Formation Python, programmation Objet

Matthieu Falce

Novembre 2022

Au programme I

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

A propos de moi – Qui suis-je ?

- Qui suis-je ?
  - Matthieu Falce
  - habite à Lille
  - ingénieur en bioinformatique (INSA Lyon)
- Qu'est ce que j'ai fait ?
  - ingénieur R&D en Interaction Homme-Machine (IHM), Inria Lille, équipe Mint puis Mjolnir
  - développeur *fullstack* / *backend* à FUN-MOOC (France Université Numérique)

A propos de moi – Actuellement

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

Où me trouver ?

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

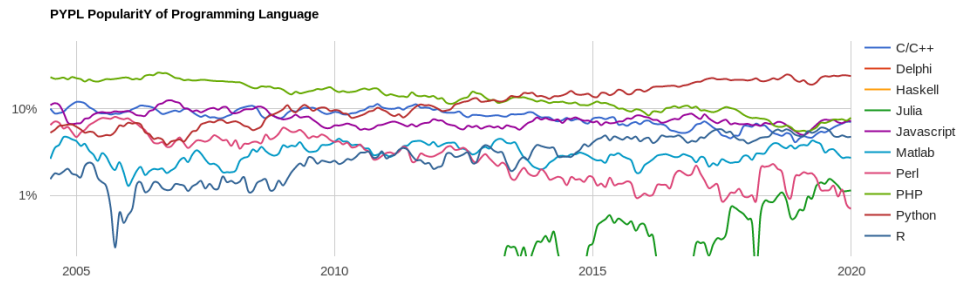
Vue d'ensemble

Un vieux langage ?

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

Un vieux langage ?

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



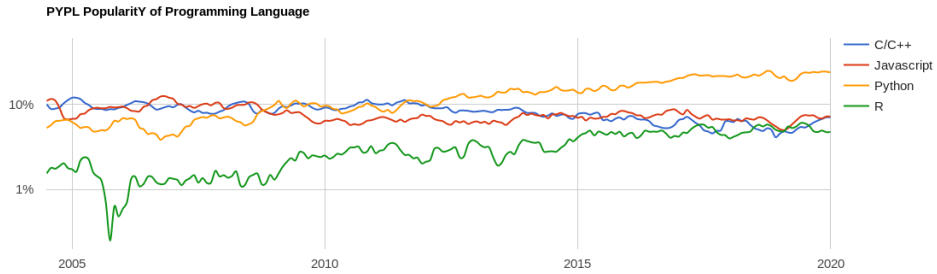
Source: http://pypl.github.io/PYPL.html

Matthieu Falce

Vue d'ensemble
Historique
Philosophie
Python, CPython, ...
Cas d'utilisations de python
Installation
Environnement de développement
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Bibliothèque standard
Interface graphiques
Code natif

Un vieux langage ?

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



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

Origine du nom

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

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

Matthieu Falce

Vue d'ensemble
Historique
Philosophie
Python, CPython, ...
Cas d'utilisations de python
Installation
Environnement de développement
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Bibliothèque standard
Interface graphiques
Code natif

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
- ...

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.

Python 2 End Of Life

Fin du support de Python le 1er janvier 2020

Matthieu Falce

Vue d'ensemble
Historique
Philosophie
Python, CPython, ...
Cas d'utilisations de python
Installation
Environnement de développement
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Bibliothèque standard
Interface graphiques
Code natif

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/

Zen of Python

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

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/

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.

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

Matthieu Falce

Vue d'ensemble
Historique
Philosophie
Python, CPython, ...
Cas d'utilisations de python
Scripting
Exemples personnels
Installation
Environnement de développement
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Bibliothèque standard
Interface graphiques
Code natif

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

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.

Langage Python

Votre premier programme Python

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

Comment lancer un programme python ?

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

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

print("Bonjour le monde")
```

```
## En lançant un fichier.py

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

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

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

Analyse de la syntaxe

```python
# Factorielle en Python

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

```c
// factorielle en C

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


Analyse de la syntaxe

```python
# Factorielle en Python

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

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

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

```c
// factorielle en C

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

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

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

Matthieu Falce

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Analyse de la syntaxe

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

Ne jamais mélanger espaces et tabulation dans un fichier.

Types numériques

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

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

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

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

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

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

Calculs

- divisions flottantes par défaut
- ordre des opérateurs mathématiques

```
import math
import cmath

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

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

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


Chaînes

On peut manipuler facilement les chaînes :

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

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


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

Chaînes – performances

Concaténation des chaînes $\neq$ rapide :

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


Chaînes – contenu spécial

```
# Problème d'échappement

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

print(”C:\\FOO\\Bar\\Baz”)

# raw strings (un seul \)
print(r”C:\Foo\Bar\Baz\ ”)

# Les chaines en Python 3 sont unicodes
print(”éàùµ”)
```

Conteneurs

Les conteneurs permettent de regrouper plusieurs valeurs

```python
# différents types de conteneurs

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

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

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

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


Conteneurs

Les conteneurs permettent de regrouper plusieurs valeurs

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

Matthieu Falce

Vue d'ensemble
Langage Python
Syntaxe
Types standards
Gestion des variables
Duck typing
Slicing
Gestion des fichiers
Encodage des caractères
Contrôle de flux
Fonctions
Exceptions
Bibliographie
Programmation Orientée objet (POO)
Bonnes pratiques
Bibliothèque standard
Interface graphiques
Code natif

Conteneurs

Les conteneurs permettent de regrouper plusieurs valeurs

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

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

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

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


Conteneurs

Les conteneurs permettent de regrouper plusieurs valeurs


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

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

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

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

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

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


Matthieu Falce

Vue d'ensemble

Langage Python

Syntaxe

Types standards

Gestion des variables

Duck typing

Slicing

Gestion des fichiers

Encodage des caractères

Contrôle de flux

Fonctions

Exceptions

Bibliographie

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Chaînes comme conteneurs

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

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

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


Trouver le type d'une variable

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

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

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

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

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

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.

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
```

Matthieu Falce

Vue d'ensemble

Langage Python

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

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Mutabilité

```python
# 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]
```

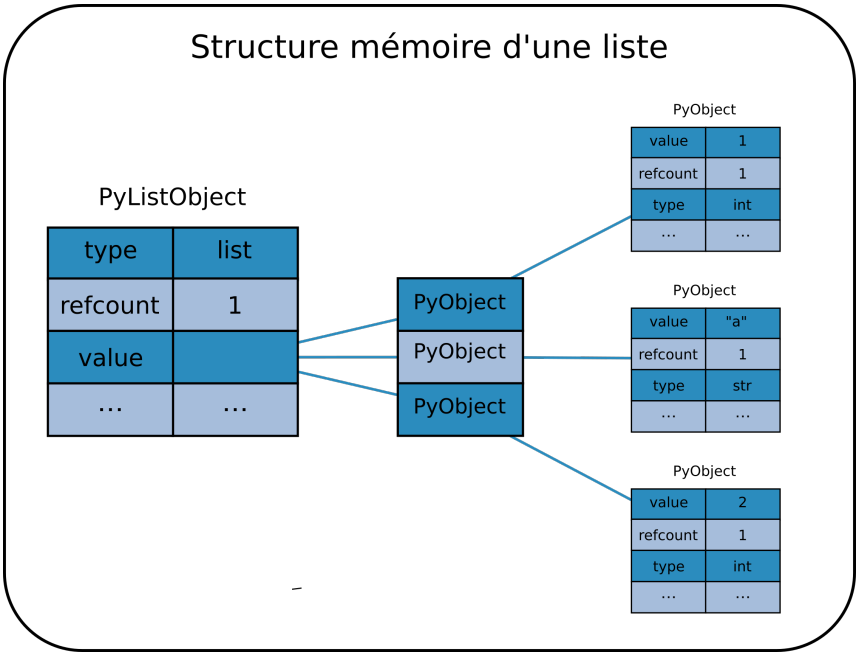

Mutabilité

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

Matthieu Falce

Vue d'ensemble
Langage Python
Syntaxe
Types standards
Gestion des variables
Duck typing
Slicing
Gestion des fichiers
Encodage des caractères
Contrôle de flux
Fonctions
Exceptions
Bibliographie
Programmation Orientée objet (POO)
Bonnes pratiques
Bibliothèque standard
Interface graphiques
Code natif

Construction des conteneurs



Pour les classes

```python
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
```


Matthieu Falce

Vue d'ensemble

Langage Python

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

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Cycle de vie

Copier une variable

```python
import copy

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

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

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

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


Cycle de vie

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.

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

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

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

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`

Matthieu Falce

Vue d'ensemble

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

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

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

Exemple

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

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


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

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

Matthieu Falce

Vue d'ensemble
Langage Python
Syntaxe
Types standards
Gestion des variables
Duck typing
Slicing
Gestion des fichiers
Encodage des caractères
Contrôle de flux
Fonctions
Exceptions
Bibliographie
Programmation Orientée objet (POO)
Bonnes pratiques
Bibliothèque standard
Interface graphiques
Code natif

Les *able

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

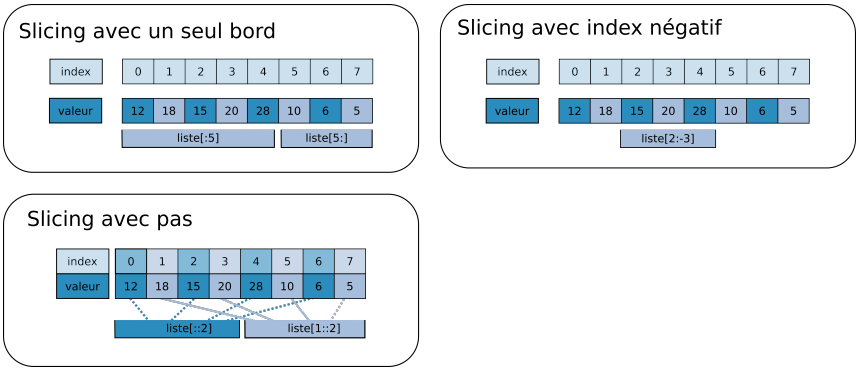
- `iterable` : on peut appliquer une boucle `for`
- `callable` : on peut utiliser `x()` dessus
- `hashable` : peut être passé à la fonction `hash`
- `indexable` : on peut récupérer un élément précis
- `slicable` : on peut appliquer une slice
- ...

Slicing

```python
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
```

Slicing

Explication des slices



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)
```

Lecture de fichiers

```python
# 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
```


Ecriture de fichiers

```python
# 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"
```

Context Manager – gestionnaire de contexte

```python
# 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.

Encodage des caractères

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

```python
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 »
#  ...
```

Boucles

```python
# 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
```


Boucles – contrôles

On peut contrôler une boucle avec :

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

Matthieu Falce

Vue d'ensemble
Langage Python
Syntaxe
Types standards
Gestion des variables
Duck typing
Slicing
Gestion des fichiers
Encodage des caractères
Contrôle de flux
Fonctions
Exceptions
Bibliographie
Programmation Orientée objet (POO)
Bonnes pratiques
Bibliothèque standard
Interface graphiques
Code natif

Boucles – “pythonique et non pythonique”

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

```python
# OUI :o)

elements = [3, 2, 40, 10]

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

```python
# NON :'(

elements = [3, 2, 40, 10]

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


Tuple unpacking

On peut déconstruire des tuples à la volée.

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

* en compréhension

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)}
```


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(":'(")
```

Matthieu Falce

Vue d'ensemble
Langage Python
Syntaxe
Types standards
Gestion des variables
Duck typing
Slicing
Gestion des fichiers
Encodage des caractères
Contrôle de flux
Fonctions
Exceptions
Bibliographie
Programmation Orientée objet (POO)
Bonnes pratiques
Bibliothèque standard
Interface graphiques
Code natif

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
```


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
```

Déclaration d'une fonction

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

def autre_fonction(param1):
    return param1 * 2

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

b = autre_fonction(2)
print(b)

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

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


Déclaration d'une fonction

```python
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)
```

Matthieu Falce

Vue d'ensemble

Langage Python
Syntaxe
Types standards
Gestion des variables
Duck typing
Slicing
Gestion des fichiers
Encodage des caractères
Contrôle de flux
Fonctions
Gestion des arguments
Gotchas
Higher order functions
Closures
Décorateurs
Exceptions
Bibliographie

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

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)
```


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))
```

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)
```


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)
```

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)
```


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})
```


Matthieu Falce

Vue d'ensemble

Langage Python

Syntaxe
Types standards
Gestion des variables
Duck typing
Slicing
Gestion des fichiers
Encodage des caractères
Contrôle de flux
Fonctions
Gestion des arguments
Gotchas
Higher order functions
Closures
Décorateurs
Exceptions
Bibliographie

Programmation Orientée objet (POO)

Bonnes pratiques

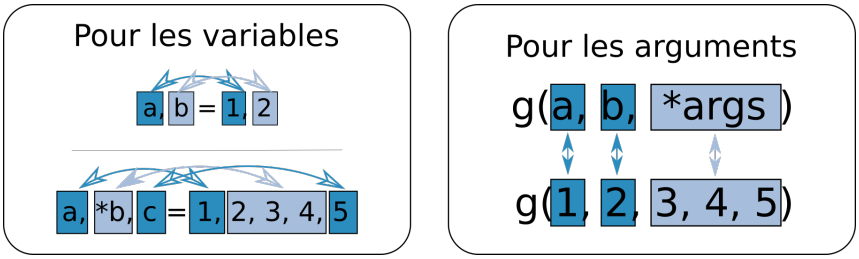
Bibliothèque standard

Interface graphiques

Code natif

Liens avec le unpacking

Unpacking



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

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

Problèmes classiques – éléments mutables [14] [15]

```python
# Attention voilà ce qu'il ne faut pas faire.
# Ne pas mettre d'éléments mutables dans les
# arguments par défaut


def append_wrong(value, li=[]):
    """On s'attend à toujours avoir une liste d'un élément."""
    li.append(value)
    return li


a = append_wrong(1)
b = append_wrong(2)
print(a, b)
# [1, 2], [1, 2]

# on peut également tester en mettant arg=time.time() pour comprendre
# le moment de l'évaluation des paramètres

def append_correct(value, li=None):
    """On met une valeur nulle par défaut et on regarde
    si elle est renseignée ou pas."""
    if li is None:
        li = []
    li.append(value)
    return li


a = append_correct(1)
b = append_correct(2)
```

14. http://docs.python-guide.org/en/latest/writing/gotchas/
15. http://blog.notdot.net/2009/11/Python-Gotchas

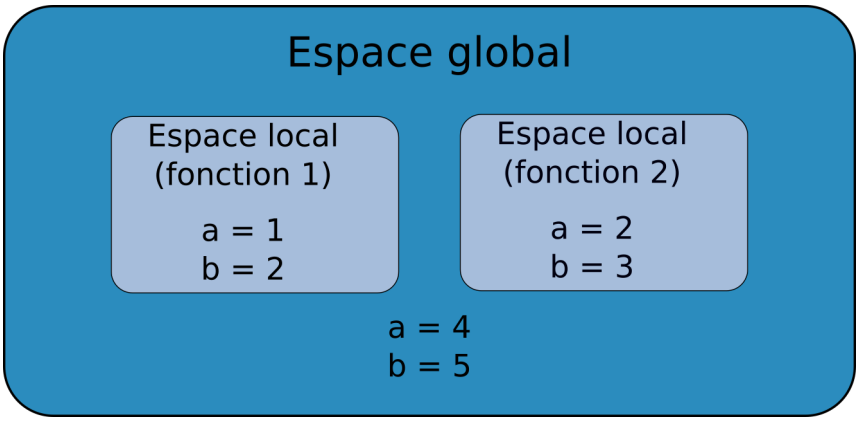
Matthieu Falce
Vue d'ensemble
Langage Python
Syntaxe
Types standards
Gestion des variables
Duck typing
Slicing
Gestion des fichiers
Encodage des caractères
Contrôle de flux
Fonctions
Gestion des arguments
Gotchas
Higher order functions
Closures
Décorateurs
Exceptions
Bibliographie
Programmation Orientée objet (POO)
Bonnes pratiques
Bibliothèque standard
Interface graphiques
Code natif

Problèmes classiques – portée des variables

```python
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()
```


Problèmes classiques – portée des variables

Espaces de noms



Matthieu Falce

Vue d'ensemble

Langage Python

Syntaxe
Types standards
Gestion des variables
Duck typing
Slicing
Gestion des fichiers
Encodage des caractères
Contrôle de flux
Fonctions
Gestion des arguments
Gotchas
Higher order functions
Closures
Décorateurs
Exceptions
Bibliographie

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Fonctions d'ordre supérieur

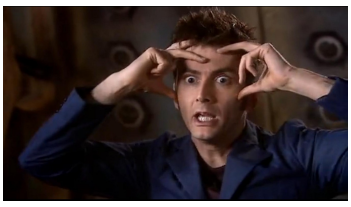
- les fonctions sont des variables comme les autres
- on peut les passer comme argument à d'autres fonctions
- on dit que les fonctions sont des *first class citizen*

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])
```


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)
```



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)

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"
```

Matthieu Falce

Vue d'ensemble

Langage Python
Syntaxe
Types standards
Gestion des variables
Duck typing
Slicing
Gestion des fichiers
Encodage des caractères
Contrôle de flux
Fonctions
Gestion des arguments
Gotchas
Higher order functions
Closures
Décorateurs
Exceptions
Bibliographie

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Closures – Exemples

```python
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
```


Décorateurs – Syntaxe

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

Décorateurs – Syntaxe

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

```
@decorateur
def fonction():
    pass

fonction = decorateur(fonction)
```


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())
```

Décorateurs – Syntaxe

```python
# 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))
```


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
- ...

Exceptions

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

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")
```

Matthieu Falce

Vue d'ensemble
Langage Python
Syntaxe
Types standards
Gestion des variables
Duck typing
Slicing
Gestion des fichiers
Encodage des caractères
Contrôle de flux
Fonctions
Exceptions
Bibliographie
Programmation Orientée objet (POO)
Bonnes pratiques
Bibliothèque standard
Interface graphiques
Code natif

Exceptions

```python
# 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)]))
```


Exceptions

```python
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())
```

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

Bibliographie I

- Décorateurs
  - http://sametmax.com/comprendre-les-decorateur-python-pas-a-pas-partie-2/
  - http://sametmax.com/le-pattern-observer-en-utilisant-des-decorateurs/
  - https://python-3-patterns-idioms-test.readthedocs.io/en/latest/PythonDecorators.html
- Utilisation des astérisques
  - http://treyhunner.com/2018/10/asterisks-in-python-what-they-are-and-how-to-use-them/
- Variables :
  - http://sametmax.com/valeurs-et-references-en-python/
  - http://sametmax.com/id-none-et-bidouilleries-memoire-en-python/

Matthieu Falce

Vue d'ensemble

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

Programmation Orientée objet (POO)

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

Bibliographie II

- https://medium.com/@tyastropheus/tricky-python-i-memory-management-for-mutable-immutable-objects-21507d1e5b95
- Exceptions :
  - http://sametmax.com/gestion-des-erreurs-en-python/
  - http://sametmax.com/comment-recruter-un-developpeur-python/
  - http://sametmax.com/pourquoi-utiliser-un-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

Bibliographie III

- *Duck Typing*
  - https://stackoverflow.com/questions/4205130/what-is-duck-typing
  - https://hackernoon.com/python-duck-typing-or-automatic-interfaces-73988ec9037f
  - https://en.wikipedia.org/wiki/Duck_typing
  - http://sametmax.com/quest-ce-que-le-duck-typing-et-a-quoi-ca-sert/
  - 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

Matthieu Falce

Vue d'ensemble
Langage Python
Syntaxe
Types standards
Gestion des variables
Duck typing
Slicing
Gestion des fichiers
Encodage des caractères
Contrôle de flux
Fonctions
Exceptions
Bibliographie
Programmation Orientée objet (POO)
Bonnes pratiques
Bibliothèque standard
Interface graphiques
Code natif

Programmation Orientée objet (POO)

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

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Concepts
Association
Modélisation
POO en python
Gestion des exceptions
Classe ou pas ?
Méthodes
Bibliographie

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

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

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.

Vocabulaire

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

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

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)

UML

Différents types de diagrammes

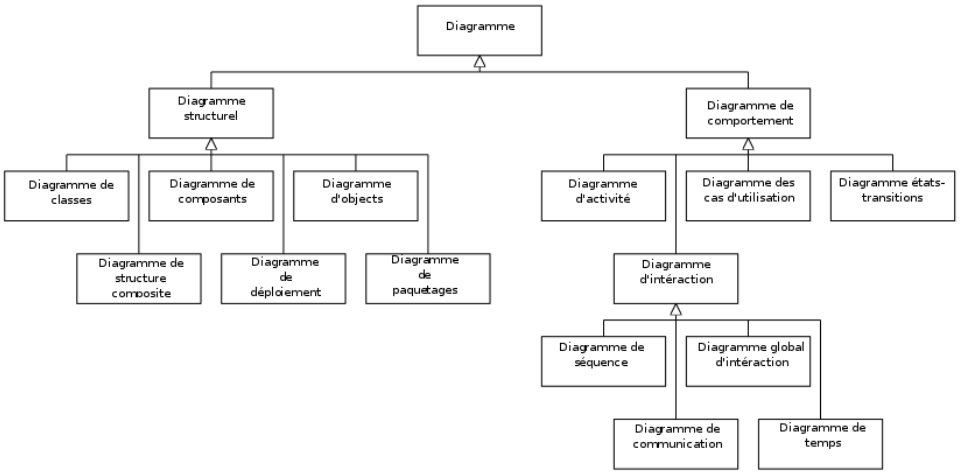
- *diagramme de classes* : représente les classes intervenant dans le système
- diagramme d'objets : représente les instances de classes
- diagramme d'activité : représente la suite des actions à effectuer dans le programme
- ...

Matthieu Falce

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

UML

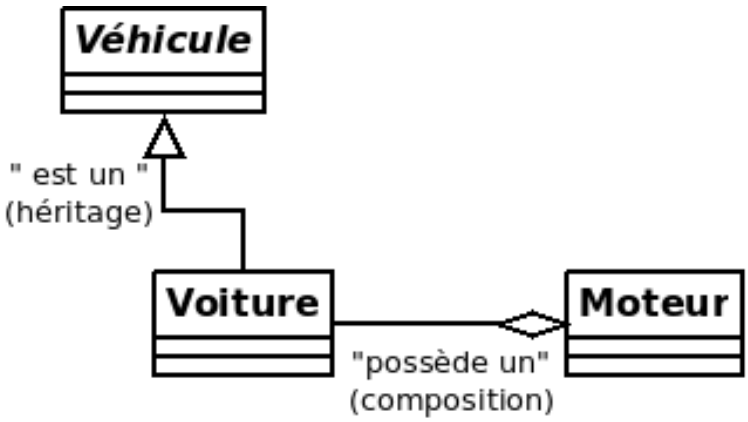
Diagramme montrant la hiérarchie de types de diagrammes UML



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

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Concepts

Association

Modélisation

POO en python

Gestion des exceptions

Classe ou pas ?

Méthodes

Bibliographie

Bonnes pratiques

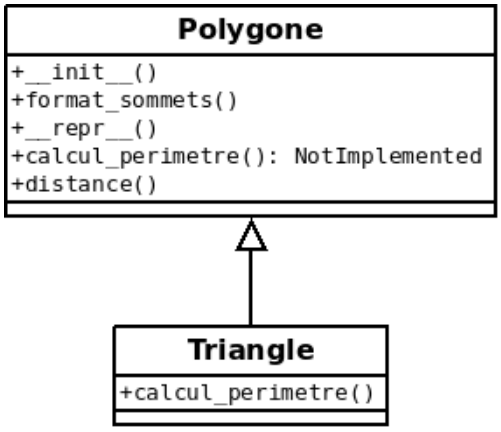
Bibliothèque standard

Interface graphiques

Code natif

Héritage

Diagramme de classes montrant un exemple d'héritage



Créer une classe

```
class MonObjet():
    pass


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

Créer une classe

```python
#  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()
```


Créer une classe

```python
#  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()
```

Matthieu Falce

Vue d'ensemble

Langage Python

Programmation Orientée objet (POO)

Concepts
Association
Modélisation
POO en python
Gestion des exceptions
Classe ou pas ?
Méthodes
Bibliographie

Bonnes pratiques

Bibliothèque standard

Interface graphiques

Code natif

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.

Méthodes magiques

```python
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
```

Matthieu Falce

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

Héritage

```python
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
```


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)
```

Matthieu Falce

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

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`

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)
```

Matthieu Falce

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

Lever une exception – Personnalisation

```python
# 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
```


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
```

Matthieu Falce

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

Quand utiliser une classe ?

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

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


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

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

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


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

Matthieu Falce

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

Conteneurs

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

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


Dataclasses

Version python $\geqslant$ 3.7

```python
@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
```

Différents types de méthodes

```python
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`)

Différents types de méthodes

```python
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

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)
```


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

Résumé

Quel est le résultat ?

```python
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())
```


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/

Bibliographie II

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

Bonnes pratiques

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)

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, ...)

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

Écosystème

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

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.

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)

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`

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
```

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]"
```


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]"
```

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
```


Commandes classiques

Fichier `requirements.txt`

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

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

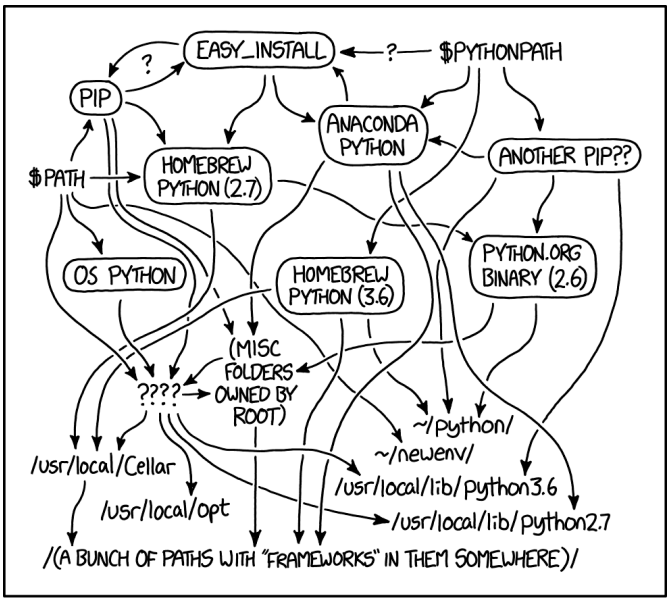
Matthieu Falce

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Pourquoi ?
Installation de paquets
Avant propos
Écosystème
pip
Environnements virtuels
Débug
Linter
Analyse des performances
Tests
Documentation
Bibliothèque standard
Interface graphiques
Code natif

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)

Environnement d'installation sain



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

https://xkcd.com/1987/

Matthieu Falce

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Pourquoi ?
Installation de paquets
Avant propos
Écosystème
pip
Environnements virtuels
Débug
Linter
Analyse des performances
Tests
Documentation
Bibliothèque standard
Interface graphiques
Code natif

Environnement d'installation sain

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

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...)

Matthieu Falce

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Pourquoi ?
Installation de paquets
Avant propos
Écosystème
pip
Environnements virtuels
Débug
Linter
Analyse des performances
Tests
Documentation
Bibliothèque standard
Interface graphiques
Code natif

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)

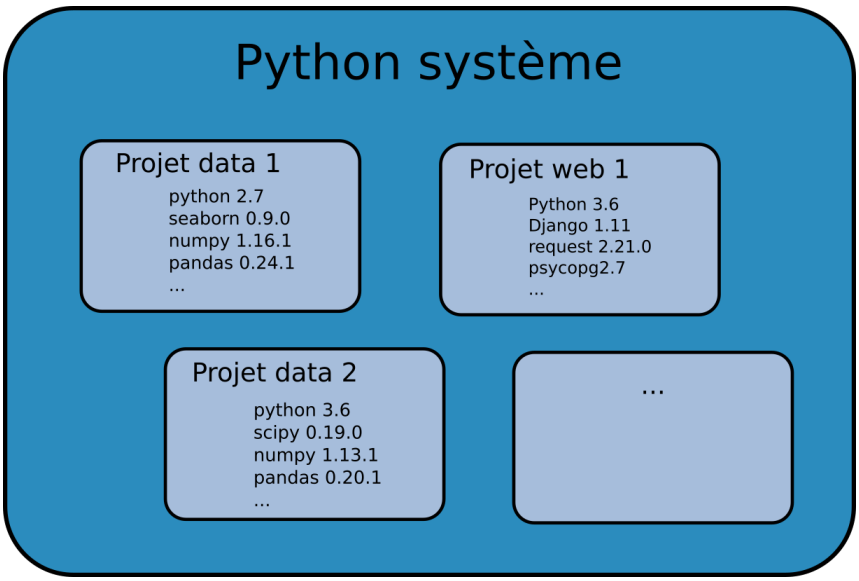
virtualenv

```
#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
```

Matthieu Falce

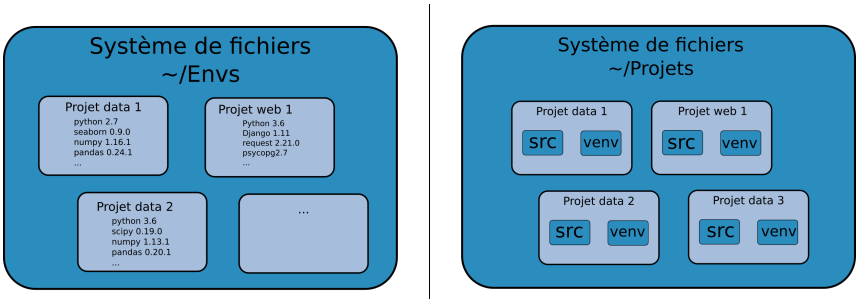
Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Pourquoi ?
Installation de paquets
Avant propos
Écosystème
pip
Environnements virtuels
Débug
Linter
Analyse des performances
Tests
Documentation
Bibliothèque standard
Interface graphiques
Code natif

virtualenv



Coexistence de plusieurs versions de Python

virtualenv



Organisation des environnements virtuels

Matthieu Falce

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Pourquoi ?
Installation de paquets
Avant propos
Écosystème
pip
Environnements virtuels
Débug
Linter
Analyse des performances
Tests
Documentation
Bibliothèque standard
Interface graphiques
Code natif

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

Outils de débuggage

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

```python
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)
```

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
- ...

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.

Matthieu Falce

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Pourquoi ?
Installation de paquets
Débug
Linter
Analyse des performances
Tests
Documentation
Bibliothèque standard
Interface graphiques
Code natif

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.

En pratique

- faire attention en cas de projet long [16] / 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

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

Matthieu Falce

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Pourquoi ?
Installation de paquets
Débug
Linter
Analyse des performances
Tests
Documentation
Bibliothèque standard
Interface graphiques
Code natif

Timing et profilage

```python
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}
```

Matthieu Falce

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Pourquoi ?
Installation de paquets
Débug
Linter
Analyse des performances
Tests
Documentation
Bibliothèque standard
Interface graphiques
Code natif

> 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 [17]

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

Matthieu Falce

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Pourquoi ?
Installation de paquets
Débug
Linter
Analyse des performances
Tests
Documentation
Bibliothèque standard
Interface graphiques
Code natif

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()
```

Résultat :

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

Ran 4 tests in 0.001s

OK
```

Tests unitaires – tests rouges

```
import unittest

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

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

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

Résultat :

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

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

----------------------------------------------------------------------
```


Tests unitaires – fixtures

```
import unittest

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

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

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

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

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

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

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

Matthieu Falce

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Pourquoi ?
Installation de paquets
Débug
Linter
Analyse des performances
Tests
Documentation
Bibliothèque standard
Interface graphiques
Code natif

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
- ...

Cycle TDD (*Test Driven Development*)

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

Matthieu Falce

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Pourquoi ?
Installation de paquets
Débug
Linter
Analyse des performances
Tests
Documentation
Bibliothèque standard
Interface graphiques
Code natif

Aller plus loin

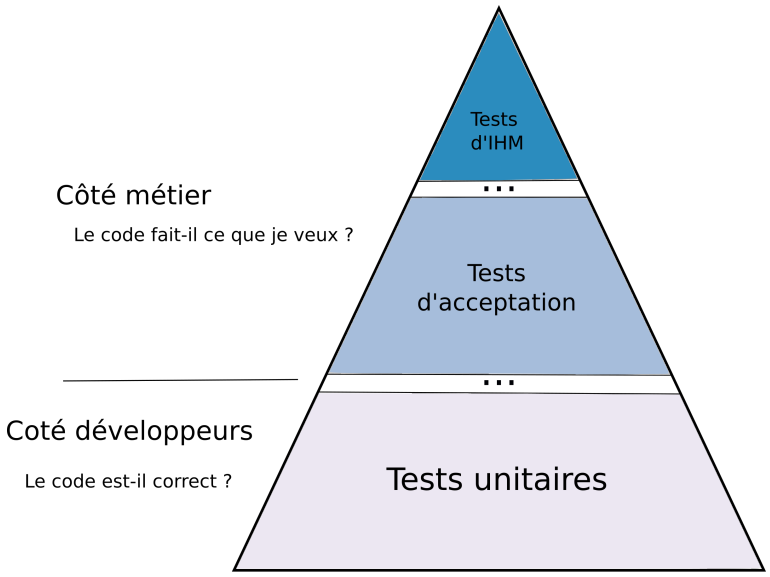
Il existe d'autres modules pour lancer les tests ('testrunners') [18] :

- (doctest [19])
- nose [20]
- pytest (allège la syntaxe des tests) [21]

Les tests sont souvent utilisés avec des 'mocks' [22] pour modifier le comportement des modules externes.

18. https://stackoverflow.com/questions/28408750/unittest-vs-pytest-vs-nose
19. https://docs.python.org/3.6/library/doctest.html
20. https://nose.readthedocs.io/en/latest/
21. https://docs.pytest.org/en/latest/
22. https://docs.python.org/3.6/library/unittest.mock.html

Aller plus loin – autres types de tests



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

Documentation ?

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

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

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

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


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__)
```

Comment écrire sa documentation ?

Exemple minimal

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


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)
```

Matthieu Falce

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Pourquoi ?
Installation de paquets
Débug
Linter
Analyse des performances
Tests
Documentation
Bibliothèque standard
Interface graphiques
Code natif

Outils d'extraction de documentation

- sphinx (semi automatique) avec :
  - autosummary [23]
  - autodoc [24]
- sphinx (automatique) avec :
  - autoapi [25]
  - sphinx-autoapi [26]
- pdoc [27]
- pydoc [28]
- doxygen [29]

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

Syntaxe pour extraction automatique

- PEP 8 : https://www.python.org/dev/peps/pep-0008/#documentation-strings
- PEP 257 : https://www.python.org/dev/peps/pep-0257/
- pdoc : markdown [30]
- doxygen : markdown + syntaxe spécifique [31]
- sphinx : RestructuredText [32]
- sphinx avec extension Napoleon [33]
  - Google [34]
  - Numpy [35]

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

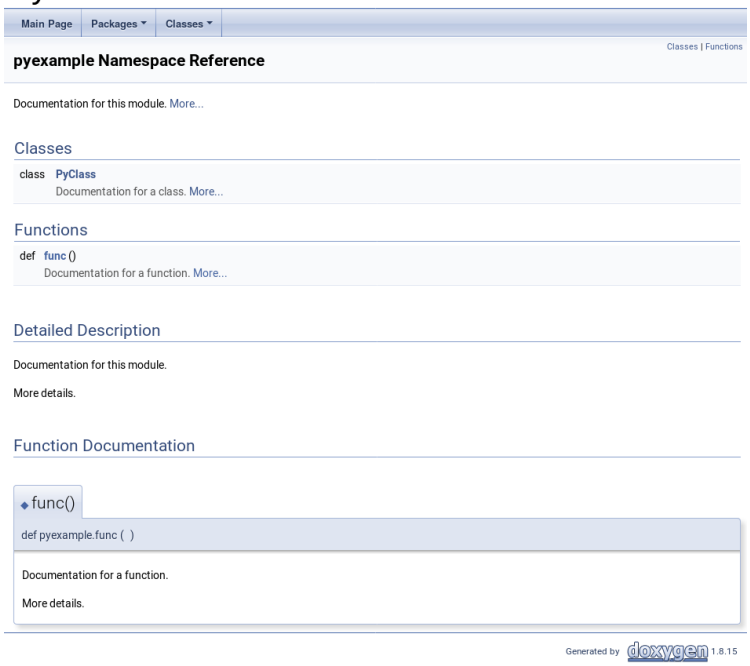
Matthieu Falce
Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Pourquoi ?
Installation de paquets
Débug
Linter
Analyse des performances
Tests
Documentation
Bibliothèque standard
Interface graphiques
Code natif

Formatage des docstrings – Doxygen

```python
## @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



Résultat HTML de l'exemple précédent

Matthieu Falce

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Pourquoi ?
Installation de paquets
Débug
Linter
Analyse des performances
Tests
Documentation
Bibliothèque standard
Interface graphiques
Code natif

Formatage des docstrings – reST

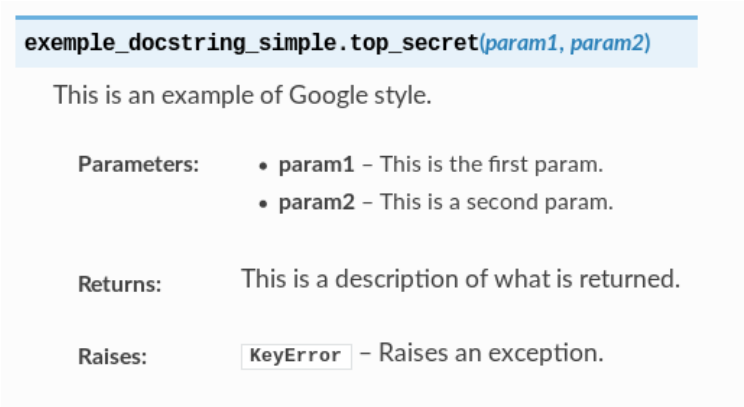
```
"""
This is a reST style.

: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

Bibliographie

Matthieu Falce

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Pourquoi ?
Installation de paquets
Débug
Linter
Analyse des performances
Tests
Documentation
Bibliothèque standard
Interface graphiques
Code natif

- 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/

Bibliothèque standard

"Batteries included"

Python est un langage avec beaucoup de fonctionnalités incluses par défaut

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

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

Matthieu Falce

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

sys

Manipulation des variables en lien avec l'interpréteur.

```python
import sys

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

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

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

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

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

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

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


os

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

```python
import os

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

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

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

# séparateur de fichiers
print(os.sep)

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

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

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

OS

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

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

from os

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

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

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

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

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


Subprocess

Dédié à lancer des commandes "système" depuis python

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

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

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

Subprocess

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

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

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

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

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


Subprocess

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

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

# non bloquant, permet de communiquer
# avec plusieurs process

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

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

Subprocess

Plus d'informations :

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

Outils mathématiques

Ne pas forcément utiliser ceux là pour les calculs scientifiques.

Ils sont plus lents que ceux de numpy / scipy

```python
import random, decimal, fractions, statistics

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

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

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

D = decimal.Decimal
F = fractions.Fraction

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

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

Matthieu Falce

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

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`

Matthieu Falce

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

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

Matthieu Falce

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

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)

En python

Python rajoute des caractères spéciaux pour des cas courants :

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

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

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

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
```


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')]
```

Matthieu Falce

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

Accès aux bases de données

- Python permet de se connecter à des bases de données
- Normalisation avec la DB API (database API) [36]
  - 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)

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

Présentation DB API

Avec SQLite

```python
import sqlite3

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

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

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

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

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

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

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

Présentation DB API

Avec Mysql

```python
# 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()
```


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 $\Rightarrow$ données potentiellement non enregistrées accessibles

Matthieu Falce

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

Insérer / récupérer 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)

# 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()
```

Matthieu Falce

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

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 ?

```python
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()
```

Matthieu Falce

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

Erreurs et exceptions

Quelles données en base à la fin du script ?

```python
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()
```


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 [37] $\Rightarrow$ abstraire les différences entre moteurs
  - SQLAlchemy
  - Pewee
  - PonyORM
  - ORM Django

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

Matthieu Falce

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

XML [38]


```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")
```

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

XML [38]

Matthieu Falce

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

```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)
```

38.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

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
```


JSON

Certaines données ne sont pas JSON sérialisables. Il faut créer son propre serialiseur JSON dans ce cas. [39]

```python
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))
```

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

Matthieu Falce

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

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")]
```


CSV – excel

```
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))
```

Matthieu Falce

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

CSV – excel

On peut utiliser `xlrd`, `openpyxl` ou `pandas` (qui se base sur ces dernières) [40]

```python
# 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
)
```

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

Web – http

Avec la lib standard

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

urlopen = urllib.request.urlopen

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

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

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

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

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

Web – http

Avec requests

```python
# 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())
```


Web – http

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

`python -m http.server`

Matthieu Falce

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

Sockets

```
# Requête HTTP à la main

# exemple socket client
import socket

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

#================================

# Echo server program
# test avec `echo -en "1\n2\n" | nc localhost 50007 -q1`
HOST = ''                 # Symbolic name meaning all available interfaces
PORT = 50007              # Arbitrary non-privileged port
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind((HOST, PORT))
    s.listen(1)
    conn, addr = s.accept()
    with conn:
        print('Connected by', addr)
        while True:
            data = conn.recv(1024)
            if not data:
                break
            conn.sendall(data)
```


Manipulation de fichiers (archivage)

```
import zipfile

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

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

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

Manipulation de fichiers (archivage)

```python
from shutil import make_archive, copy
import os

archive_name = os.path.expanduser(os.path.join("~", "myarchive"))
root_dir = os.path.expanduser(os.path.join("~", ".ssh"))
make_archive(archive_name, "gztar", root_dir)
copy(archive_name, "/tmp/my_archive")
```


Autres modules intéressants I

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

Autres modules intéressants II

En voici quelques un :

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

Autres modules intéressants III

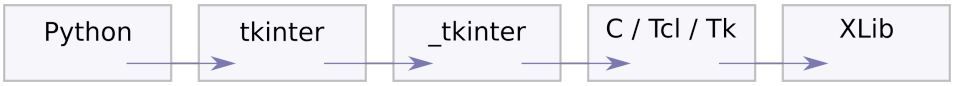
Pour les modules tiers, il faudra faire de la veille pour déterminer les paquets intéressants et pertinents.
Pour évaluer la pérennité du projet, il faut considérer:

- le nombre de développeurs
- l'activité du développement
- le soutien éventuel de grandes entreprises
- la renommée des mainteneurs

Interface graphiques

Contexte

- `Tcl` : langage de programmation [41]
- `Tk` : toolkit d'IHM de `Tcl` [42]
- `Tkinter` : binding python pour `Tcl` / `Tk`



Etapes de traduction du code

41. https://fr.wikipedia.org/wiki/Tool_Command_Language
42. https://fr.wikipedia.org/wiki/Tk_(informatique)

Matthieu Falce

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

Principe de fonctionnement des IHM

Par définition : on intéragit avec une interface graphique

Problématiques :

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

Programmation événementielle

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

Matthieu Falce

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

Programmation événementielle

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

---

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

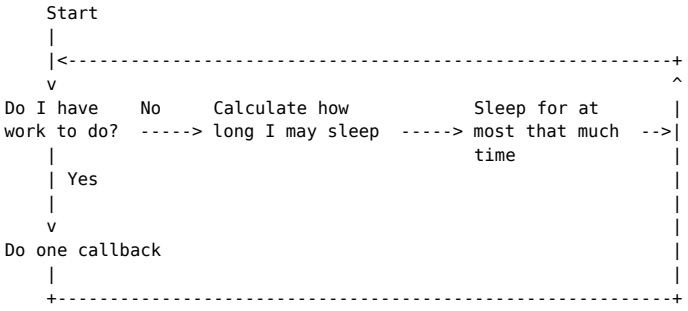
Programmation événementielle

- déclenchement d'événements suite à une interaction
- déclenchement d'événements programmés périodiques
- déclenchement d'événements programmés ponctuels
- du code va réagir à ces événements

Matthieu Falce

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

Programmation événementielle



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

Programmation événementielle

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

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

Matthieu Falce

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

Programmation événementielle

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

Programmation événementielle

Bonus : boucle d'événement minimale (en tkinter)

```
import tkinter

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

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

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

Matthieu Falce

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

Patron de conception : Modèle - Vue - Contrôleur

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

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

Patron de conception : Modèle - Vue - Contrôleur

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

Matthieu Falce

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

Patron de conception : Modèle - Vue - Contrôleur

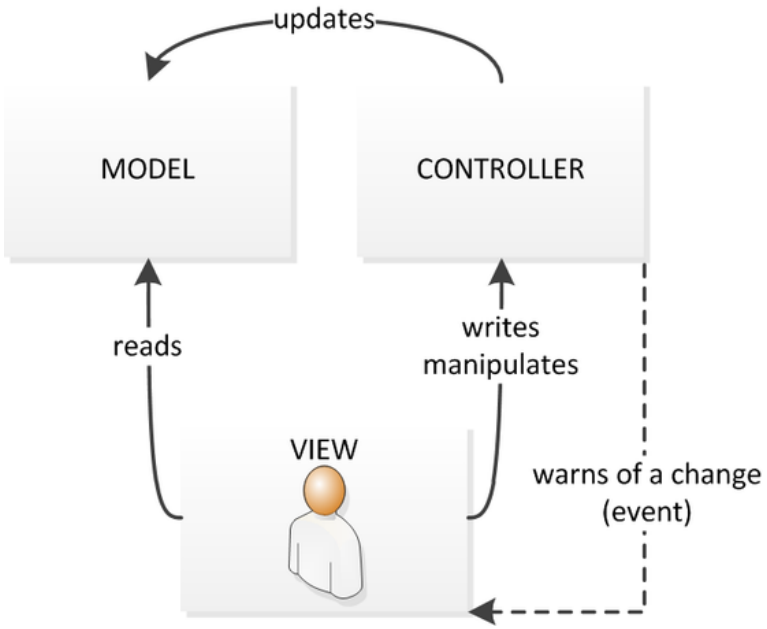


Schéma du modèle MVC

Source: https://fr.wikipedia.org/wiki/Modèle-vue-contrôleur#/media/File:ModeleMVC.png

Patron de conception : Modèle - Vue - Contrôleur

Et `Tcl / Tk` dans tout ça ?

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

---

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

Inspiration du MVC pour découpler et éviter le code spaghetti.

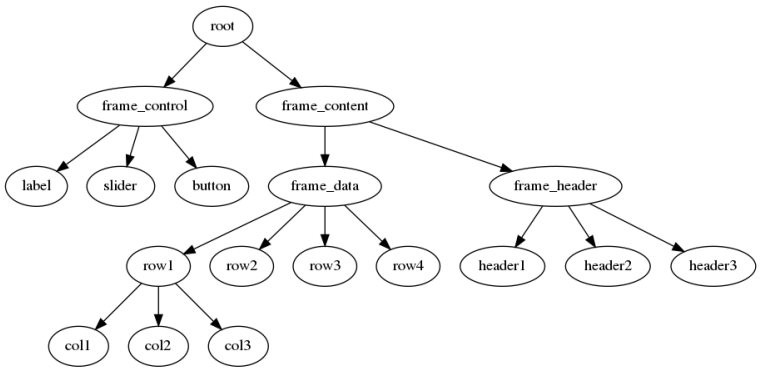
Matthieu Falce

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

Conteneurs

Le conteneur principal est un cadre (Frame).

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

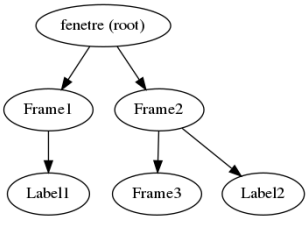


Example de hiérarchie de widgets

Conteneurs [43]

Frames

```
from tkinter import *

fenetre = Tk(); fenetre['bg']='white'

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

fenetre.mainloop()
```



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

Matthieu Falce

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

Conteneurs [43]

Frames

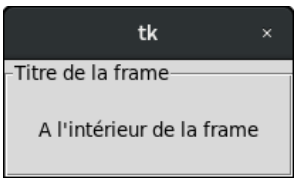
```
from tkinter import *

fenetre = Tk(); fenetre['bg']='white'

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

fenetre.mainloop()
```

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

Conteneurs [43]

LabelFrame

```
from tkinter import *


fenetre = Tk()

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

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

fenetre.mainloop()
```



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

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

Conteneurs [43]

Paned window (peuvent se redimensionner)

```
from tkinter import *

fenetre = Tk()

p = PanedWindow(fenetre, orient=HORIZONTAL)
p.pack(side=TOP, expand=Y, fill=BOTH, pady=2, padx=2)
p.add(Label(p, text='Volet 1', background='blue', anchor=CENTER))
p.add(Label(p, text='Volet 2', background='white', anchor=CENTER) )
p.add(Label(p, text='Volet 3', background='red', anchor=CENTER) )
p.pack()

p2 = PanedWindow(fenetre, orient=VERTICAL)
p2.pack(side=BOTTOM, expand=Y, fill=BOTH, pady=2, padx=2)
p2.add(Label(p2, text='Volet 1', background='blue', anchor=CENTER))
p2.add(Label(p2, text='Volet 2', background='white', anchor=CENTER) )
p2.add(Label(p2, text='Volet 3', background='red', anchor=CENTER) )
p2.pack()

fenetre.mainloop()
```

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

Widgets [45]

Composant d'interface graphiques avec lequel on peut interagir [44]

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

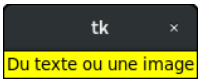
44.https://fr.wikipedia.org/wiki/Composant_d'interface_graphique

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

Widgets [44]

Label

```
from tkinter import *


fenetre = Tk()

label = Label(fenetre, text="Du texte ou une image", bg="yellow")
label.pack()

fenetre.mainloop()
```



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

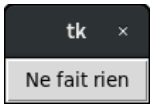
Widgets [44]

Matthieu Falce

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

Button

```
from tkinter import *


fenetre = Tk()

bouton = Button(fenetre, text="Ne fait rien")
bouton.pack()


fenetre.mainloop()
```



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

Widgets [44]

List

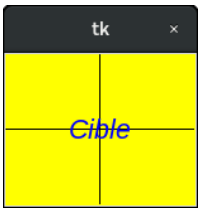
```
from tkinter import *


fenetre = Tk()

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

liste.pack()

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

fenetre.mainloop()
```

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

Widgets [44]

Canvas

```
from tkinter import *


fenetre = Tk()

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

fenetre.mainloop()
```



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

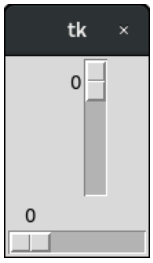
Matthieu Falce

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

Widgets [44]

Scale

```
from tkinter import *


fenetre = Tk()

scale_ver = Scale(fenetre)
scale_ver.pack()

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

# TODO : get value

fenetre.mainloop()
```



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

Widgets [44]

Scrollbar

```
from tkinter import *

fenetre = Tk()

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

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

scrollbar.config(command=listbox.yview)

fenetre.mainloop()
```

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

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

Variables de contrôle

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

Les classes `Variables`

- `BooleanVar`
- `DoubleVar`
- `IntVar`
- `StringVar`

Certains *widgets* en ont besoin pour fonctionner

Variables de contrôle

CheckBox

```
from tkinter import *

fenetre = Tk()
var = IntVar()

bouton = Checkbutton(fenetre, text="J'accepte les CGU", variable=var)
bouton.pack()

# récupération de la valeur
print(var.get())

fenetre.mainloop()
```

Matthieu Falce

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

Variables de contrôle

RadioButton

```
from tkinter import *


fenetre = Tk()

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

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

fenetre.mainloop()
```

Variables de contrôle

Scale – la suite

```
from tkinter import *


fenetre = Tk()

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

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

fenetre.mainloop()
```

Variables de contrôle

Entry

```python
from tkinter import *


fenetre = Tk()


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

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

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

fenetre.mainloop()
```


Variables de contrôle

Entry – validation

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

from tkinter import *

fenetre = Tk()

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

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

fenetre.mainloop()
```

Matthieu Falce

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

Widget quizz

Quels types de widgets pour quelle interaction ?

- entrer un numéro de téléphone [45]
- sélectionner un volume [46]
- créer un mot de passe [47]
- choisir dans une liste d'actions [48]
- choisir un login / mot de passe [49]

45. https://qz.com/679782/programmers-imagine-the-most-ridiculous-ways-to-input-a-phone-number/
46. https://uxdesign.cc/the-worst-volume-control-ui-in-the-world-60713dc86950
47. https://www.reddit.com/r/ProgrammerHumor/comments/904mko/password_input_with_extra_security/
48. https://www.extremetech.com/extreme/262166-hawaiis-missile-scare-driven-terrible-ui-fcc-launches-investigation
49. https://www.reddit.com/r/ProgrammerHumor/comments/8r9xua/so_ive_heard_we_are_now_making_logins_right/

Barre de menu

```python
from tkinter import *

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

fenetre = Tk()
menubar = Menu(fenetre)

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

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

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

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

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

Barre de menu

- `add_command` : ajoute un élément cliquable à une colonne de menu
- `add_checkbutton` : ajoute une case à cocher à une colonne de menu
- `add_radiobutton` : ajoute un radio à une colonne de menu
- `add_cascade` : ajoute une colonne au menu global

Structure du code [52]

- gros codes → encapsulation dans des classes [50]
- soit classe normale / soit widget custom
  - pour une classe normale on passe le widget parent
  - si on hérite de `Tk.frame` / de `Tk` on créée un widget [51]
  - permet une réutilisation facile dans d'autres projets

50. https://softwareengineering.stackexchange.com/questions/213935/why-use-classes-when-programming-a-tkinter-gui-in-python
51. https://stackoverflow.com/questions/7300072/inheriting-from-frame-or-not-in-a-tkinter-application
52. https://stackoverflow.com/questions/17466561/best-way-to-structure-a-tkinter-application/17470842

Approche orientée objet

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

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


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

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

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

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

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


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


Approche orientée objet

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

import tkinter as tk

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

        <create the rest of your GUI here>

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

Layout Managers

2 algorithmes de layout :

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

Options :

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

Packing

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

from tkinter import *

root = Tk()

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

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

mainloop()
```

Matthieu Falce

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Bibliothèque standard
Interface graphiques
Tkinter
Structure du code
Placement widgets
Evénements
QT
Conclusion
Code natif

Packing


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

from tkinter import *

root = Tk()

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

mainloop()
```

Packing


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

from tkinter import *

root = Tk()

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

mainloop()
```

Grid layout

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

from tkinter import *

fen1 = Tk()

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

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

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

# démarrage :
fen1.mainloop()
```


Gestion des événéments

Plusieurs façons de réagir aux événements

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

Gestion des événements

command

La plupart des widgets ont une méthode command

```
from tkinter import *

def on_click():
    print("clic")

fenetre = Tk()

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


fenetre.mainloop()
```

Gestion des événements

command

Comment passer des paramètres à la fonction ?

```
from tkinter import *

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

fenetre = Tk()

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

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


fenetre.mainloop()
```

Gestion des événements

bind

```
from tkinter import *

fenetre = Tk()

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

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

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

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

canvas.pack()
label.pack()

fenetre.mainloop()
```

Gestion des événements

Matthieu Falce

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Bibliothèque standard
Interface graphiques
Tkinter
Structure du code
Placement widgets
Evénements
Gestionnaire de fenêtre
Multifenêtre
QT
Conclusion
Code natif

bind

L'objet event [53]

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

53. http://tkinter.fdex.eu/doc/event.html

Gestion des événements

`bind`

Liste des événements que l'on peut binder :

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

Gestion des événements

`bind`

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

Matthieu Falce

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Bibliothèque standard
Interface graphiques
Tkinter
Structure du code
Placement widgets
Evénements
Gestionnaire de fenêtre
Multifenêtre
QT
Conclusion
Code natif

Gestion des événéments

trace

```
from tkinter import *


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

fenetre = Tk()

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

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


fenetre.mainloop()
```

wm [53]

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

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

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

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

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

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

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

frame.mainloop()
```

53. https://wiki.tcl.tk/9457

Applications Multifenêtre

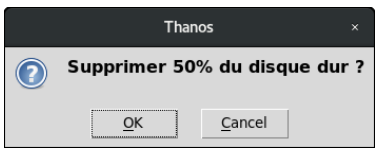
- choix d'un fichier / dossier
- réponse à une question
- formulaire supplémentaire pour finir une action
- "simplifier" la présentation

Message / dialogues / popup

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

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

```python
from tkinter import messagebox

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

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



Matthieu Falce

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Bibliothèque standard
Interface graphiques
Tkinter
Structure du code
Placement widgets
Evénements
Gestionnaire de fenêtre
Multifenêtre
QT
Conclusion
Code natif

Message / dialogues / popup

Interaction ponctuelle avec l'utilisateur.

Choisir d'un fichier / dossier [54]

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

54.http://tkinter.fdex.eu/doc/popdial.html

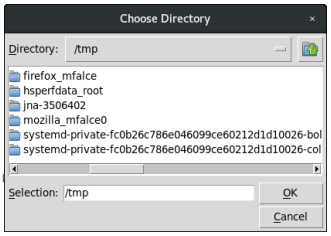
Message / dialogues / popup

Interaction ponctuelle avec l'utilisateur.

Choisir d'un fichier / dossier [54]

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

```
from tkinter import filedialog

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



54.http://tkinter.fdex.eu/doc/popdial.html

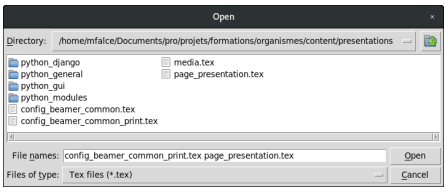
Message / dialogues / popup

Interaction ponctuelle avec l'utilisateur.

Choisir d'un fichier / dossier [54]

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

```
from tkinter import filedialog

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



54.http://tkinter.fdex.eu/doc/popdial.html

Fenetres secondaires

On utilise `TopLevel` [55] :

```
from tkinter import *

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

fenetre = Tk()

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

fenetre.mainloop()
```

55.http://effbot.org/tkinterbook/toplevel.htm

Style

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

Bibliographie / Aller plus loin I

Méthodes communes aux widgets :
`http://tkinter.fdex.eu/doc/uwm.html`
Event loop :

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

MVC :

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

Matthieu Falce

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Bibliothèque standard
Interface graphiques
Tkinter
Structure du code
Placement widgets
Evénements
Gestionnaire de fenêtre
Multifenêtre
QT
Conclusion
Code natif

Bibliographie / Aller plus loin II

- https://openclassrooms.com/fr/courses/1894236-programmez-avec-le-langage-c/1902176-larchitecture-mvc-avec-les-widgets-complexes
- https://www.codeguru.com/cpp/cpp/implementing-an-mvc-model-with-the-qt-c-framework.html

- MVC en Tkinter https://codereview.stackexchange.com/questions/163342/applying-model-view-controller-to-tkinter-matplotlib-application

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

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

Contexte

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

Contexte

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

Qt5 / Qt6

Une nouvelle version majeure de Qt est sortie en 2021 : Qt6. Il y a des différences entre Qt5 et Qt6 et donc également dans les versions Python. Cette page liste les modifications à effectuer : https://www.pythonguis.com/faq/pyqt5-vs-pyqt6/.

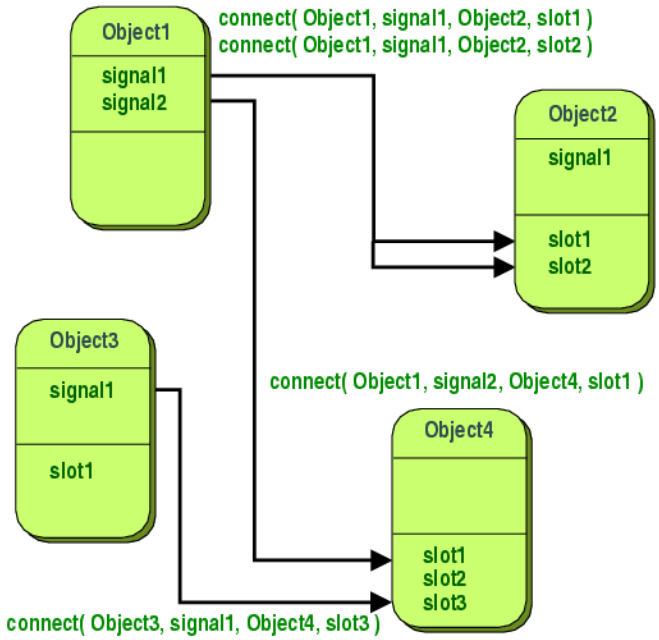
Par quoi commencer ?

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

Signaux et slots

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

Signaux et slots



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

Exemples de code

Des ressources peuvent se trouver ici :

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

Exemples de code

```python
# Source : https://www.pythonguis.com/tutorials/pyqt-signals-slots-events/

import sys
from PyQt5.QtWidgets import QApplication, QMainWindow, QPushButton


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

        self.setWindowTitle("My App")


app = QApplication(sys.argv)

window = MainWindow()
window.show()

app.exec()
```

Matthieu Falce

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Bibliothèque standard
Interface graphiques
Tkinter
Structure du code
Placement widgets
Evénements
QT
Contexte
Exemples
Conclusion
Code natif

Exemples de code

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


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

        self.button_is_checked = True

        self.setWindowTitle("My App")

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

        self.setCentralWidget(button)

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

        print(self.button_is_checked)


app = QApplication(sys.argv)

window = MainWindow()
window.show()

app.exec()
```


Elements à considérer

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

Comparaison

	Qt	Tkinter
Avantages	* Multi-plateforme / widgets spécifiques * Flexible / permet d'organiser le code * Qt creator (création d'interfaces en glissé déposé) * Fourni un écosystème d'outils (connexion aux bases de données, threads, fichiers...) * Nombreux widgets * Beaucoup de ressources en ligne	* Disponible de base en python sans rien installer * Facile à prendre en main
Inconvénients	* Complexe (POO, il faut chercher la documentation pour le C++) * Mécanisme de licence compliqué quand on ne fait pas de l'open source * Doit être installé	* Pas de widgets avancés (un tableau par exemple) * Intégration au style de l'OS compliqué * Gestion de la complexité compliquée

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

Listing

Il existe d'autre framework d'interfaces graphiques

- GTK
- wxPython
- Kivy

Il existe également des bibliothèques permettant d'abstraire le choix du framework qui peuvent être intéressantes : `https://pysimplegui.readthedocs.io/en/latest/` (tk, qt, wxpython et web)

Code natif

Ctypes

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

test1.c

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

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

---

test2.c

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

Matthieu Falce

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Bibliothèque standard
Interface graphiques
Code natif
Ctypes
Cython
Embarquer du code Python dans du C
Bibliographie

Ctypes

Makefile :

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

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


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


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

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


Ctypes

main1.c

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

#define STR_LEN 40

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

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

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

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

  return EXIT_SUCCESS;
}
```

Matthieu Falce

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Bibliothèque standard
Interface graphiques
Code natif
Ctypes
Cython
Embarquer du code Python dans du C
Bibliographie

Ctypes

main2.c

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

#define STR_LEN 40

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

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

  return EXIT_SUCCESS;
}
```


Ctypes

main.py

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

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

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

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

    res = create_string_buffer(40)

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

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

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

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

Matthieu Falce

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Bibliothèque standard
Interface graphiques
Code natif
Ctypes
Cython
Embarquer du code Python dans du C
Bibliographie

Ctypes

Pratique pour intégrer rapidement du code depuis une bibliothèque native.

Assez compliqué à maintenir.

Pas de construction graduelle vers le C.

Cython

Cython est un compilateur statique / langage permettant :

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

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

Cython

tools.c :

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

#define STRING_SIZE 50

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

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

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


Cython

wrapper.pyx :

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

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


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

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

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

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

Matthieu Falce

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Bibliothèque standard
Interface graphiques
Code natif
Ctypes
Cython
Embarquer du code Python dans du C
Bibliographie

Cython

setup.py (python setup.py build_ext –inplace) :

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

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

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


Cython

main.py :

```
import numpy as np

from tools_wrapper import hello, sum_np_array

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

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

Cython

Résultat du `cython -a tools_wrapper.pyx` :

Generated by Cython 0.27.3

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

Raw output: tools_wrapper.c

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


Explications

Il y a deux façons de faire cohabiter Python et C

- augmenter Python avec des routines C (ce que l'on a vu)
- embarquer l'interpréteur Python dans le C (ce que l'on va voir)

Nécessite de connaître suffisamment le C pour comprendre l'API C de Python

Embarquer du code

Il existe 3 niveaux d'embarquement :

- vu que l'on initialise un interpréteur, on peut appeler des chaînes de code directement
- on peut appeler des fonctions python et récupérer leur valeurs (échange des paramètres et des valeurs retournées)
- on peut mettre à disposition des variables C dans un module que l'on importe dans le code interprété

Toutes les infos sont ici : `https://docs.python.org/3/extending/embedding.html`

Exemple d'embarquement de code


```c
#define PY_SSIZE_T_CLEAN
#include <Python.h>

int
main(int argc, char *argv[])
{
    wchar_t *program = Py_DecodeLocale(argv[0], NULL);
    if (program == NULL) {
        fprintf(stderr, "Fatal error: cannot decode argv[0]\n");
        exit(1);
    }
    Py_SetProgramName(program);  /* optional but recommended */
    Py_Initialize();
    PyRun_SimpleString("from time import time,ctime\n"
                       "print('Today is', ctime(time()))\n");
    if (Py_FinalizeEx() < 0) {
        exit(120);
    }
    PyMem_RawFree(program);
    return 0;
}
```

Embarquer une chaîne de Python

Exemple d'embarquement de code

```
...
Py_Initialize();
pName = PyUnicode_DecodeFSDefault(argv[1]);
pModule = PyImport_Import(pName);
Py_DECREF(pName);

if (pModule != NULL) {
    pFunc = PyObject_GetAttrString(pModule, argv[2]);
    /* pFunc is a new reference */

    if (pFunc && PyCallable_Check(pFunc)) {
        pArgs = PyTuple_New(argc - 3);
        for (i = 0; i < argc - 3; ++i) {
            pValue = PyLong_FromLong(atoi(argv[i + 3]));
            // ... removed check if not pValue
            PyTuple_SetItem(pArgs, i, pValue);
        }
        pValue = PyObject_CallObject(pFunc, pArgs);
        Py_DECREF(pArgs);
        if (pValue != NULL) {
            printf("Result of call: %ld\n", PyLong_AsLong(pValue));
            ...
```

Appeler un module Python (attention au PYTHONPATH)

Gotchas

- cette partie fonctionne sous Linux (Ubuntu au moins), pour Windows je n'ai pas testé
- il faut déterminer les paramètres de compilation pour sa machine :
  - `CFLAGS` : lancer `python-config –cflags`
  - `LDFLAGS` : lancer `python-config –ldflags`
- l'interpréteur embarqué ne semble pas régler `PYTHONPATH` avec le dossier courant, il faut le faire à la main, sinon `ImportError` (`PyRun_SimpleString("import sys, os; sys.path.append(os.getcwd())");`)
- l'intégration de code Python et C est vue comme de la *magie noire*. Ce n'est pas vrai, c'est faisable, cependant, cela nécessite de bonnes connaissances dans les deux langages.

Matthieu Falce

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Bibliothèque standard
Interface graphiques
Code natif
Ctypes
Cython
Embarquer du code Python dans du C
Bibliographie

Bibliographie I

- étendre python avec du C
  - https://wiki.python.org/moin/IntegratingPythonWithOtherLanguages
  - https://docs.scipy.org/doc/numpy/user/c-info.python-as-glue.html
  - https://stackoverflow.com/questions/145270/calling-c-c-from-python
  - SIP (binding Qt et GTK)
  - www.swig.org
  - http://sametmax.com/appeler-du-code-c-depuis-python-avec-ctypes/
  - http://www.boost.org/doc/libs/1_49_0/libs/python/doc/
  - https://github.com/pybind/pybind11
  - https://cffi.readthedocs.io/en/latest/overview.html#simple-example-abi-level-in-line
  - http://sametmax.com/introduction-aux-extensions-python-avec-cffi

Bibliographie II

- sur Windows : https://docs.python.org/3/extending/windows.html
- https://docs.python.org/3/extending/building.html
- https://realpython.com/python-bindings-overview/
- https://realpython.com/build-python-c-extension-module/

- embarquer python dans du C
  - https://docs.python.org/3/c-api/
  - https://docs.python.org/3/extending/embedding.html

Matthieu Falce

Vue d'ensemble
Langage Python
Programmation Orientée objet (POO)
Bonnes pratiques
Bibliothèque standard
Interface graphiques
Code natif
Ctypes
Cython
Embarquer du code Python dans du C
Bibliographie