stackoverflow.com/questions/14088294/multithreaded-web-server-in-python>
- ▶ Métaclasses:
- ▶ <https://stackoverflow.com/questions/392160/what-are-some-concrete-use-cases-for-metaclasses>
 - ▶ <http://www.dabeaz.com/py3meta/Py3Meta.pdf>
 - ▶ <https://stackoverflow.com/questions/6760685/creating-a-singleton-in-python>

Matthieu Falce

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Concepts
Association
Modélisation
POO en python
Gestion des exceptions
Classe ou pas ?
Accès attributs
Méthodes
Classes abstraites
Héritage multiple
Design Patterns
Metaclasses
Bibliographie
Bonnes pratiques
Création de modules
Programmation concurrente
Code natif
Python scientifique
Succès du langage

Bibliographie VI

- ▶ <http://sametmax.com/le-guide-ultra-complet-sur-la-programmation-orientee-objet-en-python-a-lusage-des-debutants-qui-sont-rassurés-par-les-textes-détaillés-qui-prennent-le-temps-de-tout-expliquer-partie-8/>
 - ▶ <https://stackoverflow.com/questions/5730211/how-does-get-field-display-in-django-work>
- ▶ Monkey Patching (pas au programme):
- ▶ <https://stackoverflow.com/questions/2375403/how-do-you-monkey-patch-a-function-in-python>
 - ▶ <https://stackoverflow.com/questions/5626193/what-is-monkey-patching>
 - ▶ <https://makina-corpus.com/blog/metier/2016/how-to-make-a-python-method>
 - ▶ <https://twitter.com/raymondh/status/771628923100667904>

Matthieu Falce

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Concepts
Association
Modélisation
POO en python
Gestion des exceptions
Classe ou pas ?
Accès attributs
Méthodes
Classes abstraites
Héritage multiple
Design Patterns
Metaclasses
Bibliographie
Bonnes pratiques
Création de modules
Programmation concurrente
Code natif
Python scientifique
Succès du langage

Bibliographie VII

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Accès attributs

Méthodes

Classes abstraites

Héritage multiple

Design Patterns

Métaclasses

Bibliographie

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Bonnes pratiques

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Débug

Linter

Analysse des performances

Tests

Documentation

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

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

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

La QA (*Quality Assurance*)

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

Pourquoi ?

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Débug

Linter

Analyse des performances

Tests

Documentation

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

- ▶ 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, ...)

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

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Écosystème

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

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

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

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

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

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

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

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

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

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

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

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

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

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

- ▶ 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

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

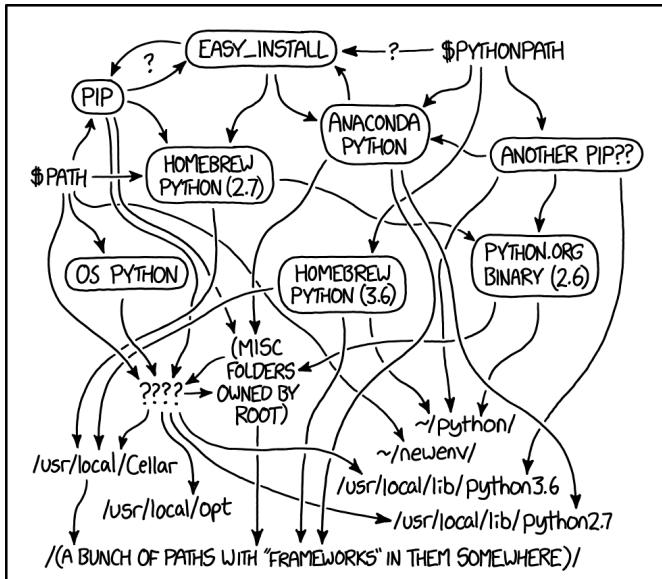
Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage



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

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

Environnement d'installation sain

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

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

- ▶ 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

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

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

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

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

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

```
#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
```

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

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Python système

Projet data 1

python 2.7
seaborn 0.9.0
numpy 1.16.1
pandas 0.24.1
...

Projet web 1

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

Projet data 2

python 3.6
scipy 0.19.0
numpy 1.13.1
pandas 0.20.1
...

...

Coexistence de plusieurs versions de Python

virtualenv

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Avant propos

Écosystème

pip

Environnements virtuels

Débug

Linter

Analyse des performances

Tests

Documentation

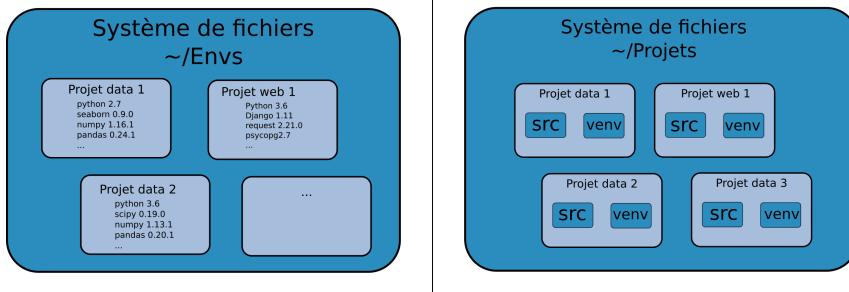
Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage



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

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

- ▶ 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

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

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

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Qualité du code – pep8 / linters

Matthieu Falce

Il existe des “linters” pour vous assister dans l’écriture.²⁰ ²¹
Ils peuvent lister les erreurs, variables non déclarées, typos,
mauvais import...

- ▶ flake8
- ▶ pylint
- ▶ ...

Intégration avec les éditeurs de texte.

20.<https://stackoverflow.com/questions/5611776/what-are-the-comprehensive-lint-checkers-for-python>
21.<https://stackoverflow.com/questions/1428872/pylint-pychecker-or-pyflakes?noredirect=1&lq=1>

Qualité du code – pep8 / linters

Matthieu Falce

Certains outils reformatent automatiquement le code que
vous leur donnez (concentration sur le code plutôt que la
présentation).²⁰ ²¹

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

Intégration avec les éditeurs de texte.

20.<https://medium.com/3yourmind/auto-formatters-for-python-8925065f9505>
21.<https://news.ycombinator.com/item?id=17155048>

Timing et profilage

Matthieu Falce

```
import time, timeit, cProfile

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

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

tic = time.time()
fonction_1()
print("fonction 1 : {}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()')
```

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Débug

Linter

Analyse des performances

Tests

Documentation

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Timing et profilage

Matthieu Falce

Résultat

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

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

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Débug

Linter

Analyse des performances

Tests

Documentation

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

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 unittest²²

22.<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, 0.3333333333)

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

Résultat :

```
python test_unittest.py
```

```
....
```

```
-----
```

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

```
OK
```

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

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

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

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Tests unitaires – fixtures

Matthieu Falce

Voilà le résultat :

```
In setUpClass()

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

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

In tearDownClass()
3

-----
Ran 2 tests in 0.000s

OK
```

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Débug

Linter

Analyse des performances

Tests

Documentation

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Aller plus loin

Matthieu Falce

Bonne explication du module unittest :

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

Pour aller plus loin:

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

Cycle TDD (*Test Driven Development*)

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

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Débug

Linter

Analyse des performances

Tests

Documentation

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Aller plus loin

Il existe d'autres modules pour lancer les tests ('testrunners')
23.

- ▶ (doctest²⁴)
- ▶ nose²⁵
- ▶ pytest (allège la syntaxe des tests)²⁶

Les tests sont souvent utilisés avec des 'mocks'²⁷ pour modifier le comportement des modules externes.

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

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

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

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

27.<https://docs.python.org/3.6/library/unittest.mock.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

Création de
modules

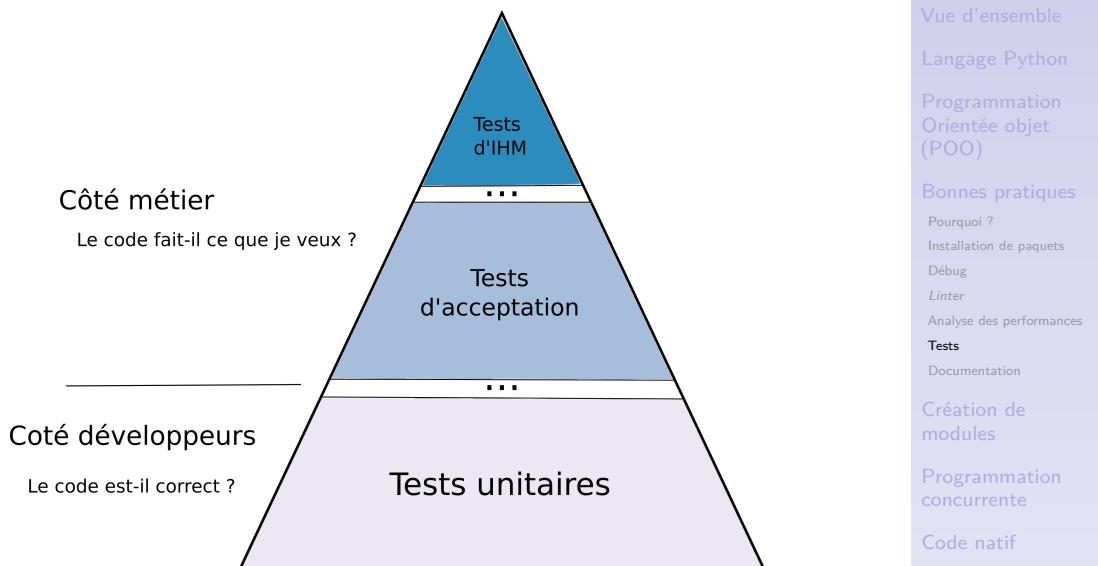
Programmation
concurrente

Code natif

Python scientifique

Succès du langage

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

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

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

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

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

```
""" Show how to display docstrings in python."""

```

```
# help(int)
# print(int.__doc__)
```

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

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Exemple minimal

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

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

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

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)
```

Outils d'extraction de documentation

Matthieu Falce

- ▶ sphinx (semi automatique) avec :
 - ▶ autosummary ²⁸
 - ▶ autodoc ²⁹
- ▶ sphinx (automatique) avec :
 - ▶ autoapi ³⁰
 - ▶ sphinx-autoapi ³¹
- ▶ pdoc ³²
- ▶ pydoc ³³
- ▶ doxygen ³⁴

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

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

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

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

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

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

34.<https://www.stack.nl/~dimitri/doxygen/>

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Débug

Linter

Analyse des performances

Tests

Documentation

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Syntaxe pour extraction automatique

Matthieu Falce

- ▶ PEP 8 : <https://www.python.org/dev/peps/pep-0008/#documentation-strings>
- ▶ PEP 257 :
<https://www.python.org/dev/peps/pep-0257/>
- ▶ pdoc : markdown ³⁵
- ▶ doxygen : markdown + syntaxe spécifique ³⁶
- ▶ sphinx : RestructuredText ³⁷
- ▶ sphinx avec extension Napoleon ³⁸
 - ▶ Google ³⁹
 - ▶ Numpy ⁴⁰

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

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

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

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

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

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

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Débug

Linter

Analyse des performances

Tests

Documentation

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Formatage des docstrings – Doxygen

Matthieu Falce

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

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

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

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

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

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Débug

Linter

Analyse des performances

Tests

Documentation

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Formatage des docstrings – Doxygen

Matthieu Falce

Python

The screenshot shows a Python namespace reference generated by Doxygen. At the top, there's a navigation bar with 'Main Page', 'Packages', 'Classes', and 'Functions'. Below it, the title 'pyexample Namespace Reference' is displayed. The page content is organized into sections: 'Classes' (listing 'PyClass' with its docstring), 'Functions' (listing 'func()' with its docstring), and 'Detailed Description' (containing general module documentation). A 'Function Documentation' section is also present, showing the detailed view for 'func()'. At the bottom right, a footer notes 'Generated by doxygen 1.8.15'.

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Débug

Linter

Analyse des performances

Tests

Documentation

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Résultat HTML de l'exemple précédent

Formatage des docstrings – reST

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Débug

Linter

Analyse des performances

Tests

Documentation

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

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

Formatage des docstrings – Google vs Numpy

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Débug

Linter

Analyse des performances

Tests

Documentation

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

```
"""
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
"""


```

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.

Résultat HTML de l'exemple précédent

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Débug

Linter

Analyse des performances

Tests

Documentation

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Bibliographie

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Pourquoi ?

Installation de paquets

Débug

Linter

Analyse des performances

Tests

Documentation

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

► 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>
- generation / 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

Création de
modules

Setuptools / Distutils

Structure d'un module

Installation

Mise en ligne PyPI

Bibliographie

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Création de modules

Setuptools / Distutils

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Setuptools / Distutils

Structure d'un module

Installation

Mise en ligne PyPI

Bibliographie

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Permettent d'empaqueter un module python.

Coexistence de **Setuptools** et **Distutils** → confusion

On va utiliser **setuptools**

<https://stackoverflow.com/questions/25337706/setuptools-vs-distutils-why-is-distutils-still-a-thing>

Arborescence et fichiers spécifiques à respecter

Structure d'un module

Code (`__init__.py`) :

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

Arborescence :

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



Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Setuptools / Distutils

Structure d'un module

Installation

Mise en ligne PyPI

Bibliographie

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

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

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Setuptools / Distutils

Structure d'un module

Installation

Mise en ligne PyPI

Bibliographie

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Installation

Contenu de setup.py

```
from setuptools import setup

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

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Setuptools / Distutils
Structure d'un module

Installation

Mise en ligne PyPI
Bibliographie

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Installation

Installation avec pip

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

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

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

Utilisation du module

Depuis n'importe où sur l'ordinateur

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

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Setuptools / Distutils
Structure d'un module

Installation

Mise en ligne PyPI
Bibliographie

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Exemple plus complet

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Setuptools / Distutils
Structure d'un module

Installation

Mise en ligne PyPI
Bibliographie

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

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

Exemple plus complet

Matthieu Falce

Arborescence :

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

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

```
2 directories, 7 files
```

Exemple plus complet

Matthieu Falce

Contenu de `text.py` :

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

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

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

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Setuptools / Distutils
Structure d'un module

Installation

Mise en ligne PyPI
Bibliographie

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Exemple plus complet

Matthieu Falce

Contenu de
`__init__.py` :

```
from .text import joke
```

Contenu de
`command_line.py` :

```
import funniest

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

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Setuptools / Distutils
Structure d'un module

Installation

Mise en ligne PyPI
Bibliographie

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Contenu de `blagues.txt` :

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

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

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

Exemple plus complet

Contenu de setup.py :

```
from setuptools import setup

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

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

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Setuptools / Distutils
Structure d'un module

Installation

Mise en ligne PyPI
Bibliographie

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Qui / Où ?

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Setuptools / Distutils
Structure d'un module
Installation

Mise en ligne PyPI
Bibliographie

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

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

Qui / Où ?

Matthieu Falce

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

A placer dans `~/.pypirc`

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

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

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

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

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Setuptools / Distutils
Structure d'un module
Installation

Mise en ligne PyPI
Bibliographie

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Pas à pas

Matthieu Falce

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

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

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

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

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Setuptools / Distutils
Structure d'un module
Installation

Mise en ligne PyPI
Bibliographie

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Bibliographie I

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Setuptools / Distutils

Structure d'un module

Installation

Mise en ligne PyPI

Bibliographie

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

► Packaging / installation

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

Programmation concurrente

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Avant propos / définitions
GIL

Parallélisation

Bibliographie

Code natif

Python scientifique

Succès du langage

Définition

Matthieu Falce

Programmation concurrente

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Avant propos / définitions

GIL

Parallélisation

Bibliographie

Code natif

Python scientifique

Succès du langage

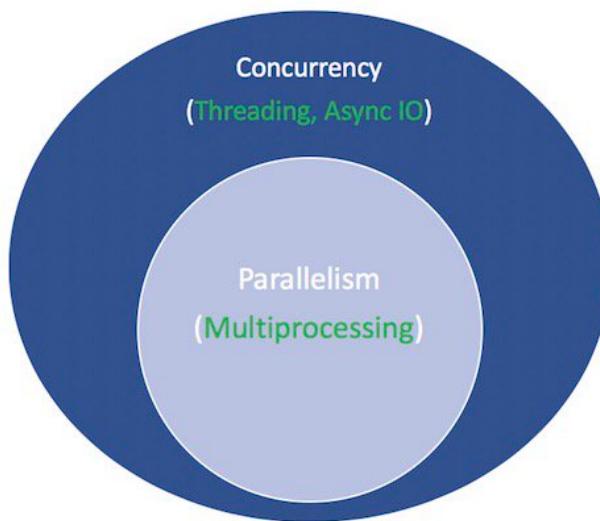
Les programmes concurrents permettent d'exécuter plusieurs tâches en même temps.

Enfin, plus ou moins, mais à l'échelle humaine ils semblent tous faire plusieurs choses en même temps...

- ▶ programmation parallèle : on effectue des opérations en parallèle (sur CPU ou GPU)
- ▶ programmation concurrente : plus générale que le parallélisme, on a plusieurs entités indépendantes (un client et un serveur)

Illustration

Matthieu Falce



Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Avant propos / définitions

GIL

Parallélisation

Bibliographie

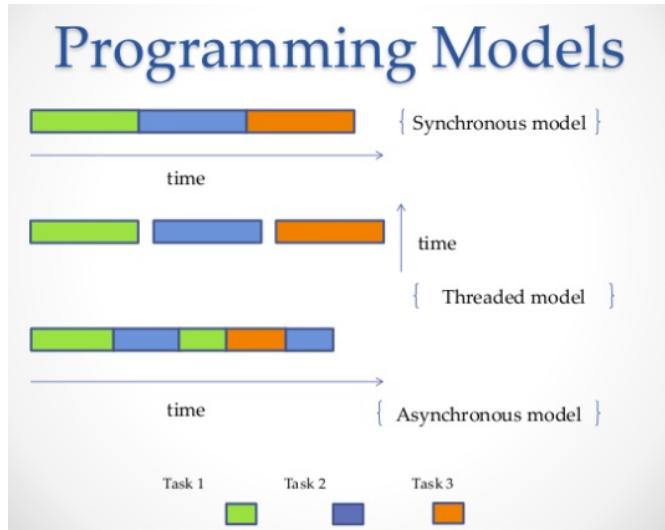
Code natif

Python scientifique

Succès du langage

Source : <https://realpython.com/async-io-python/>

Illustration



Source :

<https://medium.com/velotio-perspectives/an-introduction-to-asynchronous-programming-in-python-af0189a88bbb>

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Avant propos / définitions

GIL

Parallélisation

Bibliographie

Code natif

Python scientifique

Succès du langage

En python

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Avant propos / définitions

GIL

Parallélisation

Bibliographie

Code natif

Python scientifique

Succès du langage

Probablement l'un des plus gros chantiers de Python récemment :

- ▶ on peut faire les deux : `threading`, `asyncio`, `multiprocess`
- ▶ semble complexe à mettre en place
- ▶ problème du GIL
- ▶ introduction en marche forcée de `async` (pour ne pas être à la traîne par rapport aux langages à la mode)

Quand utiliser quoi ?

Dans tous les cas, les programmes devant intéragir avec l'extérieur doivent être concurrents d'une façon où d'une autre.

- ▶ si forte utilisation CPU : paralléliser le code sur plusieurs coeurs / processeurs / machines
 - ▶ exemple culinaire : être plusieurs à peeler des patates
 - ▶ les processus sont indépendants
 - ▶ nécessite un moyen d'échanger les données entre processus (réseaux, pipes, bdd...)
- ▶ si forte utilisation d'IO : privilégier les threads (processus légers) ou les coroutines (asynchrone)
 - ▶ exemple culinaire : peeler les tomates pendant la cuisson des pâtes
 - ▶ très sensible aux performances et aux appels longs
 - ▶ un seul processus donc on peut utiliser des techniques *classiques* de partage de données

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Avant propos / définitions

GIL

Parallélisation

Bibliographie

Code natif

Python scientifique

Succès du langage

Définition

Global Interpreter Lock

- ▶ simplicité de l'interpréteur Python
- ▶ bloque l'exécution de plusieurs threads à la fois
 - ▶ la thread qui s'exécute bloque le GIL
 - ▶ le GIL est relâché sur les opérations d'IO / périodiquement
 - ▶ → multitâche coopératif

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Avant propos / définitions

GIL

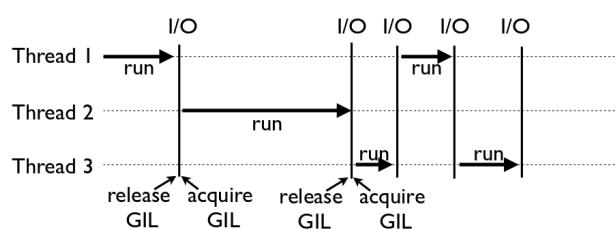
Parallélisation

Bibliographie

Code natif

Python scientifique

Succès du langage

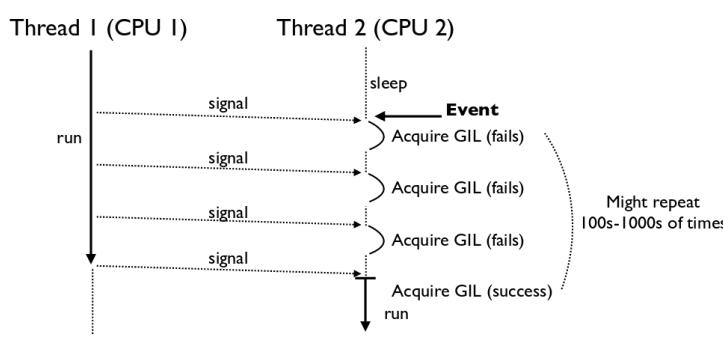


<http://www.dabeaz.com/python/UnderstandingGIL.pdf>

Problèmes du GIL

Matthieu Falce

Architectures multi coeurs
Les threads peuvent s'exécuter sur plusieurs coeurs *en même temps*



<http://www.dabeaz.com/python/UnderstandingGIL.pdf>

Problèmes du GIL

Matthieu Falce

- Gestion des IO
- ▶ les IO ne sont pas forcément bloquants
 - ▶ le *buffering* permet à l'OS de compléter des requêtes IO immédiatement
 - ▶ le GIL est toujours libéré → trop de changement de contexte si charge importante

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Avant propos / définitions
GIL
Parallélisation
Bibliographie
Code natif
Python scientifique
Succès du langage

threads – fonction, classe et verrous

Matthieu Falce

```
from threading import Thread
import time

# Define a function for the thread
def print_time(threadName, delay):
    count = 0
    while count < 5:
        time.sleep(delay)
        count += 1
        print("{} , {}".format(
            threadName,
            time.ctime(time.time())))
    )

# Create two threads as follows
t1 = Thread(target=print_time, args=("Thread-1", 0.1,))
t2 = Thread(target=print_time, args=("Thread-2", 0.2,))

t1.start(); t2.start()
print("Pas bloqué")
t1.join(); t2.join()
print("Fini")
```

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Avant propos / définitions
GIL

Parallélisation

Bibliographie

Code natif

Python scientifique

Succès du langage

threads – fonction, classe et verrous

Matthieu Falce

```
from threading import Thread
import time

class PrintNoBLock(Thread):
    def __init__(self, name, delay):
        super().__init__()
        self.running = False
        self.name = name
        self.delay = delay

    def run(self):
        self.running = True
        count = 0
        while count < 5 and self.running:
            time.sleep(self.delay)
            count += 1
            print("{} , {}".format(
                self.name,
                time.ctime(time.time())))
        )

# Create two threads as follows
t1 = PrintNoBLock("Thread-1", 0.1)
t2 = PrintNoBLock("Thread-2", 0.2)

t1.start(); t2.start()
print("Pas bloqué")
t1.join(); t2.join()
print("Fini")
```

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Avant propos / définitions
GIL

Parallélisation

Bibliographie

Code natif

Python scientifique

Succès du langage

threads – fonction, classe et verrous

```
from threading import Thread, RLock
import time, random, sys

lock = RLock()

class NoProblemSync(Thread):
    def __init__(self, name):
        super().__init__()
        self.name = name; self.running = True
        self.s = ["tortue", "cheval", "tomate"]

    def run(self):
        count = 0
        while count < 3 and self.running:
            with lock:
                if not self.running: return
                print(self.name, end=" ")
                for char in random.choice(self.s):
                    if not self.running: return
                    print(char, end='')
                    sys.stdout.flush()
                    time.sleep(random.random())
            print()
            time.sleep(random.random() / 5)
            count += 1

    # Create two threads as follows
    t1 = NoProblemSync("Thread-1")
    t2 = NoProblemSync("Thread-2")

    try:
        t1.start(); t2.start(); t1.join(); t2.join()
    except KeyboardInterrupt:
        t1.running = False; t2.running = False
```

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Avant propos / définitions
GIL

Parallélisation

Bibliographie

Code natif

Python scientifique

Succès du langage

Multiprocess

```
import multiprocessing
import time

print("Outside worker")

def worker(num):
    """worker function"""
    print("Worker:", num, time.time())

if __name__ == "__main__":
    jobs = []
    for i in range(10):
        p = multiprocessing.Process(target=worker, args=(i,))
        jobs.append(p)
        p.start()
    for job in jobs:
        job.join()
```

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Avant propos / définitions
GIL

Parallélisation

Bibliographie

Code natif

Python scientifique

Succès du langage

Performances Threading – Multiprocess

```
import time

def cpu_bound(nb):
    res = 0
    for val in range(nb * 10_000):
        res += val
    return res

def io_bound(nb):
    for _ in range(nb):
        res = open("/etc/fstab", "r").readlines()
    return res

def timer(f):
    def wrapper(nb, diff):
        tstart = time.time()
        f(nb, diff)
        res = "workers: {}\tdifficulty: {}\t name: {} \t time: {:.4f}"
        print(res.format(nb, diff, f.__name__, time.time()-tstart))
    return wrapper

def wrapper_parallel(methode, target, nb_workers, difficulty):
    jobs = []
    for _ in range(nb_workers):
        p = methode(target=target, args=(int(difficulty / nb_workers),))
        jobs.append(p)
        p.start()

    for job in jobs:
        job.join()
```

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Avant propos / définitions
GIL

Parallélisation

Bibliographie

Code natif

Python scientifique

Succès du langage

Performances Threading – Multiprocess

```
#!/python

import threading
import multiprocessing
from workers import cpu_bound, io_bound, timer, wrapper_parallel

@timer
def mp_io(nb, diff):
    return wrapper_parallel(multiprocessing.Process, io_bound, nb, diff)

@timer
def mp_cpu(nb, diff):
    return wrapper_parallel(multiprocessing.Process, cpu_bound, nb, diff)

@timer
def th_io(nb, diff):
    return wrapper_parallel(threading.Thread, io_bound, nb, diff)

@timer
def th_cpu(nb, diff):
    return wrapper_parallel(threading.Thread, cpu_bound, nb, diff)

if __name__ == "__main__":
    difficulties = [10, 1000, 10_000]
    nbs = [1, 2, 4, 8]
    for diff in difficulties:
        for function in [mp_cpu, th_cpu, mp_io, th_io]:
            for nb in nbs:
                function(nb, diff)
                print("#####\n")
print("#####\n")
```

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Avant propos / définitions
GIL

Parallélisation

Bibliographie

Code natif

Python scientifique

Succès du langage

Performances Threading – Multiprocess

Matthieu Falce

Nb CPU gérés par Ubuntu 18.04 (linux 4.15.0-34-generic)

```
workers: 1      difficulty: 10          name: mp_cpu        time: 0.0317
workers: 2      difficulty: 10          name: mp_cpu        time: 0.0207
workers: 4      difficulty: 10          name: mp_cpu        time: 0.0313
workers: 8      difficulty: 10          name: mp_cpu        time: 0.0720

workers: 1      difficulty: 10          name: th_cpu         time: 0.0299
workers: 2      difficulty: 10          name: th_cpu         time: 0.0295
workers: 4      difficulty: 10          name: th_cpu         time: 0.0163
workers: 8      difficulty: 10          name: th_cpu         time: 0.0120

#####
workers: 1      difficulty: 10          name: mp_io          time: 0.0091
workers: 2      difficulty: 10          name: mp_io          time: 0.0062
workers: 4      difficulty: 10          name: mp_io          time: 0.0144
workers: 8      difficulty: 10          name: mp_io          time: 0.0318

workers: 1      difficulty: 10          name: th_io           time: 0.0020
workers: 2      difficulty: 10          name: th_io           time: 0.0045
workers: 4      difficulty: 10          name: th_io           time: 0.0038
workers: 8      difficulty: 10          name: th_io           time: 0.0016
```

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Avant propos / définitions GIL

Parallélisation

Bibliographie

Code natif

Python scientifique

Succès du langage

Performances Threading – Multiprocess

Matthieu Falce

Nb CPU gérés par Ubuntu 18.04 (linux 4.15.0-34-generic)

```
workers: 1      difficulty: 1000         name: mp_cpu        time: 0.8607
workers: 2      difficulty: 1000         name: mp_cpu        time: 0.5459
workers: 4      difficulty: 1000         name: mp_cpu        time: 0.3685
workers: 8      difficulty: 1000         name: mp_cpu        time: 0.4328

workers: 1      difficulty: 1000         name: th_cpu         time: 0.7932
workers: 2      difficulty: 1000         name: th_cpu         time: 1.0545
workers: 4      difficulty: 1000         name: th_cpu         time: 0.9198
workers: 8      difficulty: 1000         name: th_cpu         time: 0.8794

#####
workers: 1      difficulty: 1000         name: mp_io          time: 0.0460
workers: 2      difficulty: 1000         name: mp_io          time: 0.0389
workers: 4      difficulty: 1000         name: mp_io          time: 0.0365
workers: 8      difficulty: 1000         name: mp_io          time: 0.0429

workers: 1      difficulty: 1000         name: th_io           time: 0.0594
workers: 2      difficulty: 1000         name: th_io           time: 0.1138
workers: 4      difficulty: 1000         name: th_io           time: 0.1544
workers: 8      difficulty: 1000         name: th_io           time: 0.1556
```

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Avant propos / définitions GIL

Parallélisation

Bibliographie

Code natif

Python scientifique

Succès du langage

Performances Threading – Multiprocess

Matthieu Falce

Nb CPU gérés par Ubuntu 18.04 (linux 4.15.0-34-generic)

```
workers: 1      difficulty: 10000      name: mp_cpu      time: 5.4023
workers: 2      difficulty: 10000      name: mp_cpu      time: 3.2131
workers: 4      difficulty: 10000      name: mp_cpu      time: 2.7226
workers: 8      difficulty: 10000      name: mp_cpu      time: 2.8217

workers: 1      difficulty: 10000      name: th_cpu       time: 4.8232
workers: 2      difficulty: 10000      name: th_cpu       time: 5.8625
workers: 4      difficulty: 10000      name: th_cpu       time: 6.3729
workers: 8      difficulty: 10000      name: th_cpu       time: 7.1965

#####
workers: 1      difficulty: 10000      name: mp_io       time: 0.2956
workers: 2      difficulty: 10000      name: mp_io       time: 0.1519
workers: 4      difficulty: 10000      name: mp_io       time: 0.1939
workers: 8      difficulty: 10000      name: mp_io       time: 0.1940

workers: 1      difficulty: 10000      name: th_io        time: 0.3264
workers: 2      difficulty: 10000      name: th_io        time: 0.9546
workers: 4      difficulty: 10000      name: th_io        time: 1.1755
workers: 8      difficulty: 10000      name: th_io        time: 1.3245
```

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Avant propos / définitions GIL

Parallélisation

Bibliographie

Code natif

Python scientifique

Succès du langage

Performances Threading – Multiprocess

Matthieu Falce

Limite à 1 CPU ⁴¹

```
workers: 1      difficulty: 10      name: mp_cpu      time: 0.0194
workers: 2      difficulty: 10      name: mp_cpu      time: 0.0237
workers: 4      difficulty: 10      name: mp_cpu      time: 0.0379
workers: 8      difficulty: 10      name: mp_cpu      time: 0.0496

workers: 1      difficulty: 10      name: th_cpu       time: 0.0347
workers: 2      difficulty: 10      name: th_cpu       time: 0.0162
workers: 4      difficulty: 10      name: th_cpu       time: 0.0088
workers: 8      difficulty: 10      name: th_cpu       time: 0.0133

#####
workers: 1      difficulty: 10      name: mp_io       time: 0.0122
workers: 2      difficulty: 10      name: mp_io       time: 0.0126
workers: 4      difficulty: 10      name: mp_io       time: 0.0318
workers: 8      difficulty: 10      name: mp_io       time: 0.0600

workers: 1      difficulty: 10      name: th_io        time: 0.0022
workers: 2      difficulty: 10      name: th_io        time: 0.0020
workers: 4      difficulty: 10      name: th_io        time: 0.0024
workers: 8      difficulty: 10      name: th_io        time: 0.0037
```

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Avant propos / définitions GIL

Parallélisation

Bibliographie

Code natif

Python scientifique

Succès du langage

41.<http://xmodulo.com/run-program-process-specific-cpu-cores-linux.html>

Performances Threading – Multiprocess

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Avant propos / définitions
GIL

Parallélisation

Bibliographie

Code natif

Python scientifique

Succès du langage

Limite à 1 CPU 41

```
workers: 1      difficulty: 1000      name: mp_cpu      time: 0.7073
workers: 2      difficulty: 1000      name: mp_cpu      time: 0.6151
workers: 4      difficulty: 1000      name: mp_cpu      time: 0.6643
workers: 8      difficulty: 1000      name: mp_cpu      time: 0.8287

workers: 1      difficulty: 1000      name: th_cpu       time: 0.6797
workers: 2      difficulty: 1000      name: th_cpu       time: 0.7070
workers: 4      difficulty: 1000      name: th_cpu       time: 0.7149
workers: 8      difficulty: 1000      name: th_cpu       time: 0.9501

#####
workers: 1      difficulty: 1000      name: mp_io       time: 0.0297
workers: 2      difficulty: 1000      name: mp_io       time: 0.0396
workers: 4      difficulty: 1000      name: mp_io       time: 0.1407
workers: 8      difficulty: 1000      name: mp_io       time: 0.1114

workers: 1      difficulty: 1000      name: th_io        time: 0.0312
workers: 2      difficulty: 1000      name: th_io        time: 0.0992
workers: 4      difficulty: 1000      name: th_io        time: 0.1273
workers: 8      difficulty: 1000      name: th_io        time: 0.1380
```

41.<http://xmodulo.com/run-program-process-specific-cpu-cores-linux.html>

Performances Threading – Multiprocess

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Avant propos / définitions
GIL

Parallélisation

Bibliographie

Code natif

Python scientifique

Succès du langage

Limite à 1 CPU 41

```
workers: 1      difficulty: 10000     name: mp_cpu      time: 8.7052
workers: 2      difficulty: 10000     name: mp_cpu      time: 6.0837
workers: 4      difficulty: 10000     name: mp_cpu      time: 5.2096
workers: 8      difficulty: 10000     name: mp_cpu      time: 5.6368

workers: 1      difficulty: 10000     name: th_cpu       time: 5.1897
workers: 2      difficulty: 10000     name: th_cpu       time: 5.6029
workers: 4      difficulty: 10000     name: th_cpu       time: 4.7810
workers: 8      difficulty: 10000     name: th_cpu       time: 4.6669

#####
workers: 1      difficulty: 10000     name: mp_io       time: 0.2579
workers: 2      difficulty: 10000     name: mp_io       time: 0.3199
workers: 4      difficulty: 10000     name: mp_io       time: 0.3941
workers: 8      difficulty: 10000     name: mp_io       time: 0.2785

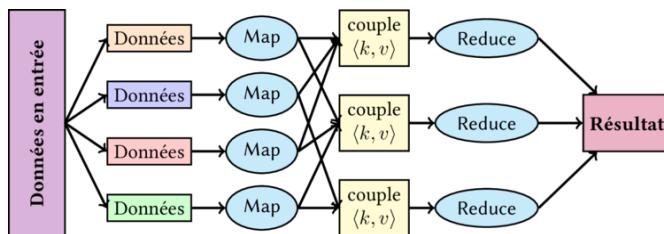
workers: 1      difficulty: 10000     name: th_io        time: 0.2330
workers: 2      difficulty: 10000     name: th_io        time: 0.2845
workers: 4      difficulty: 10000     name: th_io        time: 0.2648
workers: 8      difficulty: 10000     name: th_io        time: 0.3064
```

41.<http://xmodulo.com/run-program-process-specific-cpu-cores-linux.html>

Map Reduce

Matthieu Falce

- ▶ patron d'architecture pour le calcul parallèle (et distribué)
- ▶ utilisé pour traiter de grande quantités de données



Source :

<https://fr.wikipedia.org/wiki/MapReduce#/media/File:Mapreduce.png>

Bibliographie I

Matthieu Falce

- ▶ GIL
 - ▶ <http://www.dabeaz.com/python/UnderstandingGIL.pdf>
 - ▶ [https://en.wikipedia.org/wiki/Preemption_\(computing\)](https://en.wikipedia.org/wiki/Preemption_(computing))
 - ▶ <https://lwn.net/Articles/612483/>

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Avant propos / définitions
GIL
Parallélisation
Bibliographie
Code natif
Python scientifique
Succès du langage

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Ctypes

Cython

Embarquer du code Python
dans du C

Bibliographie

Python scientifique

Succès du langage

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

Création de
modules

Programmation
concurrente

Code natif

Ctypes

Cython

Embarquer du code Python
dans du C

Bibliographie

Python scientifique

Succès du langage

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

Création de modules

Programmation concurrente

Code natif

Ctypes

Cython

Embarquer du code Python dans du C

Bibliographie

Python scientifique

Succès du langage

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

Création de modules

Programmation concurrente

Code natif

Ctypes

Cython

Embarquer du code Python dans du C

Bibliographie

Python scientifique

Succès du langage

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

Création de modules

Programmation concurrente

Code natif

Ctypes

Cython
Embarquer du code Python dans du C

Bibliographie

Python scientifique

Succès du langage

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

Création de modules

Programmation concurrente

Code natif

Ctypes
Cython
Embarquer du code Python dans du C
Bibliographie

Python scientifique

Succès du langage

Ctypes

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Ctypes

Cython

Embarquer du code Python
dans du C

Bibliographie

Python scientifique

Succès du langage

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

Création de
modules

Programmation
concurrente

Code natif

Ctypes

Cython

Embarquer du code Python
dans du C

Bibliographie

Python scientifique

Succès du langage

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

Création de modules

Programmation concurrente

Code natif

Ctypes

Cython

Embarquer du code Python dans du C

Bibliographie

Python scientifique

Succès du langage

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

Création de modules

Programmation concurrente

Code natif

Ctypes

Cython

Embarquer du code Python dans du C

Bibliographie

Python scientifique

Succès du langage

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

Création de modules

Programmation concurrente

Code natif

Ctypes

Cython

Embarquer du code Python dans du C

Bibliographie

Python scientifique

Succès du langage

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

Création de modules

Programmation concurrente

Code natif

Ctypes

Cython

Embarquer du code Python dans du C

Bibliographie

Python scientifique

Succès du langage

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

Création de modules

Programmation concurrente

Code natif

Ctypes

Cython

Embarquer du code Python dans du C

Bibliographie

Python scientifique

Succès du langage

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

Création de modules

Programmation concurrente

Code natif

Ctypes

Cython

Embarquer du code Python dans du C

Bibliographie

Python scientifique

Succès du langage

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

Création de modules

Programmation concurrente

Code natif

Ctypes

Cython

Embarquer du code Python dans du C

Bibliographie

Python scientifique

Succès du langage

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

Création de modules

Programmation concurrente

Code natif

Ctypes

Cython

Embarquer du code Python dans du C

Bibliographie

Python scientifique

Succès du langage

Bibliographie I

- ▶ 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>

Matthieu Falce

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Code natif
Ctypes
Cython
Embarquer du code Python dans du C
Bibliographie
Python scientifique
Succès du langage

Bibliographie II

- ▶ sur Windows : <https://docs.python.org/3/extending/windows.html>
- ▶ <https://docs.python.org/3/extending/building.html>
- ▶ embarquer python dans du C
 - ▶ <https://docs.python.org/3/c-api/>
 - ▶ <https://docs.python.org/3/extending/embedding.html>

Matthieu Falce

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Code natif
Ctypes
Cython
Embarquer du code Python dans du C
Bibliographie
Python scientifique
Succès du langage

Python scientifique

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /
statistiques

Graphiques

Performances

Dask

Bibliographie

Succès du langage

Écosystème

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /
statistiques

Graphiques

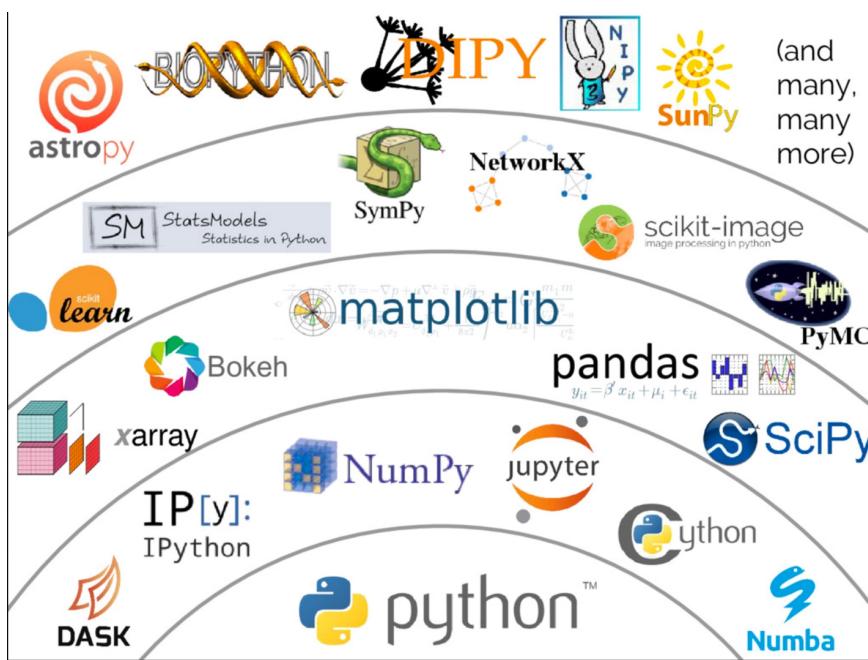
Performances

Dask

Bibliographie

Succès du langage

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

Création de modules

Programmation concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning / statistiques

Graphiques

Performances

Dask

Bibliographie

Succès du langage

Écosystème

Calculs :

- ▶ numpy ⁴²
- ▶ scipy ⁴³
- ▶ pandas ⁴⁴
- ▶ ...

Plotting :

- ▶ matplotlib ⁴⁵
- ▶ seaborn ⁴⁶
- ▶ bokeh ⁴⁷
- ▶ ...

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

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

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

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

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

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

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning / statistiques

Graphiques

Performances

Dask

Bibliographie

Succès du langage

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

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /
statistiques

Graphiques

Performances

Dask

Bibliographie

Succès du langage

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

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /
statistiques

Graphiques

Performances

Dask

Bibliographie

Succès du langage

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

Création de modules

Programmation concurrente

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

Machine learning / statistiques

Tableaux numpy

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

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

Machine learning / statistiques

Même manipulation que Matlab

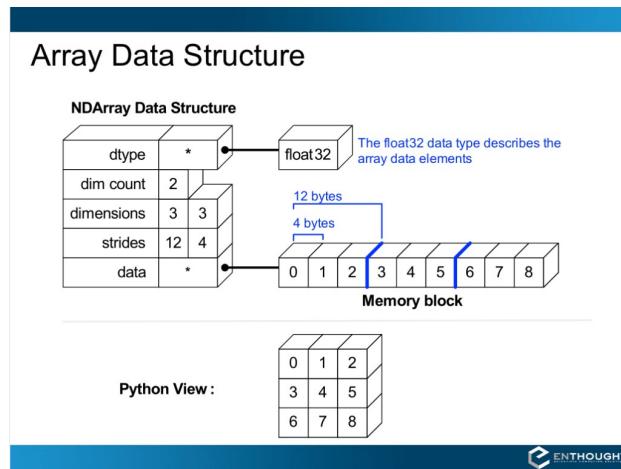
```
import numpy as np
```

```
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

Création de modules

Programmation concurrente

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

Machine learning / statistiques

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

Création de modules

Programmation concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Présentation

Tableaux

Structure des tableaux

Création des tableaux

Manipulations usuelles

Broadcasting

Fonctions universelles

Scipy

Pandas

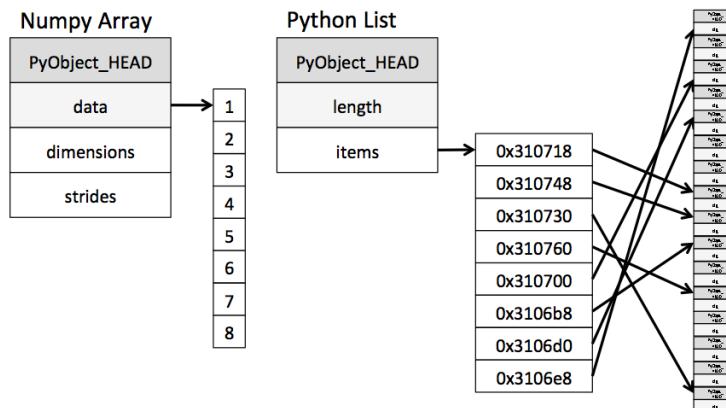
Sympy

Analyse de réseaux

Machine learning / statistiques

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>

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

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

Machine learning / statistiques

Création des tableaux

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

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

Machine learning / statistiques

```
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)
```

Création des tableaux

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

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

Machine learning /
statistiques

Creation			
Code	Result	Code	Result
<code>Z = zeros(9)</code>		<code>Z = zeros((5,9))</code>	
<code>Z = ones(9)</code>		<code>Z = ones((5,9))</code>	
<code>Z = array([0,0,0,0,0,0,0,0,0])</code>		<code>Z = array([[0,0,0,0,0,0,0,0,0], [0,0,0,0,0,0,0,0,0], [0,0,0,0,0,0,0,0,0], [0,0,0,0,0,0,0,0,0], [0,0,0,0,0,0,0,0,0]])</code>	
<code>Z = arange(9)</code>		<code>Z = arange(5*9).reshape(5,9)</code>	
<code>Z = random.uniform(0,1,9)</code>		<code>Z = random.uniform(0,1,(5,9))</code>	

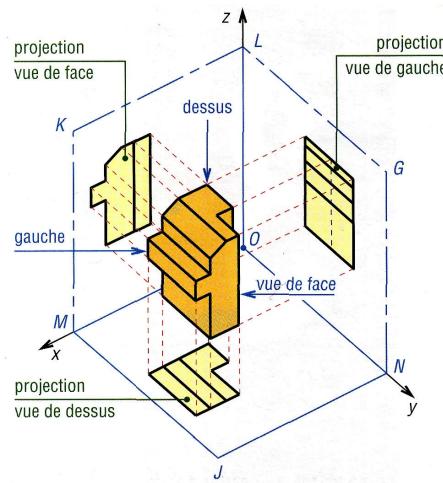
<http://www.labri.fr/perso/nrougier/teaching/numpy/numplib.html>

Changements de forme / dimensions

Matthieu Falce

Certaines opérations vont s'opérer sur certaines dimensions.

On peut les visualiser comme étant des projections.



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

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

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

Machine learning /
statistiques

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

Création de
modules

Programmation
concurrente

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

Machine learning /
statistiques

Changements de forme / dimensions

Matthieu Falce

Reshaping

Code

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



Result

Code

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



Result

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



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



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



<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

Création de modules

Programmation concurrente

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

Machine learning / statistiques

Indexing

Matthieu Falce

Slicing

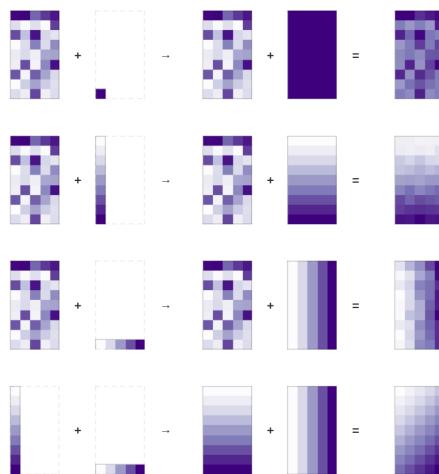
Code	Result	Code	Result
z		z[...]=1	
z[1,1]=1		z[:,0]=1	
z[0,:]=1		z[:,2:]=1	
z[:,::2]=1		z[:,::2,:]=1	
z[:-2,:,:2]=1		z[2:4,2:4]=1	
z[:,2,:,:2]=1		z[3::2,3::2]=1	

<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

Création de modules

Programmation concurrente

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

Machine learning / statistiques

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

Création de modules

Programmation concurrente

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

Machine learning / statistiques

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

Création de modules

Programmation concurrente

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

Machine learning / statistiques

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 Scipy⁴⁸

Scipy utilise LAPACK, Numpy pas forcément⁴⁹

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

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

Scikits

Matthieu Falce

SciPy Toolkits → extensions pour Scipy indépendantes

Liste non exhaustive 50

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

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

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Présentation

Scikits

Fonctionnalités

Pandas

Sympy

Analyse de réseaux

Machine learning /
statistiques

Graphiques

Performances

Dask

Bibliographie

FFT 51

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Présentation

Scikits

Fonctionnalités

Pandas

Sympy

Analyse de réseaux

Machine learning /
statistiques

Graphiques

Performances

Dask

Bibliographie

```
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()
```

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

Optimisation 52

Matthieu Falce

```
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])
```

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

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Présentation

Scikits

Fonctionnalités

Pandas

Sympy

Analyse de réseaux

Machine learning /
statistiques

Graphiques

Performances

Dask

Bibliographie

Optimisation 52

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Présentation

Scikits

Fonctionnalités

Pandas

Sympy

Analyse de réseaux

Machine learning /
statistiques

Graphiques

Performances

Dask

Bibliographie

On peut mettre des contraintes sur les paramètres et
changer de méthode d'optimisation aussi.

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

Interpolation 53

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()
```

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

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Présentation

Scikits

Fonctionnalités

Pandas

Sympy

Analyse de réseaux

Machine learning / statistiques

Graphiques

Performances

Dask

Bibliographie

Intégration – équations différentielles 54 55

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Présentation

Scikits

Fonctionnalités

Pandas

Sympy

Analyse de réseaux

Machine learning / statistiques

Graphiques

Performances

Dask

Bibliographie

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

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

Intégration – équations différentielles 54 55

Matthieu Falce

```
from scipy.integrate import quad

def fonction_a_integrer(x, a, b):
    return a * x ** 3 + b

a = 1
b = 0
surface = quad(
    fonction_a_integrer,
    -10,
    10,
    args=(a, b)
)
print(surface)
```

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

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

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Présentation

Scikits

Fonctionnalités

Pandas

Sympy

Analyse de réseaux

Machine learning /
statistiques

Graphiques

Performances

Dask

Bibliographie

Intégration – équations différentielles 54 55

Matthieu Falce

```
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt

def carre_der(t, x):
    return 2*x

ts = np.arange(10)
ys = odeint(carre_der, 2, ts)
plt.plot(ts, ys)
plt.show()
```

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

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

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Présentation

Scikits

Fonctionnalités

Pandas

Sympy

Analyse de réseaux

Machine learning /
statistiques

Graphiques

Performances

Dask

Bibliographie

Traitement signal⁵⁷

“Tout” pour le traitement du signal :⁵⁶

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

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Présentation

Scikits

Fonctionnalités

Pandas

Sympy

Analyse de réseaux

Machine learning /
statistiques

Graphiques

Performances

Dask

Bibliographie

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

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

Algèbre linéaire⁵⁸



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

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Présentation

Scikits

Fonctionnalités

Pandas

Sympy

Analyse de réseaux

Machine learning /
statistiques

Graphiques

Performances

Dask

Bibliographie

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

Matrices creuses 59 60

Numpy supporte les matrices creuses de différents types :

- ▶ csc_matrix: Compressed Sparse Column format
- ▶ csr_matrix: Compressed Sparse Row format
- ▶ bsr_matrix: Block Sparse Row format
- ▶ lil_matrix: List of Lists format
- ▶ dok_matrix: Dictionary of Keys format
- ▶ coo_matrix: COOrdinate format (aka IJV, triplet format)
- ▶ dia_matrix: DIAGONAL format

Utiliser les méthodes de `scipy.sparse.linalg` pour faire des opérations et garder des matrices creuses. Selon le type de matrices utilisées ; différentes possibilités (perte du slicing...)

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

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

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Présentation

Scikits

Fonctionnalités

Pandas

Sympy

Analyse de réseaux

Machine learning /
statistiques

Graphiques

Performances

Dask

Bibliographie

Lois statistiques 61 62

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Présentation

Scikits

Fonctionnalités

Pandas

Sympy

Analyse de réseaux

Machine learning /
statistiques

Graphiques

Performances

Dask

Bibliographie

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

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

Manipulation d'images

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

Voici les principales bibliothèques pour en faire :

- ▶ *Pillow*⁶³ : plutôt utilisée pour les manipulations classiques (redimensionnement, rotation, changement de format, ...)
- ▶ *numpy / scipy* classique⁶⁴ : permet de manipuler les tableaux de données de pixels directement
- ▶ *scikit image*^{65 66} : Propose de nombreux algorithmes pour faire du traitement d'images
- ▶ *openCV*⁶⁷ : le port de la Bibliothèque d'analyse d'image openCV. Permet de travailler en temps réel (à partir d'une caméra par exemple)

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

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

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

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

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

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Présentation

Scikits

Fonctionnalités

Pandas

Sympy

Analyse de réseaux

Machine learning /
statistiques

Graphiques

Performances

Dask

Bibliographie

Manipulation d'images

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Présentation

Scikits

Fonctionnalités

Pandas

Sympy

Analyse de réseaux

Machine learning /
statistiques

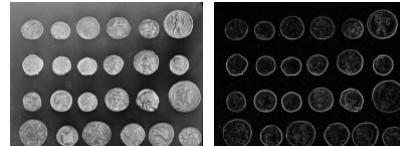
Graphiques

Performances

Dask

Bibliographie

```
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

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Présentation

Scikits

Fonctionnalités

Pandas

Sympy

Analyse de réseaux

Machine learning /
statistiques

Graphiques

Performances

Dask

Bibliographie

Présentation

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Présentation

Dataframes

Manipulations de
dataframes

Sympy

Analyse de réseaux

Machine learning /
statistiques

Graphiques

Performances

Dask

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

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Présentation

Dataframes

Manipulations de
dataframes

Sympy

Analyse de réseaux

Machine learning /
statistiques

Graphiques

Performances

Dask

- ▶ **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

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Présentation

Dataframes

Manipulations de
dataframes

Sympy

Analyse de réseaux

Machine learning /
statistiques

Graphiques

Performances

Dask

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

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Présentation

Dataframes

Manipulations de
dataframes

Sympy

Analyse de réseaux

Machine learning /
statistiques

Graphiques

Performances

Dask

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

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Présentation

Dataframes

Manipulations de
dataframes

Sympy

Analyse de réseaux

Machine learning /
statistiques

Graphiques

Performances

Dask

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

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Présentation

Dataframes

Manipulations de
dataframes

Sympy

Analyse de réseaux

Machine learning /
statistiques

Graphiques

Performances

Dask

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

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Présentation

Dataframes

Manipulations de
dataframes

Sympy

Analyse de réseaux

Machine learning /
statistiques

Graphiques

Performances

Dask

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

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Présentation

Dataframes

Manipulations de
dataframes

Sympy

Analyse de réseaux

Machine learning /
statistiques

Graphiques

Performances

Dask

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

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Présentation

Dataframes

Manipulations de
dataframes

Sympy

Analyse de réseaux

Machine learning /
statistiques

Graphiques

Performances

Dask

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/>

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

Création de modules

Programmation concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Présentation

Dataframes

Manipulations de dataframes

Sympy

Analysé de réseaux

Machine learning / statistiques

Graphiques

Performances

Dask

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

Sympy⁶³ est une bibliothèque permettant le calcul symbolique :

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

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

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Présentation

Dataframes

Manipulations de dataframes

Sympy

Analyse de réseaux

Machine learning / statistiques

Graphiques

Performances

Dask

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning / statistiques

Graphiques

Performances

Dask

Bibliographie

Succès du langage

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

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /
statistiques

Graphiques

Performances

Dask

Bibliographie

Succès du langage

Présentation

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /
statistiques

Graphiques

Performances

Dask

Bibliographie

Succès du langage

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

- ▶ [igraph](http://igraph.org)⁶⁴
- ▶ [networkx](http://networkx.github.io)⁶⁵
- ▶ [graph-tool](http://graph-tool.skewed.de)⁶⁶

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

64.<http://igraph.org>

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

66.<http://graph-tool.skewed.de>

Manipulation de graphes

Matthieu Falce

```
import networkx as nx

# on crée un graphe en 2 parties
G = nx.Graph()
G.add_edges_from([(1, 2), (1, 3)])
G.add_node(4)

# on calcule des propriétés du graphe
nx.connected_components(G)
list(nx.connected_components(G))
sorted(d for n, d in G.degree)
nx.clustering(G)
```

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /
statistiques

Graphiques

Performances

Dask

Bibliographie

Succès du langage

Etat des lieux

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /
statistiques

Graphiques

Performances

Dask

Bibliographie

Succès du langage

Il existe plusieurs bibliothèques dédiées apprentissage :

- ▶ modèles statistiques (régressions, tests statistiques, méthodes sur séries temporelles) : statsmodels⁶⁷
- ▶ algorithmes de *machine learning* : scikit-learn⁶⁸
- ▶ *curve feating* : prophet⁶⁹ (analyse de séries temporelles)
- ▶ *deep learning* : tensorflow⁷⁰, keras⁷¹

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

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

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

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

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

Graphiques

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /
statistiques

Graphiques

Matplotlib

Personnalisation

Seaborn

Concurrents 2D

Graphiques 3D

Performances

1. amélioration et diversification des outils
2. réalisation de graphiques de qualité
3. écosystème riche (seuls R et Javascript ont de meilleures bibliothèques à mon avis)

Matplotlib

Matthieu Falce

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

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /
statistiques

Graphiques

Matplotlib

Personnalisation

Seaborn

Concurrents 2D

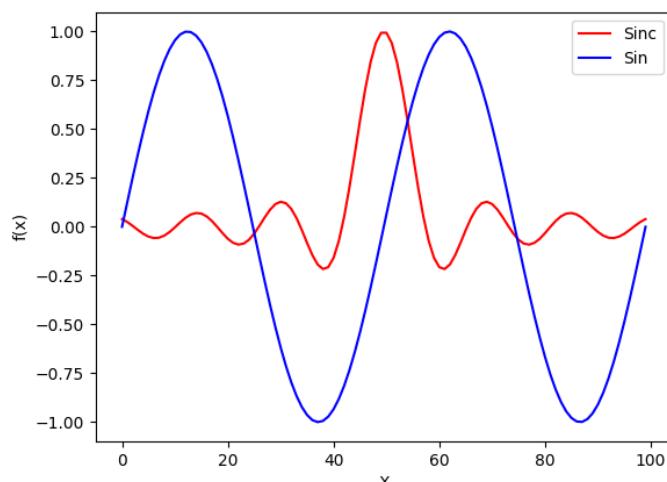
Graphiques 3D

Performances

Matplotlib

Bibliothèques de plot la plus célèbre

API inspirée de Matlab
Le style aussi



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

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /
statistiques

Graphiques

Matplotlib

Personnalisation

Seaborn

Concurrents 2D

Graphiques 3D

Performances

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /
statistiques

Graphiques

Matplotlib

Personnalisation

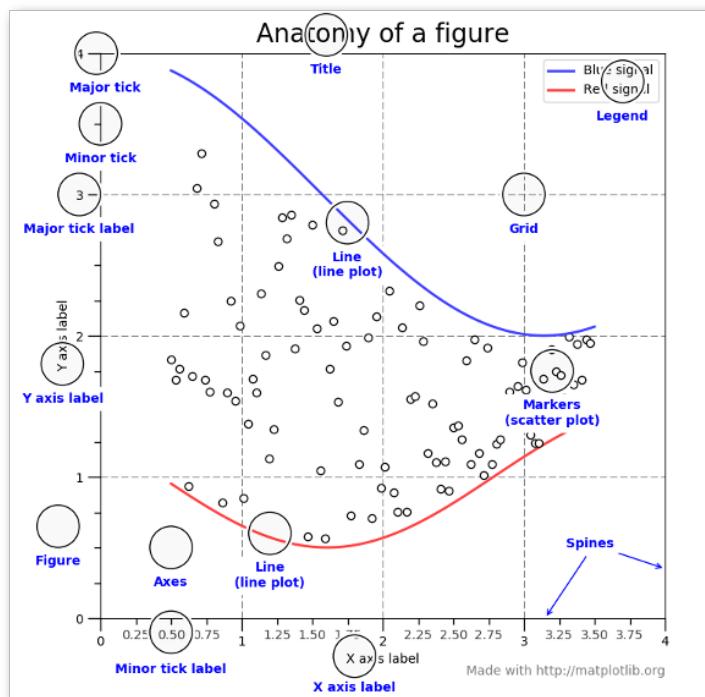
Seaborn

Concurrents 2D

Graphiques 3D

Performances

Anatomie d'une figure



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

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /
statistiques

Graphiques

Matplotlib

Personnalisation

Seaborn

Concurrents 2D

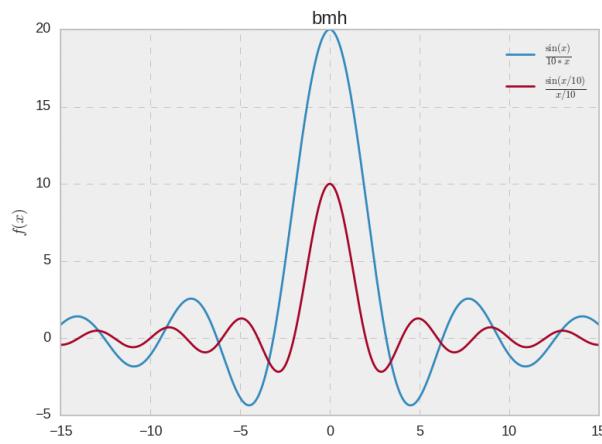
Graphiques 3D

Performances

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

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /
statistiques

Graphiques

Matplotlib

Personnalisation

Seaborn

Concurrents 2D

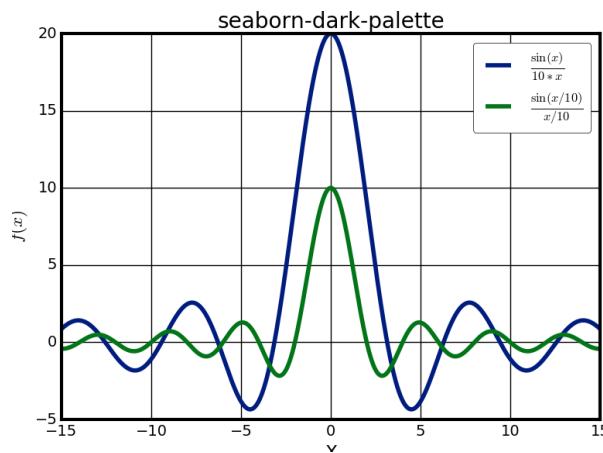
Graphiques 3D

Performances

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

Création de modules

Programmation concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning / statistiques

Graphiques

Matplotlib

Personnalisation

Seaborn

Concurrents 2D

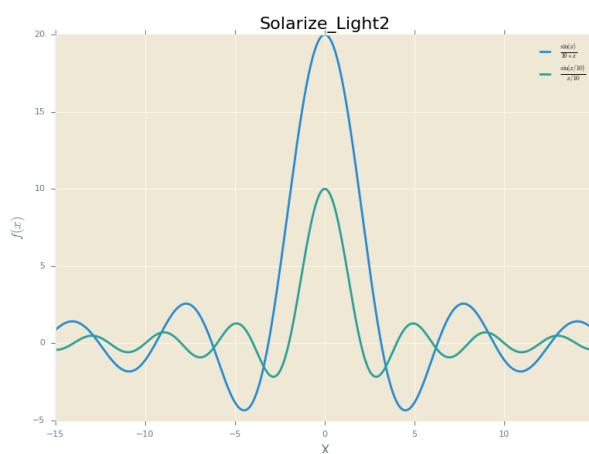
Graphiques 3D

Performances

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

Création de modules

Programmation concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning / statistiques

Graphiques

Matplotlib

Personnalisation

Seaborn

Concurrents 2D

Graphiques 3D

Performances

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 / 10)

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

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /
statistiques

Graphiques

Matplotlib

Personnalisation

Seaborn

Concurrents 2D

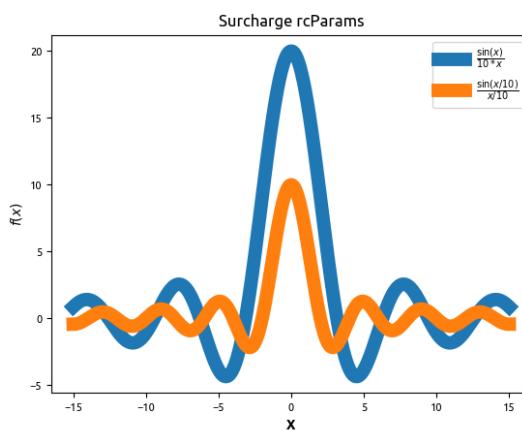
Graphiques 3D

Performances

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

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /
statistiques

Graphiques

Matplotlib

Personnalisation

Seaborn

Concurrents 2D

Graphiques 3D

Performances

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

Création de modules

Programmation concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning / statistiques

Graphiques

Matplotlib

Personnalisation

Seaborn

Concurrents 2D

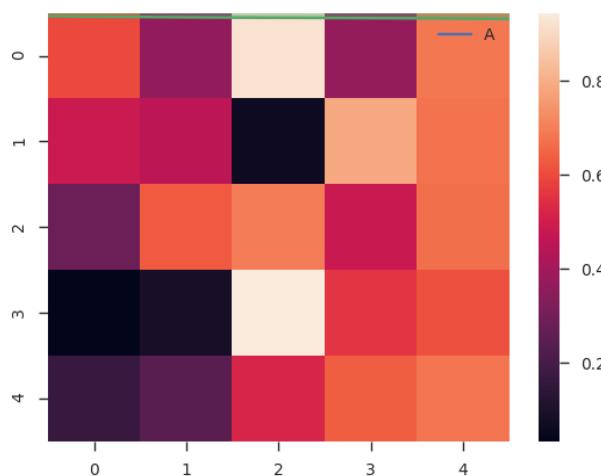
Graphiques 3D

Performances

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

Création de modules

Programmation concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning / statistiques

Graphiques

Matplotlib

Personnalisation

Seaborn

Concurrents 2D

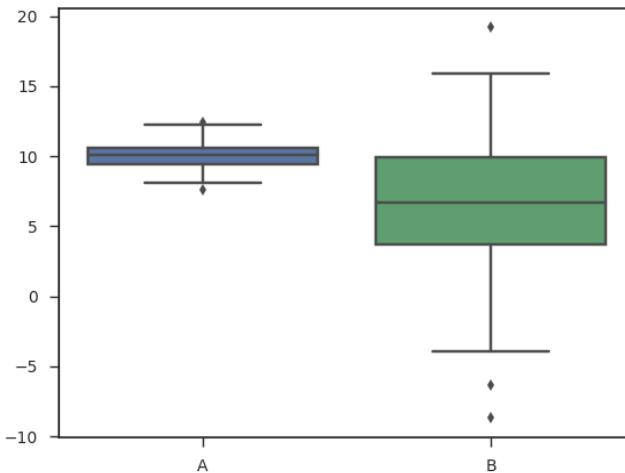
Graphiques 3D

Performances

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

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /
statistiques

Graphiques

Matplotlib

Personnalisation

Seaborn

Concurrents 2D

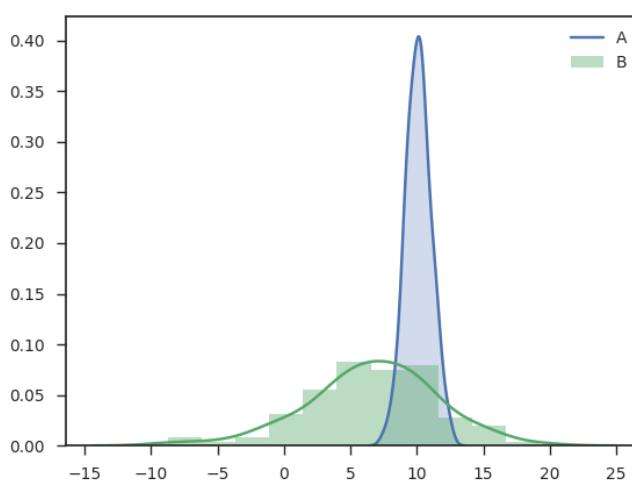
Graphiques 3D

Performances

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

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /
statistiques

Graphiques

Matplotlib

Personnalisation

Seaborn

Concurrents 2D

Graphiques 3D

Performances

Concurrents 2D

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /
statistiques

Graphiques

Matplotlib

Personnalisation

Seaborn

Concurrents 2D

Graphiques 3D

Performances

Matplotlib n'est pas la seule bibliothèque de graphiques pour Python :

- ▶ (seaborn)
- ▶ [plotly](#) (figures web interactives)
- ▶ [mpld3](#) (transforme une figure mpl en Javascript)
- ▶ [bokeh](#) (figures web interactive)
- ▶ [plotly](#) (figures web interactive)
- ▶ [ggplot](#) (port de la bibliothèque ggplot2 de R)

Concurrent : *plotly*

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /
statistiques

Graphiques

Matplotlib

Personnalisation

Seaborn

Concurrents 2D

Graphiques 3D

Performances

```
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()
```

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

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /
statistiques

Graphiques

Matplotlib

Personnalisation

Seaborn

Concurrents 2D

Graphiques 3D

Performances

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

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /
statistiques

Graphiques

Matplotlib

Personnalisation

Seaborn

Concurrents 2D

Graphiques 3D

Performances

Dashboards

Matthieu Falce

On peut utiliser Dash⁷²

- ▶ 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/>

72.<https://plot.ly/dash/>

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning / statistiques

Graphiques

Matplotlib

Personnalisation

Seaborn

Concurrents 2D

Graphiques 3D

Performances

Graphiques 3D

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning / statistiques

Graphiques

Matplotlib

Personnalisation

Seaborn

Concurrents 2D

Graphiques 3D

Performances

```
# 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()
```

Graphiques 3D

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /
statistiques

Graphiques

Matplotlib

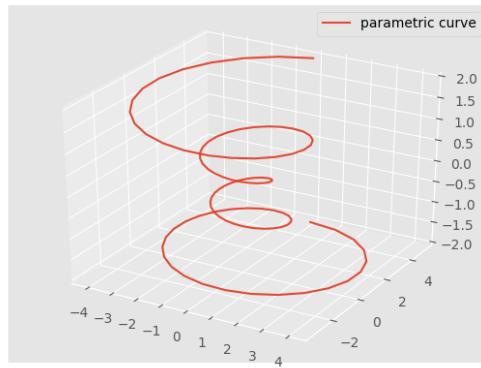
Personnalisation

Seaborn

Concurrents 2D

Graphiques 3D

Performances



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

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /
statistiques

Graphiques

Matplotlib

Personnalisation

Seaborn

Concurrents 2D

Graphiques 3D

Performances

Pour faire de la vraie 3d :

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

Avant-propos



N'optimisez que ce qui est nécessaire

- ▶ faîtes 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
- ▶ ...

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /
statistiques

Graphiques

Performances

Numexpr

Dask

Bibliographie

Succès du langage

Numexpr⁷³

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)
```

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

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /
statistiques

Graphiques

Performances

Numexpr

Dask

Bibliographie

Succès du langage

Dask 74

Matthieu Falce

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

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

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning / statistiques

Graphiques

Performances

Dask

Autres

Bibliographie

Succès du langage

Dask

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning / statistiques

Graphiques

Performances

Dask

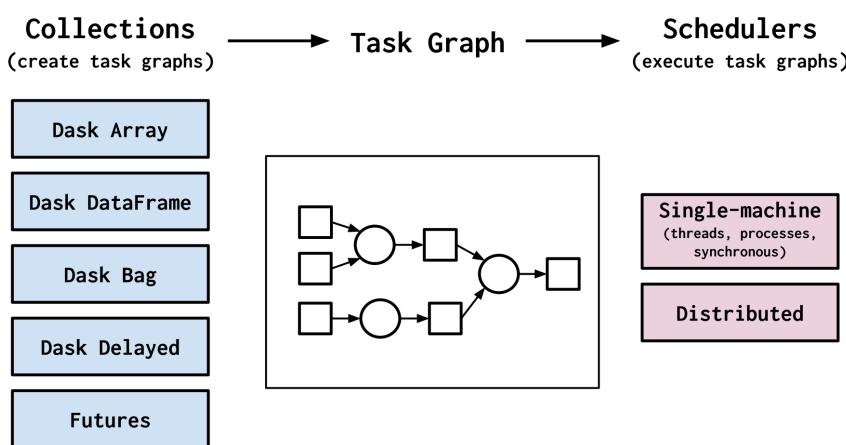
Autres

Bibliographie

Succès du langage

Dask : vue d'ensemble

Source: <https://docs.dask.org/en/latest/>



Dask

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Écosystème scientifique
Jupyter
Numpy
Scipy
Pandas
Sympy

Analyse de réseaux
Machine learning /
statistiques

Graphiques
Performances

Dask

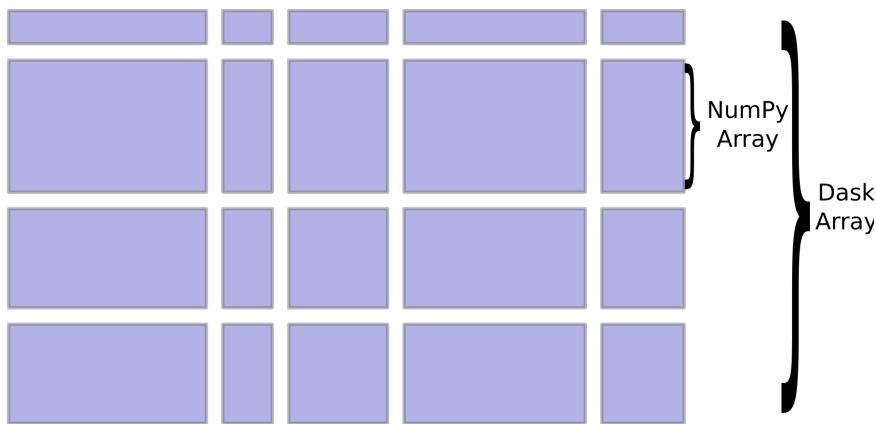
Autres

Bibliographie

Succès du langage

Dask : structure d'un *dask array*

Source: <https://docs.dask.org/en/latest/array.html>



Dask

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Écosystème scientifique
Jupyter
Numpy
Scipy
Pandas
Sympy

Analyse de réseaux
Machine learning /
statistiques

Graphiques
Performances

Dask

Autres

Bibliographie

Succès du langage

```
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
```

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

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /
statistiques

Graphiques

Performances

Dask

Autres

Bibliographie

Succès du langage

Autres techniques

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /
statistiques

Graphiques

Performances

Dask

Autres

Bibliographie

Succès du langage

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/internals.html>

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning / statistiques

Graphiques

Performances

Dask

Bibliographie

Succès du langage

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(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

Création de modules

Programmation concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning / statistiques

Graphiques

Performances

Dask

Bibliographie

Succès du langage

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

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Écosystème scientifique
Jupyter
NumPy
SciPy
Pandas
Sympy

Analyse de réseaux

Machine learning /
statistiques

Graphiques

Performances

Dask

Bibliographie

Succès du langage

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

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Écosystème scientifique
Jupyter
NumPy
SciPy
Pandas
Sympy

Analyse de réseaux

Machine learning /
statistiques

Graphiques

Performances

Dask

Bibliographie

Succès du langage

Bibliographie V

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Écosystème scientifique

Jupyter

Numpy

Scipy

Pandas

Sympy

Analyse de réseaux

Machine learning /
statistiques

Graphiques

Performances

Dask

Bibliographie

Succès du langage

Succès du langage

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Expressions régulières

Base de données

XML

JSON

Réseau

Calcul distribué et
asynchrone

Écosystème scientifique

Jupyter

Pandas

Intelligence artificielle

Expressions Régulières ?

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Expressions régulières

Base de données

XML

JSON

Réseau

Calcul distribué et
asynchrone

Écosystème scientifique

Jupyter

Pandas

Intelligence artificielle

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

https://fr.wikipedia.org/wiki/Expression_r%C3%A9gul%C3%A8re

Syntaxe

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Expressions régulières

Base de données

XML

JSON

Réseau

Calcul distribué et
asynchrone

Écosystème scientifique

Jupyter

Pandas

Intelligence artificielle

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

Exemples

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Expressions régulières

Base de données

XML

JSON

Réseau

Calcul distribué et asynchrone

Écosystème scientifique

Jupyter

Pandas

Intelligence artificielle

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

Exemples

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Expressions régulières

Base de données

XML

JSON

Réseau

Calcul distribué et asynchrone

Écosystème scientifique

Jupyter

Pandas

Intelligence artificielle

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 / ""

Cas d'usages

Matthieu Falce

Quand les utiliser :

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

Quand ne pas les utiliser :

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

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Expressions régulières

Base de données

XML

JSON

Réseau

Calcul distribué et asynchrone

Écosystème scientifique

Jupyter

Pandas

Intelligence artificielle

En python

Matthieu Falce

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)

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Expressions régulières

Base de données

XML

JSON

Réseau

Calcul distribué et asynchrone

Écosystème scientifique

Jupyter

Pandas

Intelligence artificielle

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

En python

Matthieu Falce

```
import re

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

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

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Expressions régulières

Base de données

XML

JSON

Réseau

Calcul distribué et
asynchrone

Écosystème scientifique

Jupyter

Pandas

Intelligence artificielle

En python

Matthieu Falce

```
import re

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

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

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

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

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

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

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Expressions régulières

Base de données

XML

JSON

Réseau

Calcul distribué et
asynchrone

Écosystème scientifique

Jupyter

Pandas

Intelligence artificielle

Accès aux bases de données

Matthieu Falce

- ▶ Python permet de se connecter à des bases de données
- ▶ Normalisation avec la DB API (database API)⁷⁵
 - ▶ comme un pilote d'imprimante ⇒ on lui dit ce qu'on veut imprimer, il s'occupe des spécificités
 - ▶ augmente la compréhension du code
 - ▶ facilite le changement de SGBD
 - ▶ inspirée de Open Database Connectivity (ODBC) et Java Database Connectivity (JDBC)

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

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Expressions régulières

Base de données

XML

JSON

Réseau

Calcul distribué et asynchrone

Écosystème scientifique

Jupyter

Pandas

Intelligence artificielle

Présentation DB API

Matthieu Falce

Avec SQLite

```
import sqlite3

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

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

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

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

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

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

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Expressions régulières

Base de données

XML

JSON

Réseau

Calcul distribué et asynchrone

Écosystème scientifique

Jupyter

Pandas

Intelligence artificielle

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

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Expressions régulières

Base de données

XML

JSON

Réseau

Calcul distribué et asynchrone

Écosystème scientifique

Jupyter

Pandas

Intelligence artificielle

Présentation DB API

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

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Expressions régulières

Base de données

XML

JSON

Réseau

Calcul distribué et asynchrone

Écosystème scientifique

Jupyter

Pandas

Intelligence artificielle

Insérer / récupérer des données

```
import sqlite3

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

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

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

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

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Expressions régulières

Base de données

XML

JSON

Réseau

Calcul distribué et
asynchrone

Écosystème scientifique

Jupyter

Pandas

Intelligence artificielle

Supprimer / mettre à jour des données

```
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()
```

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Expressions régulières

Base de données

XML

JSON

Réseau

Calcul distribué et
asynchrone

Écosystème scientifique

Jupyter

Pandas

Intelligence artificielle

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

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Expressions régulières

Base de données

XML

JSON

Réseau

Calcul distribué et asynchrone

Écosystème scientifique

Jupyter

Pandas

Intelligence artificielle

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

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Expressions régulières

Base de données

XML

JSON

Réseau

Calcul distribué et asynchrone

Écosystème scientifique

Jupyter

Pandas

Intelligence artificielle

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

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Expressions régulières

Base de données

XML

JSON

Réseau

Calcul distribué et
asynchrone

Écosystème scientifique

Jupyter

Pandas

Intelligence artificielle

Bibliographie / Aller plus loin

Matthieu Falce

- ▶ <https://wiki.python.org/moin/DbApiCheatSheet>
- ▶ <http://sweetohm.net/article/python-dbapi.html>
- ▶ <https://apprendre-python.com/page-database-database-donnees-query-sql-mysql-postgre-sqlite>
- ▶ <https://www.sqlitetutorial.net/sqlite-python/>
- ▶ comment gérer le *multithreading* ?
 - ▶ curseurs non *thread safe*
 - ▶ une connexion par thread
- ▶ ORM ⁷⁶ ⇒ abstraire les différences entre moteurs
 - ▶ SQLAlchemy
 - ▶ Pewee
 - ▶ PonyORM
 - ▶ ORM Django

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Expressions régulières

Base de données

XML

JSON

Réseau

Calcul distribué et
asynchrone

Écosystème scientifique

Jupyter

Pandas

Intelligence artificielle

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

XML 77

Matthieu Falce

```
import xml.etree.ElementTree 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")
```

77.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

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Expressions régulières

Base de données

XML

JSON

Réseau

Calcul distribué et asynchrone

Écosystème scientifique

Jupyter

Pandas

Intelligence artificielle

XML 77

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

77.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

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Expressions régulières

Base de données

XML

JSON

Réseau

Calcul distribué et asynchrone

Écosystème scientifique

Jupyter

Pandas

Intelligence artificielle

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

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Expressions régulières

Base de données

XML

JSON

Réseau

Calcul distribué et
asynchrone

Écosystème scientifique

Jupyter

Pandas

Intelligence artificielle

JSON

Matthieu Falce



Certaines données ne sont pas JSON sérialisables. Il faut créer son propre serialiseur JSON dans ce cas.⁷⁸

```
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))
```

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Expressions régulières

Base de données

XML

JSON

Réseau

Calcul distribué et
asynchrone

Écosystème scientifique

Jupyter

Pandas

Intelligence artificielle

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

CSV – excel

Matthieu Falce

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

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

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

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Expressions régulières

Base de données

XML

JSON

Réseau

Calcul distribué et
asynchrone

Écosystème scientifique

Jupyter

Pandas

Intelligence artificielle

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

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Expressions régulières

Base de données

XML

JSON

Réseau

Calcul distribué et
asynchrone

Écosystème scientifique

Jupyter

Pandas

Intelligence artificielle

CSV – excel

On peut utiliser `xlrd`, `openpyxl` ou `pandas` (qui se base sur ces dernières) ⁷⁹

```
# 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
)
```

[79.<http://www.python-excel.org/>](http://www.python-excel.org/)

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Expressions régulières

Base de données

XML

JSON

Réseau

Calcul distribué et
asynchrone

Écosystème scientifique

Jupyter

Pandas

Intelligence artificielle

Inventaire

Bas niveau :

- ▶ `twisted` ⁸⁰
- ▶ `ZMQ` ⁸¹
- ▶ tous les protocoles (`PySNMP`⁸², ...)

Haut niveau (framework web):

- ▶ `django` ⁸³
- ▶ `flask` ⁸⁴
- ▶ `pyramid` ⁸⁵
- ▶ ...

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Expressions régulières

Base de données

XML

JSON

Réseau

Calcul distribué et
asynchrone

Écosystème scientifique

Jupyter

Pandas

Intelligence artificielle

[80.<https://twistedmatrix.com/trac/>](https://twistedmatrix.com/trac/)

[81.<http://zeromq.org/>](http://zeromq.org/)

[82.<http://snmplabs.com/pysnmp/index.html>](http://snmplabs.com/pysnmp/index.html)

[83.\[https://www.djangoproject.com/\]\(http://www.djangoproject.com/\)](http://www.djangoproject.com/)

[84.<http://flask.pocoo.org/>](http://flask.pocoo.org/)

[85.\[https://trypyramid.com/\]\(http://trypyramid.com/\)](http://trypyramid.com/)

Inventaire

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Expressions régulières

Base de données

XML

JSON

Réseau

Calcul distribué et
asynchrone

Écosystème scientifique

Jupyter

Pandas

Intelligence artificielle

Asynchrone :

- ▶ `asyncio / trio` ⁸⁰
- ▶ `gevent` ⁸¹
- ▶ `tornado` ⁸²

80.<https://github.com/python-trio/trio>

81.<http://www.gevent.org/>

82.<https://www.tornadoweb.org/en/stable/>

Requêtes HTTP

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Expressions régulières

Base de données

XML

JSON

Réseau

Calcul distribué et
asynchrone

Écosystème scientifique

Jupyter

Pandas

Intelligence artificielle

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())
```

Twisted

Matthieu Falce

```
from twisted.internet.protocol import Protocol, Factory
from twisted.internet.endpoints import TCP4ClientEndpoint
from twisted.internet import reactor

class serverprotocol(Protocol):
    def dataReceived(self,data):
        print("[+] got \n" + data.decode())
        def clientProtocol():
            return Clientp(data)
        endpoint = TCP4ClientEndpoint(reactor, "127.0.0.1", 8080)
        endpoint.connect(Factory.forProtocol(clientProtocol))

class Clientp(Protocol):
    def __init__(self, dataToSend):
        self.dataToSend = dataToSend

    def connectionMade(self):
        self.transport.write(self.dataToSend)

    def dataReceived(self,data):
        print("+ got reply" + data.decode())

reactor.listenTCP(3333,
                  Factory.forProtocol(serverprotocol))
reactor.run()
```

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Expressions régulières

Base de données

XML

JSON

Réseau

Calcul distribué et asynchrone

Écosystème scientifique

Jupyter

Pandas

Intelligence artificielle

PySNMP

Matthieu Falce

```
# source: http://snmplabs.com/pysnmp/quick-start.html
from pysnmp.hlapi import *

errorIndication, errorStatus, errorIndex, varBinds = next(
    getCmd(SnmpEngine(),
           CommunityData('public', mpModel=0),
           UdpTransportTarget(('demo.snmplabs.com', 161)),
           ContextData(),
           ObjectType(ObjectIdentity('SNMPv2-MIB', 'sysDescr', 0)))
)

if errorIndication:
    print(errorIndication)
elif errorStatus:
    print('%s at %s' % (errorStatus.prettyPrint(),
                        errorIndex and varBinds[int(errorIndex) - 1][0] or '?'))
else:
    for varBind in varBinds:
        print(' = '.join([x.prettyPrint() for x in varBind]))
```

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Expressions régulières

Base de données

XML

JSON

Réseau

Calcul distribué et asynchrone

Écosystème scientifique

Jupyter

Pandas

Intelligence artificielle

Flask

Matthieu Falce

```
from flask import Flask, request
app = Flask(__name__)

@app.route("/")
def index():
    return "index"

@app.route('/hello/', defaults={'username': None}, methods=['GET'])
@app.route('/hello/<username>', methods=['GET'])
def hello(username=None):
    res = 'Hello, World'
    if username:
        res = "Hello, {}".format(username)
    if request.args.get("u"):
        res = res.upper()
    return res

def main():
    app.run(port=9898, debug=True)

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

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Expressions régulières

Base de données

XML

JSON

Réseau

Calcul distribué et
asynchrone

Écosystème scientifique

Jupyter

Pandas

Intelligence artificielle

Serveur web

Matthieu Falce

On peut aussi lancer un serveur web vite fait sur sa machine :
`python -m http.server`

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Expressions régulières

Base de données

XML

JSON

Réseau

Calcul distribué et
asynchrone

Écosystème scientifique

Jupyter

Pandas

Intelligence artificielle

Inventaire

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Expressions régulières

Base de données

XML

JSON

Réseau

Calcul distribué et
asynchrone

Écosystème scientifique

Jupyter

Pandas

Intelligence artificielle

Calcul distribué :

- ▶ génériques
 - ▶ celery⁸³
 - ▶ redis queue⁸⁴
- ▶ scientifiques
 - ▶ dask distributed⁸⁵

83.<http://www.celeryproject.org/>

84.<http://python-rq.org/>

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

Celery

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Expressions régulières

Base de données

XML

JSON

Réseau

Calcul distribué et
asynchrone

Écosystème scientifique

Jupyter

Pandas

Intelligence artificielle

```
import time
from celery import Celery

app = Celery(
    'tasks',
    backend='redis://localhost',
    broker='redis://localhost'
)

@app.task
def add(x, y):
    print("dans la tâche add")
    time.sleep(2)
    return x + y + 100
```

Celery

Utilisation des fonctions

```
import time
from fonctions_metier import add

def execute(nb):
    print(time.time(), "avant le délai")
    futures = []
    for val in range(nb):
        futures.append(add.delay(val, 20))
    print(time.time(), "après le délai")
    return futures

def get_results(futures):
    results = []
    for future in futures:
        print(time.time(), "avant le get")
        res = future.get()
        print(time.time(), "après le get")
        results.append(res)
    return results

if __name__ == "__main__":
    print(time.time(), "avant execute")
    futures = execute(6)
    print("*****")
    print(time.time(), "avant get results")
    results = get_results(futures)
    print(time.time(), results)
```

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Expressions régulières

Base de données

XML

JSON

Réseau

Calcul distribué et
asynchrone

Écosystème scientifique

Jupyter

Pandas

Intelligence artificielle

Celery

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Expressions régulières

Base de données

XML

JSON

Réseau

Calcul distribué et
asynchrone

Écosystème scientifique

Jupyter

Pandas

Intelligence artificielle

```
# code pour lancer le master qui va lancer 4 workers par défaut
celery -A fonctions_metier worker --loglevel=info
```

```
# dans un autre shell
python celery_getting_started.py
```

Écosystème

- ▶ écosystème très riche
- ▶ utilisé aussi bien par les scientifiques que les entreprises
- ▶ origine : développeurs et scientifiques
- ▶ sponsorisé par de grandes entreprises (google, facebook, Enthought, Anaconda Inc)
- ▶ tout ce que vous cherchez existe probablement déjà

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Expressions régulières

Base de données

XML

JSON

Réseau

Calcul distribué et asynchrone

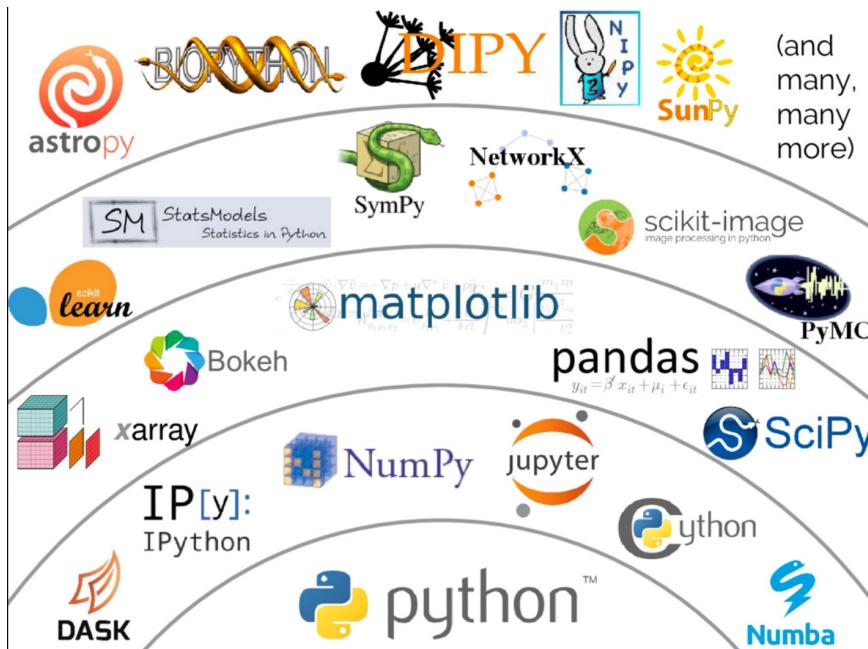
Écosystème scientifique

Jupyter

Pandas

Intelligence artificielle

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

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Expressions régulières

Base de données

XML

JSON

Réseau

Calcul distribué et asynchrone

Écosystème scientifique

Jupyter

Pandas

Intelligence artificielle

Écosystème

Calculs :

- ▶ **numpy** ⁸⁶
- ▶ **scipy** ⁸⁷
- ▶ **pandas** ⁸⁸
- ▶ ...

Plotting :

- ▶ **matplotlib** ⁸⁹
- ▶ **seaborn** ⁹⁰
- ▶ **bokeh** ⁹¹
- ▶ ...

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

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

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

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

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

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

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Expressions régulières

Base de données

XML

JSON

Réseau

Calcul distribué et asynchrone

Écosystème scientifique

Jupyter

Pandas

Intelligence artificielle

Python scientifique

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Expressions régulières

Base de données

XML

JSON

Réseau

Calcul distribué et asynchrone

Écosystème scientifique

Jupyter

Pandas

Intelligence artificielle

```
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))
```

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

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Expressions régulières

Base de données

XML

JSON

Réseau

Calcul distribué et asynchrone

Écosystème scientifique

Jupyter

Pandas

Intelligence artificielle

Numpy

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Expressions régulières

Base de données

XML

JSON

Réseau

Calcul distribué et asynchrone

Écosystème scientifique

Jupyter

Pandas

Intelligence artificielle

```
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))
```

Présentation

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Expressions régulières

Base de données

XML

JSON

Réseau

Calcul distribué et
asynchrone

Écosystème scientifique

Jupyter

Pandas

Présentation

Dataframes

Manipulations de
dataframes

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

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Expressions régulières

Base de données

XML

JSON

Réseau

Calcul distribué et
asynchrone

Écosystème scientifique

Jupyter

Pandas

Présentation

Dataframes

Manipulations de
dataframes

Manipulations de dataframes

Matthieu Falce

Création de dataframes et de séries

```
import numpy as np
import pandas as pd

# créer une série
s = pd.Series(np.random.randn(5), index=["a", "b", "c", "d", "e"])

# créer un dataframe
d = {
    "one": pd.Series([1.0, 2.0, 3.0], index=["a", "b", "c"]),
    "two": pd.Series([1.0, 2.0, 3.0, 4.0], index=["a", "b", "c", "d"]),
}
df = pd.DataFrame(d)
df["three"] = s

print("description de df :")
df.describe()
```

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Expressions régulières

Base de données

XML

JSON

Réseau

Calcul distribué et asynchrone

Écosystème scientifique

Jupyter

Pandas

Présentation

Dataframes

Manipulations de dataframes

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

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Expressions régulières

Base de données

XML

JSON

Réseau

Calcul distribué et asynchrone

Écosystème scientifique

Jupyter

Pandas

Présentation

Dataframes

Manipulations de dataframes

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
Création de modules
Programmation concurrente
Code natif
Python scientifique
Succès du langage
Expressions régulières
Base de données
XML
JSON
Réseau
Calcul distribué et asynchrone
Écosystème scientifique
Jupyter
Pandas
Présentation
Dataframes
Manipulations de dataframes

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
Création de modules
Programmation concurrente
Code natif
Python scientifique
Succès du langage
Expressions régulières
Base de données
XML
JSON
Réseau
Calcul distribué et asynchrone
Écosystème scientifique
Jupyter
Pandas
Présentation
Dataframes
Manipulations de dataframes

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

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Expressions régulières

Base de données

XML

JSON

Réseau

Calcul distribué et
asynchrone

Écosystème scientifique

Jupyter

Pandas

Présentation

Dataframes

Manipulations de
dataframes

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

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Expressions régulières

Base de données

XML

JSON

Réseau

Calcul distribué et
asynchrone

Écosystème scientifique

Jupyter

Pandas

Présentation

Dataframes

Manipulations de
dataframes

Manipulations de dataframes

Manipulation de séries temporelles

```
import pandas as pd
import numpy as np

# time index
dti = pd.date_range("2018-01-13", periods=3, freq="H")
dti = dti.tz_localize("UTC")
dti.tz_convert("US/Pacific")

## offsets
# https://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html#timeseries-offset-aliases
start, end = "2019-01-12", "2019-12-25"
pd.date_range(start, end, freq="BM")

# conversion
## https://docs.python.org/3/library/datetime.html#strftime-and-strptime-behavior
pd.to_datetime("12-11-2010 00:00", format="%d-%m-%Y %H:%M")

# resampling
idx = pd.date_range("2018-01-01", periods=48, freq="H")
ts = pd.Series(range(len(idx)), index=idx)
ts.resample("2H").mean()

s = pd.Series(range(len(idx)), index=idx)
for i in s.resample("6H"):
    print(i)
```

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Expressions régulières

Base de données

XML

JSON

Réseau

Calcul distribué et asynchrone

Écosystème scientifique

Jupyter

Pandas

Présentation

Dataframes

Manipulations de dataframes

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

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Expressions régulières

Base de données

XML

JSON

Réseau

Calcul distribué et asynchrone

Écosystème scientifique

Jupyter

Pandas

Présentation

Dataframes

Manipulations de dataframes

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]	1.764052	0.400157	0.978738	2.240893
df.loc['2000-01-01']	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

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Expressions régulières

Base de données

XML

JSON

Réseau

Calcul distribué et asynchrone

Écosystème scientifique

Jupyter

Pandas

Présentation

Dataframes

Manipulations de dataframes

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

© 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

Création de modules

Programmation concurrente

Code natif

Python scientifique

Succès du langage

Expressions régulières

Base de données

XML

JSON

Réseau

Calcul distribué et asynchrone

Écosystème scientifique

Jupyter

Pandas

Présentation

Dataframes

Manipulations de dataframes

Inventaire

Matthieu Falce

IA :

- ▶ **sklearn** 92
- ▶ **tensorflow** 93
- ▶ ...

92.<https://scikit-learn.org>
93.<https://www.tensorflow.org/>

Vue d'ensemble

Langage Python

Programmation
Orientée objet
(POO)

Bonnes pratiques

Création de
modules

Programmation
concurrente

Code natif

Python scientifique

Succès du langage

Expressions régulières

Base de données

XML

JSON

Réseau

Calcul distribué et
asynchrone

Écosystème scientifique

Jupyter

Pandas

Intelligence artificielle

sklearn

Matthieu Falce

```
# Source :  
# http://scikit-learn.org/stable/auto_examples/cluster/plot_ward_structured_vs_unstructured.html  
  
import time as time  
import numpy as np  
import matplotlib.pyplot as plt  
import mpl_toolkits.mplot3d.axes3d as p3  
from sklearn.cluster import AgglomerativeClustering  
from sklearn.datasets.samples_generator import make_swiss_roll  
  
# Generate data (swiss roll dataset)  
n_samples = 1500  
noise = 0.05  
X, _ = make_swiss_roll(n_samples, noise)  
# Make it thinner  
X[:, 1] *= .5  
  
# Compute clustering  
print("Compute unstructured hierarchical clustering...")  
st = time.time()  
ward = AgglomerativeClustering(n_clusters=6, linkage='ward').fit(X)  
elapsed_time = time.time() - st  
label = ward.labels_  
  
# Plot result  
fig = plt.figure()  
ax = p3.Axes3D(fig)  
ax.view_init(7, -80)  
for l in np.unique(label):  
    ax.scatter(X[label == l, 0], X[label == l, 1], X[label == l, 2],  
               color=plt.cm.jet(np.float(l) / np.max(label + 1)),  
               s=20, edgecolor='k')  
plt.title('Without connectivity constraints (time %.2fs)' % elapsed_time)  
plt.show()
```