Formation Python, perfectionnement

Matthieu Falce

Septembre 2024

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Succès du langage

Au programme I

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Création de modules

Programmation concurrente

Succès du langage

A propos de moi – Qui suis-je ?

- Qui suis-je ?
  - Matthieu Falce
  - habite à Lille
  - ingénieur en bioinformatique (INSA Lyon)

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Succès du langage

A propos de moi – Qui suis-je ?

- 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

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*

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*

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)

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

Où me trouver ?

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

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Succès du langage

Vue d'ensemble

Vue d'ensemble
Historique
Philosophie
Python, CPython, ...
Cas d'utilisations de python
Installation
Environnement de développement
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Succès du langage

1- Vue d'ensemble

1.1. Historique

Vue d'ensemble
Historique
Philosophie
Python, CPython, ...
Cas d'utilisations de python
Installation
Environnement de développement
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Succès du langage

Un vieux langage ?

- Créateur (et bdfl) : Guido van Rossum
- 1ère version : 20 février 1991
- dernière version stable sortie : 3.12.0 (2 octobre 2023)

Un vieux langage ?

- Créateur (et bdfl) : Guido van Rossum
- 1ère version : 20 février 1991
- dernière version stable sortie : 3.12.0 (2 octobre 2023)

Source: `http://pypl.github.io/PYPL.html`

Un vieux langage ?

- Créateur (et bdfl) : Guido van Rossum
- 1ère version : 20 février 1991
- dernière version stable sortie : 3.12.0 (2 octobre 2023)

Source: `http://pypl.github.io/PYPL.html`

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

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
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())
```

Rétrocompatibilité

- 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

Rétrocompatibilité

- 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)

    ...
```

Python 2 vs Python 3

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
Succès du langage

Python 2 vs Python 3

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
Succès du langage

Python 2 End Of Life

Fin du support de Python le 1er janvier 2020

Vue d'ensemble
Historique
Philosophie
Python, CPython, ...
Cas d'utilisations de python
Installation
Environnement de développement
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
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/

1- Vue d'ensemble

1.2. Philosophie

Vue d'ensemble
Historique
Philosophie
Python, CPython, ...
Cas d'utilisations de python
Installation
Environnement de développement
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Succès du langage

Zen of Python

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

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
Succès du langage

Zen of Python (PEP 20 [1]) [2]

```
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!
```

1. https://www.python.org/dev/peps/pep-0020/
2. https://inventwithpython.com/blog/2018/08/17/the-zen-of-python-explained/

1- Vue d'ensemble

1.3. Python, CPython, ...

Vue d'ensemble
Historique
Philosophie
Python, CPython, ...
Cas d'utilisations de python
Installation
Environnement de développement
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Succès du langage

C'est quoi python au final ?

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
Succès du langage

1- Vue d'ensemble

1.4. Cas d'utilisations de python

Vue d'ensemble
Historique
Philosophie
Python, CPython, ...
Cas d'utilisations de python
Scripting
Exemples personnels
Installation
Environnement de développement
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Succès du langage

Interpréteur embarqué dans des logiciels

Python sert de langage de script dans de nombreux logiciels :

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

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

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
Succès du langage

1- Vue d'ensemble

1.5. Installation

Vue d'ensemble
Historique
Philosophie
Python, CPython, ...
Cas d'utilisations de python
Installation
Environnement de développement
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Succès du langage

Distributions [13]

Il existe plusieurs distributions python.
Les plus connues :

- l'officielle
- anaconda
- (compilation par Intel)
- ...

Pour commencer et sous Windows, je conseille l'installation officielle. Pour les `data scientists` possiblement anaconda.

13. https://wiki.python.org/moin/PythonDistributions

1- Vue d'ensemble

1.6. Environnement de développement

Vue d'ensemble
Historique
Philosophie
Python, CPython, ...
Cas d'utilisations de python
Installation
Environnement de développement
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Succès du langage

Editeurs I

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

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

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

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
Succès du langage

Langage Python

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

Succès du langage

2- Langage Python

2.1. Gestion des variables

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
Succès du langage

Passage par référence

Python fait le maximum pour abstraire la gestion de mémoire.

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

Vue d'ensemble
Langage Python
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
Succès du langage

Mutabilité

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

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

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


Vue d'ensemble
Langage Python
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
Succès du langage

Mutabilité

```
# 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
Succès du langage

Mutabilité

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

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
Succès du langage

Construction des conteneurs

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
Succès du langage

Pour les classes

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

exemple1 = Exemple()
exemple2 = Exemple()

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

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

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


Vue d'ensemble
Langage Python
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
Succès du langage

Cycle de vie

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
```


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
Succès du langage

Cycle de vie

Il y a un *garbage collector* qui s'occupe de supprimer les variables inutilisées.
Il compte les références vers une variable.
Quand il n'y en a plus, il la supprime.

Voilà comment supprimer une référence.

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

del a
print(b)  # [1, 2]
del b  # plus de références
```


Vue d'ensemble
Langage Python
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
Succès du langage

2- Langage Python

2.2. Structures 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
Succès du langage

Structures de données

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
Succès du langage

Piles et files

- comment implémenter une pile (*stack*)
- comment implémenter un file (*queue*)

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
Succès du langage

Piles et files

- comment implémenter une pile (*stack*)
  - list.pop
  - list.append
- comment implémenter un file (*queue*)
  - list.pop(0)
  - list.append

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
Succès du langage

Piles et files

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

Vue d'ensemble
Langage Python
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
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'])
```


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
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'}}}}}}
```


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
Succès du langage

2- Langage Python

2.3. Duck typing

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
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

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

---

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

Vue d'ensemble
Langage Python
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
Succès du langage

Le *duck typing* ?

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
Succès du langage

Exemple

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

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


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

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


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
Succès du langage

Les *able

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

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

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
Succès du langage

2- Langage Python

2.4. Slicing

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
Succès du langage

Slicing

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

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

# remplacement
a[2:5] = [0, 0, 0, 0]
a[::10] = [0, 0, 0, 0, 0, 0, 0, 0] # 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
Succès du langage

Slicing

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]

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
Succès du langage

Objet slice

```
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.

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
Succès du langage

2- Langage Python

2.5. Gestion des fichiers

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
Succès du langage

Lecture de fichiers

```
# 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
Succès du langage

Lecture de fichiers

```
# 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
Succès du langage

Ecriture de fichiers

```
# 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
Succès du langage

Context Manager – gestionnaire de contexte

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

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


# on peut aussi ouvrir plusieurs fichiers
with open("./text.txt") as f_rel, open("/tmp/texte.txt") as f_abs:
    print(f_rel.readlines())
    print(f_abs.readlines())
```


Vue d'ensemble
Langage Python
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
Succès du langage

Context Manager – gestionnaire de contexte

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

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


# on peut aussi ouvrir plusieurs fichiers
with open("./text.txt") as f_rel, open("/tmp/texte.txt") as f_abs:
    print(f_rel.readlines())
    print(f_abs.readlines())
```

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

Vue d'ensemble
Langage Python
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
Succès du langage

2- Langage Python

2.6. Encodage des caractères

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
Succès du langage

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 »
#  ...
```


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
Succès du langage

2- Langage Python

2.7. Contrôle de flux

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
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
```


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
Succès du langage

Boucles – contrôles

On peut contrôler une boucle avec :

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

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
Succès du langage

Itération

Il existe une clause `else` pour les itérations qui est exécutée si les items sont épuisés.

```
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 :(")
```


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
Succès du langage

Boucles – “pythonique et non pythonique”

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

```
# OUI :o)

elements = [3, 2, 40, 10]

for element in elements:
        print(element)
```

```
# NON :'(

elements = [3, 2, 40, 10]

for index in range(len(elements)):
        print(elements[index])
```


Vue d’ensemble
Langage Python
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
Succès du langage

Tuple unpacking

On peut déconstruire des tuples à la volée.

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


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
Succès du langage

* en compréhension

On peut construire / manipuler des itérables à la volée

On appelle ça les listes en compréhension ('list comprehension') ou 'dictionnary 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
Succès du langage

Tests et conditions – syntaxe

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
Succès du langage

Tests et conditions – booléens

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

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

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


Vue d'ensemble
Langage Python
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
Succès du langage

Paresse et générateurs

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

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


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


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


Vue d'ensemble
Langage Python
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
Succès du langage

Gestionnaires de contexte

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()
```


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
Succès du langage

Gestionnaires de contexte

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)
```


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
Succès du langage

Gestionnaires de contexte

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")
```


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
Succès du langage

Gestionnaires de contexte

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)
```


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
Succès du langage

Gestionnaires de contexte

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")
```


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
Succès du langage

2- Langage Python

2.8. Fonctions

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

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

def autre_fonction(param1):
    return param1 * 2

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

b = autre_fonction(2)
print(b)

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

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


Vue d'ensemble
Langage Python
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

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


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


def example_arg_kwargs(param1, *args, **kwargs):
    """Une fonction peut accepter un nombre dynamique
    de paramètres anonymes et nommés.
    Souvent utilisés par les API de bibliothèques.
    Ou quand on ne connait 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(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

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

```
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)
```


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

```
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

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)
```


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 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

Arguments des fonctions

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)
```


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

```
f(1)
f(1, 2)
f(1, b=2)
f(1, 2, 3, b=4, c=5)
f(1, *["c", 3, 4], **{"d": 5, "e": 6})
```


Vue d'ensemble
Langage Python
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 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

Unpacking

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 – résumé

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

Vue d'ensemble
Langage Python
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

Intérêts :

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

Limites :

- complexifie la documentation / utilisation

Vue d'ensemble
Langage Python
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 – 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

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("apres appel inner", variable)


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


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

Espaces de 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
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

Fonctions d'ordre supérieur

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

Les fonctions d'ordre supérieur manipulent d'autres fonctions

Vue d'ensemble
Langage Python
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

Fonctions d'ordre supérieur

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

Les fonctions d'ordre supérieur manipulent d'autres fonctions

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


Vue d'ensemble
Langage Python
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

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,
    "divise": lambda x, y: x / y,
}

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


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

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,
    "divise": lambda x, y: x / y,
}

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

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

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)

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

```
# 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

```
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

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

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

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

```
@decorateur
def fonction():
    pass

fonction = decorateur(fonction)
```


Vue d'ensemble
Langage Python
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

```
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

```
# 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

```
# 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_decorer):
        def wrapper(*args, **kwargs):
            print("avant " * nb_avant)
            res = fonction_a_decorer(*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

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

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

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


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 ?

```
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 ?

```python
from functools import wraps

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

    @wraps(fonction_a_decorer)  # on indique que wrapper entoure une fonction
    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 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 ?

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

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

Vue d'ensemble
Langage Python
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

2- Langage Python

2.9. Générateurs / Itérateurs

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
Succès du langage

Protocole d'itération

```
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
Succès du langage

Les différents types

```
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
Succès du langage

Les différents types

```
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
Succès du langage

Les différents types

```
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
Succès du langage

A quoi ça sert ?

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

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
Succès du langage

Pipeline

Traitement de grandes quantités de données en minimisant l'empreinte mémoire

- on empile des générateurs à la chaîne
- similaire aux pipelines `Unix`

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
Succès du langage

Pipeline

```python
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
Succès du langage

Pipeline

```
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)))))
print(s)
```


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
Succès du langage

Itertools [17]

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
Succès du langage

Consommateurs / Coroutines

Les générateurs peuvent aussi recevoir des valeurs.

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
Succès du langage

Consommateurs / Coroutines

Les générateurs peuvent aussi recevoir des valeurs.

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
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)
```


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
Succès du langage

Intérets

- permet de tirer les données plutôt que de les pousser
- permet d'attendre des données / réagir à des événements
- fonctionnement inverse des consommateurs précédents
- premiers pas dans l'asynchrone

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
Succès du langage

2- Langage Python

2.10. Exceptions

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
Succès du langage

Exceptions

Toujours utiliser une exception précise et bien logguer les erreurs.
Sinon des erreurs peuvent en cacher d'autres.

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
Succès du langage

Exceptions

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


Vue d'ensemble
Langage Python
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
Succès du langage

Exceptions

```
# 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
Succès du langage

Exceptions

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


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

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


Vue d'ensemble
Langage Python
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
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

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
Succès du langage

2- Langage Python

2.11. Introspection

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
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

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
Succès du langage

Introspection ?

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
Succès du langage

Comment faire ?

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

Vue d'ensemble
Langage Python
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
Succès du langage

Comment faire ?

Module `inspect` [18] de la bibliothèque standard

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

18. https://docs.python.org/3/library/inspect.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
Succès du langage

A quoi ça sert ?

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

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
Succès du langage

2- Langage Python

2.12. Bibliographie

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
Succès du langage

Bibliographie I

- 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
Succès du langage

Bibliographie II

- 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 :

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

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 :

Bibliographie V

- http://sametmax.com/gestion-des-erreurs-en-python/
- http://sametmax.com/comment-recruter-un-developpeur-python/
- http://sametmax.com/pourquoi-utiliser-un-mecanisme-dexceptions/

- *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

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
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-trucmuchables-en-python/
- https://stackoverflow.com/questions/1952464/in-python-how-do-i-determine-if-an-object-is-iterable
- https://stackoverflow.com/questions/6589967/how-to-handle-duck-typing-in-python

Programmation Orientée objet (POO)

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

3- Programmation Orientée objet (POO)

3.1. Concepts

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
Succès du langage

Programmation orientée objet (POO)

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

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
Succès du langage

Programmation orientée objet (POO)

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

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
Succès du langage

Constitution d'une classe

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

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
Succès du langage

Constitution d'une classe

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.

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
Succès du langage

Constitution d'une classe

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
Succès du langage

Vocabulaire

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

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

3- Programmation Orientée objet (POO)

3.2. Association

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
Succès du langage

Association entre classes

2 grandes techniques pour associer des classes entre elles :

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

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

Association entre classes

2 grandes techniques pour associer des classes entre elles :

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

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

3- Programmation Orientée objet (POO)

3.3. Modélisation

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
Succès du langage

UML

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)

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
Succès du langage

UML

Différents types de diagrammes

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

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

Diagrammes de classe

Diagramme de classes montrant composition et héritage

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

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

Héritage

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
Succès du langage

3- Programmation Orientée objet (POO)

3.4. POO en 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
Succès du langage

Créer une classe

```
class MonObjet():
    pass


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


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

Créer une classe

```
#  Contructeur, 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
Succès du langage

Créer une classe

```
#  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
Succès du langage

Méthodes magiques

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

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

Mais il y en a d'autres.

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
Succès du langage

Méthodes magiques

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

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

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

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

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


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


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

Héritage

```
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
Succès du langage

Héritage

```python
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
Succès du langage

Accès aux attributs

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

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

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

3- Programmation Orientée objet (POO)

3.5. Gestion des exceptions

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
Succès du langage

Capturer une exception

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

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

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


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

Lever une exception – Personnalisation

```
# 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
Succès du langage

Taxonomie d'exceptions de la DB API

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
Succès du langage

3- Programmation Orientée objet (POO)

3.6. Classe ou pas ?

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
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())
```


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
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"))
```


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
Succès du langage

Quand utiliser une classe ?

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

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

Quand utiliser une classe ?

- Ne pas utiliser
  - quand moins de 2 méthodes...
  - seulement conteneurs, pas de méthodes (utiliser plutôt `dict`, `namedtuple`, ...)
  - gestion des ressources (plutôt `context manager`)
- Utiliser une classe
  - organisation (boîte noire)
  - conserver un état
  - profiter de l'OOP (héritage, ...)
  - surcharge d'opérateurs / méthodes magiques
  - produire une API définie

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

Conteneurs

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

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


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

Dataclasses

Version python $\geqslant$ 3.7

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

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


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

3- Programmation Orientée objet (POO)

3.7. Accès attributs

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
Succès du langage

Le problème

- 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
Succès du langage

Le problème

```python
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
Succès du langage

`__getattr__` / `__getattribute__`

- permettent de gérer les autres cas

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
Succès du langage

__getattr__ / __getattribute__

- permettent de gérer les autres cas
- permettent de créer des attributs à la volée

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
Succès du langage

__getattr__ / __getattribute__

```
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)
```


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
Succès du langage

__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)
```


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
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
```


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
Succès du langage

Descriptors

- utilisés pour les accès aux attributs
- définissent un protocole de lecture / écriture / suppression
- accès supplémentaire rajouté par 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
Succès du langage

Descriptors

```
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(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
Succès du langage

Descriptors

```
# 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
Succès du langage

Properties

- 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
Succès du langage

Properties

```python
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
Succès du langage

Conclusion

Essayer par priorité :

- `attribut classique`
- `property`
- `descriptor`
- `__getattr__`
- `__getattribute__`

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
Succès du langage

3- Programmation Orientée objet (POO)

3.8. Méthodes

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
Succès du langage

Différents types de méthodes

```
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
Succès du langage

Différents types de méthodes

```
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
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)
```


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
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

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
Succès du langage

Résumé

Quel est le résultat ?

```
class MyClass:
    def method(self):
        return ”méthode d'instance”, self

    @classmethod
    def _classmethod(cls):
        return 'méthode de classe', cls

    @staticmethod
    def _staticmethod():
        return 'méthode statique'


print(MyClass._staticmethod())
print(MyClass._classmethod())
print(MyClass.method())

m = MyClass()
print(m._staticmethod())
print(m._classmethod())
print(m.method())
```


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
Succès du langage

3- Programmation Orientée objet (POO)

3.9. Classes abstraites

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
Succès du langage

Classe abstraite ?

- classe *à trous* : il manque des méthodes
- possède des méthodes abstraites
- on doit en hériter pour l'instancier

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
Succès du langage

Exemple

```
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()
cd.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
Succès du langage

Exemple

```python
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
Succès du langage

A quoi ça sert

```python
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()
```


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
Succès du langage

A quoi ça sert

- permet de fixer un contrat
- assez rarement utilisé en pratique

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
Succès du langage

3- Programmation Orientée objet (POO)

3.10. Héritage multiple

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
Succès du langage

Syntaxe

```
class A:
    pass

class B:
    pass

class C(A, B):
    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
Succès du langage

Problème

Héritage en diamant

Quelle méthode va être appelée ?

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
Succès du langage

Method Résolution Order (MRO)

- déterminer l'ordre des méthodes à appeler (`super`)
- linéarisation l'arbre d'héritage
- utilise l'algorithme du C3 [19]

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

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
Succès du langage

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
```


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
Succès du langage

Method Résolution Order (MRO)

Héritage en diamant

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

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


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
Succès du langage

Method Résolution Order (MRO)

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
```


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
Succès du langage

Mixins

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

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
Succès du langage

Mixins

```
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
Succès du langage

3- Programmation Orientée objet (POO)

3.11. Design Patterns

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
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*

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
Succès du langage

Patrons de conception ?

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

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
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.

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
Succès du langage

Patrons de conception ?

3 familles de patrons d'après le GoF

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

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
Succès du langage

Patrons de conception ?

Quelques exemples :

- `factory`
- `adapter`
- `chain of responsability`
- `decorator`
- `facade`
- `iterator`
- `observer`
- `strategy`
- `Model View Controler` (MVC)

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
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)
```


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
Succès du langage

Comportementaux – `Iterator`

Accès aux éléments d'un conteneur séquentiellement sans avoir besoin d'exposer la structure interne du conteneur

Inclus de base dans le langage

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
Succès du langage

Comportementaux – Chain of responsability

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])
```


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
Succès du langage

Comportementaux – Observer

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

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

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

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

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


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
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 ?

```
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)
```


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
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

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
Succès du langage

Structural – Decorateur

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

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

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
Succès du langage

Structural – Decorateur

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

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

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

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

Structural – Adapter

Comment déguiser une classe en une autre ?

```
from datetime import datetime


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


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


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


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

Conclusion

- 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
Succès du langage

3- Programmation Orientée objet (POO)

3.12. Métaclasses

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
Succès du langage

Métaclasses – disclaimer

> 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
Succès du langage

Métaclasses – début

- Tout est objet en 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
Succès du langage

Métaclasses – début

- Tout est objet en python
- Même les classes

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
Succès du langage

Métaclasses – début

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

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
Succès du langage

Métaclasses – début

- 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
Succès du langage

Métaclasses – création de classes

```
> 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
 ...
```


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
Succès du langage

Métaclasses – création de classes

```
> 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
Succès du langage

Métaclasses – Création d'une classe [19]

```python
# création classique
# d'une classe

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

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

Foo = type('Foo', (), {'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
Succès du langage

Métaclasses – Création d'une classe [19]

```
# 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
Succès du langage

Métaclasses – Héritage

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

class FooChild(Foo):
        pass
```

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

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


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
Succès du langage

Métaclasses – Héritage

```
# 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
Succès du langage

Métaclasses – Méthodes

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

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


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
Succès du langage

Métaclasses – Méthodes

```
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
```


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
Succès du langage

Métaclasses – Conclusion

OK. Mais tout ça pour faire quoi ?

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
Succès du langage

Métaclasses – Conclusion

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(metaclass=UpperAttrMetaclass):
    text_exemple = "Un texte"

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


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
Succès du langage

Métaclasses – Conclusion

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
Succès du langage

Métaclasses – Conclusion

OK. Mais tout ça pour faire quoi ?

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
Succès du langage

Métaclasses – Conclusion

OK. Mais tout ça pour faire quoi ?

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

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
Succès du langage

Métaclasses – Conclusion

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
Succès du langage

3- Programmation Orientée objet (POO)

3.13. Bibliographie

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
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/

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
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-getattribute
  - 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/

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

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

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

Bibliographie VI

- http://sametmax.com/le-guide-ultime-et-definitif-sur-la-programmation-orientee-objet-en-python-a-lusage-des-debutants-qui-sont-rassures-par-les-textes-detailles-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-does-one-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

Bibliographie VII

- https://nedbatchelder.com/blog/201503/finding_temp_file_creators.html
- https://mail.python.org/pipermail/python-dev/2008-January/076194.html
- https://fr.slideshare.net/ElizavetaShashkova/monkeypatching-in-python-a-magic-trick-or-a-powerful-tool

Bonnes pratiques

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
Succès du langage

4- Bonnes pratiques

4.1. Pourquoi ?

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
Succès du langage

Qu'est-ce que c'est ?

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)

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
Succès du langage

Pourquoi ?

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

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Pourquoi ?
Installation de paquets
Débug
Linter
Analyse des performances
Tests
Documentation
Création de modules
Programmation concurrente
Succès du langage

Pourquoi ?

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

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
Succès du langage

Pourquoi ?

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

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
Succès du langage

Pourquoi ?

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

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
Succès du langage

4- Bonnes pratiques

4.2. Installation de paquets

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
Succès du langage

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"

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
Succès du langage

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

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
Succès du langage

Écosystème

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

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Pourquoi ?
Installation de paquets
Avant propos
Écosystème
pip
Environnements virtuels
Débug
Linter
Analyse des performances
Tests
Documentation
Création de modules
Programmation concurrente
Succès du langage

Comment ça marche I

En pratique, vous voulez :

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

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

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
Succès du langage

Comment ça marche II

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

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

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
Succès du langage

Comment ça marche III

Le choix est une question de gouts.

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

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
Succès du langage

Installer

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
```


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
Succès du langage

Commandes classiques

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 dépot git
## git@github.com:pypa/sampleproject.git
## on doit rajouter git+ssh:// et changer le :pypa en /pypa
pip install git+ssh://git@github.com/pypa/sampleproject.git

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


Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Pourquoi ?
Installation de paquets
Avant propos
Écosystème
pip
Environnements virtuels
Débug
Linter
Analyse des performances
Tests
Documentation
Création de modules
Programmation concurrente
Succès du langage

Commandes classiques

Installation (cas particuliers)

```
# installation depuis un chemin
pip install .

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

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


Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Pourquoi ?
Installation de paquets
Avant propos
Écosystème
pip
Environnements virtuels
Débug
Linter
Analyse des performances
Tests
Documentation
Création de modules
Programmation concurrente
Succès du langage

Commandes classiques

Cycle de vie des paquets installés

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

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

# supprimer un module
pip uninstall SomePackage
```


Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Pourquoi ?
Installation de paquets
Avant propos
Écosystème
pip
Environnements virtuels
Débug
Linter
Analyse des performances
Tests
Documentation
Création de modules
Programmation concurrente
Succès du langage

Commandes classiques

Fichier `requirements.txt`

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

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


Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Pourquoi ?
Installation de paquets
Avant propos
Écosystème
pip
Environnements virtuels
Débug
Linter
Analyse des performances
Tests
Documentation
Création de modules
Programmation concurrente
Succès du langage

Autres commandes

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

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Pourquoi ?
Installation de paquets
Avant propos
Écosystème
pip
Environnements virtuels
Débug
Linter
Analyse des performances
Tests
Documentation
Création de modules
Programmation concurrente
Succès du langage

Environnement d'installation sain

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

https://xkcd.com/1987/

Environnement d'installation sain

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

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Pourquoi ?
Installation de paquets
Avant propos
Écosystème
pip
Environnements virtuels
Débug
Linter
Analyse des performances
Tests
Documentation
Création de modules
Programmation concurrente
Succès du langage

Installer des modules externes

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...)

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
Succès du langage

virtualenv

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

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Pourquoi ?
Installation de paquets
Avant propos
Écosystème
pip
Environnements virtuels
Débug
Linter
Analyse des performances
Tests
Documentation
Création de modules
Programmation concurrente
Succès du langage

virtualenv

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

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

# on crée le venv
virtualenv venv

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

# on vérifie que ça a marché
which python

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

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


Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Pourquoi ?
Installation de paquets
Avant propos
Écosystème
pip
Environnements virtuels
Débug
Linter
Analyse des performances
Tests
Documentation
Création de modules
Programmation concurrente
Succès du langage

virtualenv

Coexistence de plusieurs versions de Python

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Pourquoi ?
Installation de paquets
Avant propos
Écosystème
pip
Environnements virtuels
Débug
Linter
Analyse des performances
Tests
Documentation
Création de modules
Programmation concurrente
Succès du langage

virtualenv

Organisation des environnements virtuels

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
Succès du langage

virtualenv

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

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Pourquoi ?
Installation de paquets
Avant propos
Écosystème
pip
Environnements virtuels
Débug
Linter
Analyse des performances
Tests
Documentation
Création de modules
Programmation concurrente
Succès du langage

4- Bonnes pratiques

4.3. Débug

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
Succès du langage

Outils de débuggage

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

4- Bonnes pratiques

4.4. *Linter*

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
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
- ...

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
Succès du langage

Qualité du code – pep8 / linters

Il existe des “linters” pour vous assister dans l’écriture.
Ils peuvent lister les erreurs, variables non déclarées, typos, mauvais import...
Ils s’exécutent sans exécuter le code (on parle d’analyse statique)

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

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

Ils peuvent s’intégrer avec les éditeurs de texte.

Vue d’ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Pourquoi ?
Installation de paquets
Débug
Linter
Analyse des performances
Tests
Documentation
Création de modules
Programmation concurrente
Succès du langage

Qualité du code – pep8 / linters

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

- black
- yapf
- autopep8
- ...

Ils peuvent s'intégrer avec les éditeurs de texte.

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Pourquoi ?
Installation de paquets
Débug
Linter
Analyse des performances
Tests
Documentation
Création de modules
Programmation concurrente
Succès du langage

En pratique

- faire attention en cas de projet long [20] / collaboratif (utiliser les mêmes outils, en même temps) en cas d'utilisation d'un formateur automatique
- outils
  - black
  - isort (mise au propre des imports)
  - mypy / pylint
- les intégrer dans des outils (par exemple à chaque sauvegarde d'un fichier)
- on peut les intégrer dans des `pre-commits hook` / un mécanisme d'intégration continue

20. https://black.readthedocs.io/en/stable/guides/introducing_black_to_your_project.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
Succès du langage

4- Bonnes pratiques

4.5. Analyse des performances

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
Succès du langage

Timing et profilage

```
import time, timeit, cProfile


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


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


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

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

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

Timing et profilage

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}
```

4- Bonnes pratiques

4.6. Tests

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
Succès du langage

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
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 unittest [21]

21. https://docs.python.org/3/library/unittest.html

Tests unitaires – tests verts

```python
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, 0.33333333333)

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


Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Pourquoi ?
Installation de paquets
Débug
Linter
Analyse des performances
Tests
Documentation
Création de modules
Programmation concurrente
Succès du langage

Tests unitaires – tests verts

Résultat :

```
python test_unittest.py
....
----------------------------------------------------

Ran 4 tests in 0.001s

OK
```

Tests unitaires – tests rouges

```python
import unittest

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

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

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


Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Pourquoi ?
Installation de paquets
Débug
Linter
Analyse des performances
Tests
Documentation
Création de modules
Programmation concurrente
Succès du langage

Tests unitaires – tests rouges

Résultat :

```
        FE.
======================================================================
ERROR: test_exception (__main__.TestMath)
----------------------------------------------------------------------
Traceback (most recent call last):
    File "../codes/modules/test_unittest2.py", line 13, in test_exception
    computation = 1/0
ZeroDivisionError: division by zero

======================================================================
FAIL: test_error (__main__.TestMath)
----------------------------------------------------------------------
Traceback (most recent call last):
    File "../codes/modules/test_unittest2.py", line 10, in test_error
    self.assertEqual(computation, should_be)
AssertionError: 4 != 3

----------------------------------------------------------------------
Ran 3 tests in 0.001s

FAILED (failures=1, errors=1)
```


Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Pourquoi ?
Installation de paquets
Débug
Linter
Analyse des performances
Tests
Documentation
Création de modules
Programmation concurrente
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()
```


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
Succès du langage

Tests unitaires – fixtures

Voilà le résultat :

```
In setUpClass()

In setUp()
    In test1()
    10
    5
In tearDown()
    set_for_function: 2
.
In setUp()
    In test2()
    1
    5
In tearDown()
    set_for_function: 4
.
In tearDownClass()
3

----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
```

Aller plus loin

Bonne explication du module unittest :
https://pymotw.com/3/unittest/
Pour aller plus loin:

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

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
Succès du langage

Aller plus loin

Bonne explication du module unittest :
https://pymotw.com/3/unittest/
Pour aller plus loin:

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

Cycle TDD (*Test Driven Development*)

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

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
Succès du langage

Aller plus loin

Il existe d'autres modules pour lancer les tests ('testrunners') [22] :

- (doctest [23])
- nose [24]
- pytest (allège la syntaxe des tests) [25]

Les tests sont souvent utilisés avec des 'mocks' [26] pour modifier le comportement des modules externes.

22. https://stackoverflow.com/questions/28408750/unittest-vs-pytest-vs-nose
23. https://docs.python.org/3.6/library/doctest.html
24. https://nose.readthedocs.io/en/latest/
25. https://docs.pytest.org/en/latest/
26. https://docs.python.org/3.6/library/unittest.mock.html

Aller plus loin – autres types de tests

Inspiration :

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

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
Succès du langage

4- Bonnes pratiques

4.7. Documentation

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
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
```

Documentation ?

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

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

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

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

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

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


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
Succès du langage

Comment écrire sa documentation ?

Exemple minimal

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


Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Pourquoi ?
Installation de paquets
Débug
Linter
Analyse des performances
Tests
Documentation
Création de modules
Programmation concurrente
Succès du langage

Comment écrire sa documentation ?

Exemple complet

```python
"""
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

- sphinx (semi automatique) avec :
  - autosummary [27]
  - autodoc [28]

27. http://www.sphinx-doc.org/en/master/usage/extensions/autosummary.html
28. http://www.sphinx-doc.org/en/master/usage/extensions/autodoc.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
Succès du langage

Outils d'extraction de documentation

- sphinx (semi automatique) avec :
  - autosummary [27]
  - autodoc [28]
- sphinx (automatique) avec :
  - autoapi [29]
  - sphinx-autoapi [30]

27. http://www.sphinx-doc.org/en/master/usage/extensions/autosummary.html
28. http://www.sphinx-doc.org/en/master/usage/extensions/autodoc.html
29. http://autoapi.readthedocs.io/
30. http://sphinx-autoapi.readthedocs.io/en/latest/index.html

Outils d'extraction de documentation

- sphinx (semi automatique) avec :
  - autosummary [27]
  - autodoc [28]
- sphinx (automatique) avec :
  - autoapi [29]
  - sphinx-autoapi [30]
- pdoc [31]
- pydoc [32]
- doxygen [33]

27. http://www.sphinx-doc.org/en/master/usage/extensions/autosummary.html
28. http://www.sphinx-doc.org/en/master/usage/extensions/autodoc.html
29. http://autoapi.readthedocs.io/
30. http://sphinx-autoapi.readthedocs.io/en/latest/index.html
31. https://github.com/mitmproxy/pdoc
32. https://docs.python.org/3.6/library/pydoc.html
33. http://www.stack.nl/~dimitri/doxygen/

Syntaxe pour extraction automatique

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

Syntaxe pour extraction automatique

- PEP 8 : https://www.python.org/dev/peps/pep-0008/#documentation-strings
- PEP 257 : https://www.python.org/dev/peps/pep-0257/
- pdoc : markdown [34]
- doxygen : markdown + syntaxe spécifique [35]
- sphinx : RestructuredText [36]
- sphinx avec extension Napoleon [37]
  - Google [38]
  - Numpy [39]

34. https://help.github.com/articles/basic-writing-and-formatting-syntax/
35. https://www.stack.nl/~dimitri/doxygen/manual/docblocks.html
36. https://thomas-cokelaer.info/tutorials/sphinx/rest_syntax.html
37. http://www.sphinx-doc.org/en/master/usage/extensions/napoleon.html
38. https://github.com/google/styleguide/blob/gh-pages/pyguide.md
39. https://numpydoc.readthedocs.io/en/latest/format.html

Formatage des docstrings – Doxygen

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

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

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

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

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

Formatage des docstrings – Doxygen

Python

Main Page | Packages | Classes

Classes | Functions

pyexample Namespace Reference

Documentation for this module. More...

Classes

class **PyClass**
Documentation for a class. More...

Functions

def **func** ()
Documentation for a function. More...

Detailed Description

Documentation for this module.

More details.

Function Documentation

func()

def pyexample.func ( )

Documentation for a function.

More details.

Generated by doxygen 1.8.15

Résultat HTML de l'exemple précédent

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Pourquoi ?
Installation de paquets
Débug
Linter
Analyse des performances
Tests
Documentation
Création de modules
Programmation concurrente
Succès du langage

Formatage des docstrings – reST

```
"""
This is a reST style.

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

Formatage des docstrings – Google vs Numpy

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

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

Returns:
    This is a description of what
    is returned.

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

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

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

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

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

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

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
Succès du langage

Bibliographie

- documentation
  - http://queirozf.com/entries/docstrings-by-example-documenting-python-code-the-right-way
  - https://stackoverflow.com/questions/3898572/what-is-the-standard-python-docstring-format
  - https://docs.python-guide.org/writing/documentation/
  - https://fr.slideshare.net/shimizukawa/sphinx-autodoc-automated-api-documentation-europython-2015-in-bilbao
  - generation / formattage automatique des docstrings : https://github.com/dadadel/pyment
- *code formaters*
  - sametmax.com/once-you-go-black-you-never-go-back/

Création de modules

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
Programmes autonomes
Bibliographie
Programmation concurrente
Succès du langage

5- Création de modules

5.1. Setuptools / Distutils

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
Programmes autonomes
Bibliographie
Programmation concurrente
Succès du langage

Setuptools / Distutils

Permettent d'empaqueter un module python.

Coexistance de Setuptools et Distutils $\rightarrow$ confusion

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
Programmes autonomes
Bibliographie
Programmation concurrente
Succès du langage

Setuptools / Distutils

Permettent d'empaqueter un module python.

Coexistance de Setuptools et Distutils → confusion

On va utiliser `setuptools`

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

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
Programmes autonomes
Bibliographie
Programmation concurrente
Succès du langage

Setuptools / Distutils

Permettent d'empaqueter un module python.

Coexistance 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

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
Programmes autonomes
Bibliographie
Programmation concurrente
Succès du langage

5- Création de modules

5.2. Structure d'un module

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
Programmes autonomes
Bibliographie
Programmation concurrente
Succès du langage

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
```


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
Programmes autonomes
Bibliographie
Programmation concurrente
Succès du langage

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
```

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
Programmes autonomes
Bibliographie
Programmation concurrente
Succès du langage

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


Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Setuptools / Distutils
Structure d'un module
Installation
Mise en ligne PyPI
Programmes autonomes
Bibliographie
Programmation concurrente
Succès du langage

5- Création de modules

5.3. Installation

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
Programmes autonomes
Bibliographie
Programmation concurrente
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
)
```


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
Programmes autonomes
Bibliographie
Programmation concurrente
Succès du langage

Installation

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
Programmes autonomes
Bibliographie
Programmation concurrente
Succès du langage

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)
```

Exemple plus complet

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

Vue d’ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Setuptools / Distutils
Structure d’un module
Installation
Mise en ligne PyPI
Programmes autonomes
Bibliographie
Programmation concurrente
Succès du langage

Exemple plus complet

Arborescence :

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

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

2 directories, 7 files
```


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
Programmes autonomes
Bibliographie
Programmation concurrente
Succès du langage

Exemple plus complet

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_'
    )
```

Exemple plus complet

Contenu de `__init__.py` :

```
from .text import joke
```

Contenu de `command_line.py` :

```
import funniest

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

Contenu de `blagues.txt` :

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

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

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


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
Programmes autonomes
Bibliographie
Programmation concurrente
Succès du langage

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
    )
```

5- Création de modules

5.4. Mise en ligne 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
Programmes autonomes
Bibliographie
Programmation concurrente
Succès du langage

Qui / Où ?

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

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Setuptools / Distutils
Structure d'un module
Installation
Mise en ligne PyPI
Programmes autonomes
Bibliographie
Programmation concurrente
Succès du langage

Qui / Où ?

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
Programmes autonomes
Bibliographie
Programmation concurrente
Succès du langage

Pas à pas

```
# 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
Programmes autonomes
Bibliographie
Programmation concurrente
Succès du langage

5- Création de modules

5.5. Programmes autonomes

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
Programmes autonomes
Bibliographie
Programmation concurrente
Succès du langage

Concept

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

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

Comment distribuer à des utilisateurs finaux ?

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Setuptools / Distutils
Structure d'un module
Installation
Mise en ligne PyPI
Programmes autonomes
Bibliographie
Programmation concurrente
Succès du langage

Solutions

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

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

Plusieurs outils existent :

- **`pyinstaller`**
- `cx_freeze`
- `py2exe`

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Setuptools / Distutils
Structure d'un module
Installation
Mise en ligne PyPI
Programmes autonomes
Bibliographie
Programmation concurrente
Succès du langage

Pyinstaller

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

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

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Setuptools / Distutils
Structure d'un module
Installation
Mise en ligne PyPI
Programmes autonomes
Bibliographie
Programmation concurrente
Succès du langage

Pyinstaller

Utiliser pyinstaller :

- simplement : `pyinstaller myscript.py`
- si plus complexe : `pyinstaller myscript.spec` (le fichier `.spec` permet de configurer l'exécutable comme on le souhaite : https://pyinstaller.org/en/stable/spec-files.html)

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
Programmes autonomes
Bibliographie
Programmation concurrente
Succès du langage

Pyinstaller

Utiliser pyinstaller :

- simplement : `pyinstaller myscript.py`
- si plus complexe : `pyinstaller myscript.spec` (le fichier `.spec` permet de configurer l'exécutable comme on le souhaite : https://pyinstaller.org/en/stable/spec-files.html)

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

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Setuptools / Distutils
Structure d'un module
Installation
Mise en ligne PyPI
Programmes autonomes
Bibliographie
Programmation concurrente
Succès du langage

5- Création de modules

5.6. Bibliographie

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
Programmes autonomes
Bibliographie
Programmation concurrente
Succès du langage

Bibliographie I

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

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Setuptools / Distutils
Structure d'un module
Installation
Mise en ligne PyPI
Programmes autonomes
Bibliographie
Programmation concurrente
Succès du langage

Programmation concurrente

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

6- Programmation concurrente

6.1. Avant propos / définitions

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

Définition

`Programmation concurrente`

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

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

Définition

`Programmation concurrente`

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...

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

Définition

`Programmation concurrente`

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

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

Illustration

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

En python

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

- on peut faire les deux : `threding`, `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)

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

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 peler 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 : peler 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

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

6- Programmation concurrente

6.2. GIL

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Avant propos / définitions
GIL
Parallélisation
Bibliograpgie
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

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
  - $\rightarrow$ multitâche coopératif

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

Problèmes du GIL

Architectures multi coeurs

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

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

Problèmes du GIL

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

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 $\rightarrow$ trop de changement de contexte si charge importante

6- Programmation concurrente

6.3. Parallélisation

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

threads – fonction, classe et verrous

```
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")
```

threads – fonction, classe et verrous

```
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")
```

threads – fonction, classe et verrous

```python
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
```

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()
```

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()
```

Performances Threading – Multiprocess

```python
#! 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")
```

Performances Threading – Multiprocess

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
```

Performances Threading – Multiprocess

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
```

Performances Threading – Multiprocess

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
```

Performances Threading – Multiprocess

Limite à 1 CPU [40]

```
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
```

40. http://xmodulo.com/run-program-process-specific-cpu-cores-linux.html

Performances Threading – Multiprocess

Limite à 1 CPU [40]

```
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
```

40. http://xmodulo.com/run-program-process-specific-cpu-cores-linux.html

Performances Threading – Multiprocess

Limite à 1 CPU [40]

```
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
```

40. http://xmodulo.com/run-program-process-specific-cpu-cores-linux.html

Map Reduce

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

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

Map Reduce

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

6- Programmation concurrente

6.4. Bibliograpgie

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

Bibliographie I

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

Succès du langage

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Intelligence artificielle

7- Succès du langage

7.1. Expressions régulieres

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Intelligence artificielle

Expressions Régulières ?

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

---

https://fr.wikipedia.org/wiki/Expression_r%C3%A9guli%C3%A8re

Syntaxe

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

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

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Intelligence artificielle

Exemples

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

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-aiquo, ex-aeko, ex-æéquo
`^Section .+`	Section 1 / Section a / Section a.a/2	“” / Sectionner / voir Section 1
`[1234567890]+ (,[1234567890]+)?`	2 / 42 / 2,32 / 0.432	3, / ,643 / “”

Cas d'usages

Quand les utiliser :

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

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Intelligence artificielle

Cas d'usages

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
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Intelligence artificielle

En python

Python rajoute des caractères spéciaux pour des cas courants :

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

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

En python

```python
import re

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

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


Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Intelligence artificielle

En python

```python
import re

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

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

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

# on modifie la regex pour gérer le nouveau cas
regex = "(?P<bien>[\w\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
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Intelligence artificielle

7- Succès du langage

7.2. Base de données

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Intelligence artificielle

Accès aux bases de données

- Python permet de se connecter à des bases de données
- Normalisation avec la DB API (database API) [41]
  - 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)

41. https://www.python.org/dev/peps/pep-0249/

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Intelligence artificielle

Présentation DB API

Avec SQLite

```
import sqlite3

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

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

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

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

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

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

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()
```


Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
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

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
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]))
```

Supprimer / mettre à jour des données

```python
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()
```

Erreurs et exceptions

Taxonomie des exceptions d'après la PEP 249

```
StandardError
|__Warning
|__Error
    |__InterfaceError
    |__DatabaseError
        |__DataError
        |__OperationalError
        |__IntegrityError
        |__InternalError
        |__ProgrammingError
        |__NotSupportedError
```

Erreurs et exceptions

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()
```

Erreurs et exceptions

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
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Intelligence artificielle

Bibliographie / Aller plus loin

- https://wiki.python.org/moin/DbApiCheatSheet
- http://sweetohm.net/article/python-dbapi.html
- https://apprendre-python.com/page-database-data-base-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 [42] $\Rightarrow$ abstraire les différences entre moteurs
  - SQLAlchemy
  - Pewee
  - PonyORM
  - ORM Django

42. https://www.fullstackpython.com/object-relational-mappers-orms.html

7- Succès du langage

7.3. XML

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Intelligence artificielle

XML [43]

```python
import xml.etree.cElementTree as ET


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

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

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

43.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
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Intelligence artificielle

XML [43]

```python
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)
```

43.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
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Intelligence artificielle

7- Succès du langage

7.4. JSON

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Intelligence artificielle

JSON

```python
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 chaine
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
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Intelligence artificielle

JSON

Certaines données ne sont pas JSON sérialisables. Il faut créer son propre serialiseur JSON dans ce cas. [44]

```
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))
```

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

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Intelligence artificielle

CSV – excel

```
##########################################
# 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
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Intelligence artificielle

CSV – excel

```python
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
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Intelligence artificielle

CSV – excel

On peut utiliser `xlrd`, `openpyxl` ou pandas (qui se base sur ces dernières) [45]

```
# 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
)
```

45. http://www.python-excel.org/

7- Succès du langage

7.5. Réseau

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Intelligence artificielle

Inventaire

Bas niveau :

- twisted [46]
- ZMQ [47]
- tous les protocoles (PySNMP[48], ...)

46. https://twistedmatrix.com/trac/
47. http://zeromq.org/
48. http://snmplabs.com/pysnmp/index.html

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Intelligence artificielle

Inventaire

Haut niveau (framework web):

- django [46]
- flask [47]
- pyramid [48]
- ...

46. https://www.djangoproject.com/
47. http://flask.pocoo.org/
48. https://trypyramid.com/

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Intelligence artificielle

Inventaire

Asynchrone :

- asyncio / trio [46]
- gevent [47]
- tornado [48]

46. https://github.com/python-trio/trio
47. http://www.gevent.org/
48. https://www.tornadoweb.org/en/stable/

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Intelligence artificielle

Requêtes HTTP

Avec requests

```
# pip install requests
import requests
import pprint

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

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

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

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

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


Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Intelligence artificielle

Twisted

```
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
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Intelligence artificielle

PySNMP

```python
# 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]))
```

Flask

```
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()
```

Serveur web

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
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Intelligence artificielle

7- Succès du langage

7.6. Emailing

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Intelligence artificielle

Envoyer des emails

Il est possible d'envoyer des emails avec Python :

- simples / riches (HTML)
- avec des pièces jointes
- sans identification / avec SSL / SSL + TLS

Plus d'informations ici :
https://realpython.com/python-send-email/

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Intelligence artificielle

Envoyer des emails

```python
# source (modifiée) : https://realpython.com/python-send-email/
from email.message import EmailMessage
import mimetypes, smtplib
from email import encoders
from email.mime.base import MIMEBase
from email.mime.text import MIMEText

from pathlib import Path

msg = EmailMessage()
msg["Subject"] = "Un Mail avec Python"
msg["From"] = "TOTO <moi@toto.fr>"
msg["To"] = ", ".join(["lui@toto.fr", "elle@toto.fr"])
msg.attach(MIMEText("Le contenu du mail", "plain"))

with open(Path("myfile.txt"), "rb") as attachment:
    part = MIMEBase("application", "octet-stream")
    part.set_payload(attachment.read())
encoders.encode_base64(part)
part.add_header("Content-Disposition", f"attachment; filename=myfile.txt")

msg.attach(part)
text = msg.as_string()

with smtplib.SMTP("smtp.toto.fr") as csmtp:
    csmtp.send_message(msg)
```


Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Intelligence artificielle

Envoyer des emails

Il existe des bibliothèques pour faciliter les manipulations :

- enveloppe [49]
- flanker [50]
- yagmail [51]

49. https://github.com/tomekwojcik/envelopes
50. https://github.com/mailgun/flanker
51. https://pypi.org/project/yagmail/

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Intelligence artificielle

Recevoir des emails

Il est possible de recevoir des emails également :

```
# source : https://stackoverflow.com/questions/18156485/
import imaplib

mail = imaplib.IMAP4_SSL("imap.gmail.com")
mail.login("myusername@gmail.com", "mypassword")
mail.list()
# Out: list of "folders" aka labels in gmail.
mail.select("inbox")  # connect to inbox.

result, data = mail.search(None, "ALL")

ids = data[0]  # data is a list.
id_list = ids.split()  # ids is a space separated string
latest_email_id = id_list[-1]  # get the latest

result, data = mail.fetch(
    latest_email_id, "(RFC822)"
)  # fetch the email body (RFC822) for the given ID

raw_email = data[0][1]  # here's the body, which is raw text of the whole email
# including headers and alternate payloads
```

On peut également lancer un serveur SMTP local pour tester ses envois avec : `python -m smtpd -c DebuggingServer -n localhost:1025`

7- Succès du langage

7.7. Calcul distribué et asynchrone

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Intelligence artificielle

Inventaire

Calcul distribué :

- génériques
  - celery [52]
  - redis queue [53]
- scientifiques
  - dask distributed [54]

52. http://www.celeryproject.org/
53. http://python-rq.org/
54. https://dask.org/

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Intelligence artificielle

Celery

Fonctions que l'on veut lancer en arrière plan (fonctions_metier.py)

```
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
```


Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Intelligence artificielle

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)
```


Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Intelligence artificielle

Celery

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

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


Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Intelligence artificielle

7- Succès du langage

7.8. Administration système

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Intelligence artificielle

Administration système d'une et plusieurs machines

Python est très utilisé en administration système, particulièrement pour effectuer des actions sur plusieurs machines distantes.
Il existe différentes bibliothèques / frameworks qui l'utilisent

- fabric [55]
- ansible [56]
- plumbum (en local, sur une machine) [57]

55. https://www.fabfile.org/
56. https://docs.ansible.com/ansible/latest/index.html
57. https://plumbum.readthedocs.io/en/latest/

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Intelligence artificielle

Administration système d'une et plusieurs machines

Lancer du code sur une machine avec `fabric`

```
# source : https://www.fabfile.org/
from fabric import Connection, Exit


def disk_free(c):
    uname = c.run("uname -s", hide=True)
    if "Linux" in uname.stdout:
        command = "df -h / | tail -n1 | awk '{print $5}'"
        return c.run(command, hide=True).stdout.strip()
    err = "No idea how to get disk space on {}!".format(uname)
    raise Exit(err)


print(disk_free(Connection("web1")))
```


Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Intelligence artificielle

Administration système d'une et plusieurs machines

Lancer du code sur plusieurs machines avec `fabric`

```
# source : https://www.fabfile.org/
from fabric import SerialGroup, Exit


def disk_free(c):
    uname = c.run("uname -s", hide=True)
    if "Linux" in uname.stdout:
        command = "df -h / | tail -n1 | awk '{print $5}'"
        return c.run(command, hide=True).stdout.strip()
    err = "No idea how to get disk space on {}!".format(uname)
    raise Exit(err)


for cxn in SerialGroup("web1", "web2", "db1"):
    print("{}: {}".format(cxn, disk_free(cxn)))
```

7- Succès du langage

7.9. Écosystème scientifique

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Intelligence artificielle

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

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Intelligence artificielle

Écosystème

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

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Intelligence artificielle

Écosystème

Calculs :

- numpy [55]
- scipy [56]
- pandas [57]
- ...

Plotting :

- matplotlib [58]
- seaborn [59]
- bokeh [60]
- ...

55. http://www.numpy.org/
56. https://www.scipy.org/
57. https://pandas.pydata.org/
58. https://matplotlib.org/
59. https://seaborn.pydata.org/
60. https://bokeh.pydata.org/en/latest/

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Intelligence artificielle

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))
```


Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Intelligence artificielle

7- Succès du langage

7.10. Jupyter

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Intelligence artificielle

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
- ...

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Intelligence artificielle

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
- ...

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Intelligence artificielle

Numpy

```
import numpy as np

xs = np.arange(-2*np.pi, 2*np.pi, 100)
ys = np.sin(xs) - 3*xs + 2

##########################

A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])
print(A.dot(B))
```


Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Intelligence artificielle

7- Succès du langage

7.11. Pandas

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Présentation
Dataframes
Manipulations de dataframes
Intelligence artificielle

Présentation

Bibliothèque essentielle à l'analyse de données.

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

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Présentation
Dataframes
Manipulations de dataframes
Intelligence artificielle

Dataframes et séries

- `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 tableur)

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Présentation
Dataframes
Manipulations de dataframes
Intelligence artificielle

Manipulations de dataframes

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
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Présentation
Dataframes
Manipulations de dataframes
Intelligence artificielle

Manipulations de dataframes

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)
```

Manipulations de dataframes

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
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Présentation
Dataframes
Manipulations de dataframes
Intelligence artificielle

Manipulations de dataframes

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
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Présentation
Dataframes
Manipulations de dataframes
Intelligence artificielle

Manipulations de dataframes

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
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Présentation
Dataframes
Manipulations de dataframes
Intelligence artificielle

Manipulations de dataframes

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
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Présentation
Dataframes
Manipulations de dataframes
Intelligence artificielle

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)
```


Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Présentation
Dataframes
Manipulations de dataframes
Intelligence artificielle

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]	*Integer list of columns: [0,1,2]*
Slice of rows: [4:7]	*Slice of columns: [4:7]*
Single values: 1	*Single column selections: 1*

loc selections - position based selection

data.loc[<row selection], <column selection>]

Index/Label value: 'john'	*Named column: 'first_name'*
List of labels: ['john', 'sarah']	*List of column names: ['first_name', 'age']*
Logical/Boolean index: data['age'] == 10	*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

df.iloc[2]
df.loc['2000-01-01']

	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

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

df['Evening']
df.Evening
df.loc[:, 'Evening']

	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

7- Succès du langage

7.12. Intelligence artificielle

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Intelligence artificielle

Inventaire

IA :

- sklearn [61]
- tensorflow [62]
- ...

61. https://scikit-learn.orh

62. https://www.tensorflow.org/

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Intelligence artificielle

sklearn

```python
# 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()
```


Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Création de modules
Programmation concurrente
Succès du langage
Expressions régulieres
Base de données
XML
JSON
Réseau
Emailing
Calcul distribué et asynchrone
Administration système
Écosystème scientifique
Jupyter
Pandas
Intelligence artificielle