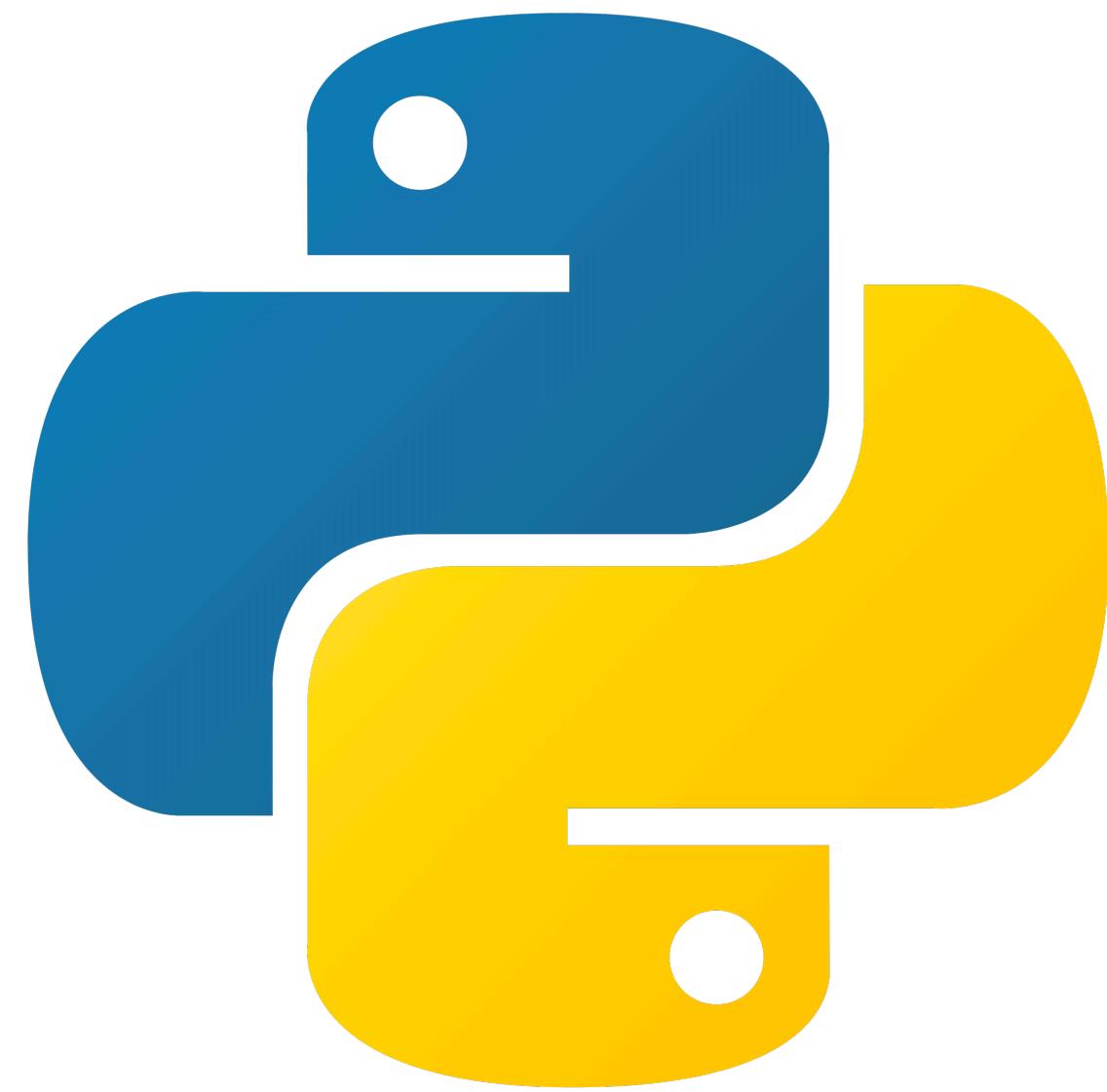


Python



Perfectionnement

Whoami

Frédéric GAURAT - Développeur / Formateur



Déroulement de la journée

9h00 : Début

14h00 : Reprise

10h45 : Pause

15h45 : Pause

11h00 : Reprise

16h00 : Reprise

12h30 : Repas

17h30 : Fin

Objectifs Pédagogiques, Niveau requis Et Public concerné

- **Objectifs pédagogiques / Compétences visées**

- Implémenter de manière rigoureuse des Design Patterns reconnus
- Utiliser les techniques avancées du langage Python : Context Manager, métaclasses, closures, fonctions avancées
- Optimiser les performances de vos programmes à l'aide du monitoring et du parallélisme
- Packager et déployer ses artefacts Python
- Exploiter des librairies contribuant au succès du langage : calcul scientifique, Intelligence Artificielle, XML, réseau

- **Niveau requis**

- Bonnes connaissances en développement Python, ou connaissances équivalentes à celles apportées par les stages THO ou PYT.
Expérience requise.

- **Public concerné**

- Ingénieurs et développeurs.

Programme de la formation

Rappels importants sur le langage

Variables, slices, structures de contrôle

Fonctions avancées

Passage de paramètres, générateurs, décorateurs

POO avancée

property, itérateur, héritage, Context Managers, ABC et métaclasses

Programme de la formation

Déploiement et qualité

Setupools, Pypi, venv, profiling

Le parallélisme : optimiser les performances de vos programmes

Multiprocessing, Celery (Docker,...)

Les librairies contribuant au succès du langage

Numpy, Matplotlib et Pandas, Scikit-Learn, XML, TCP et SNMP

Python 3.10

04/10/2021

Les nouveautés de Python 3.10

Structural Pattern Matching

Le **Structural Pattern Matching** introduit l'instruction `match/case`

L'instruction `match/case` suit le même schéma de base que **switch/case**.

Elle prend un objet, teste l'objet par rapport à un ou plusieurs motifs de correspondance, et effectue une action si elle trouve une correspondance.

Les nouveautés de Python 3.10

Structural Pattern Matching

```
def http_error(status):
    match status:
        case 400:
            return "Bad request"
        case 404:
            return "Not found"
        case 418:
            return "I'm a teapot"
        case _:
            return "Something's wrong with the internet"
```

Les nouveautés de Python 3.10

Parenthesized Context Managers

L'utilisation de parenthèses pour la continuation sur plusieurs lignes dans les gestionnaires de contexte est désormais prise en charge.

Cela permet de formater une longue collection de gestionnaires de contexte sur plusieurs lignes de la même manière qu'il était possible auparavant avec les instructions d'importation.

Les nouveautés de Python 3.10

Parenthesized Context Managers

```
● ● ●

with (open("file1.txt") as example):
    ...

with (
    open("file1.txt"),
    open("file2.txt")
):
    ...

with (open("file1.txt") as example,
      open("file2.txt")):
    ...

with (open("file1.txt"),
      open("file2.txt") as example):
    ...

with (
    open("file1.txt") as example1,
    open("file2.txt") as example2
):
    ...
```

Les nouveautés de Python 3.10

Better error messages

Avant



```
expected = {9: 1, 18: 2, 19: 2, 27: 3, 28: 3, 29: 3, 36: 4, 37: 4,
            38: 4, 39: 4, 45: 5, 46: 5, 47: 5, 48: 5, 49: 5, 54: 6,
some_other_code = foo()
```



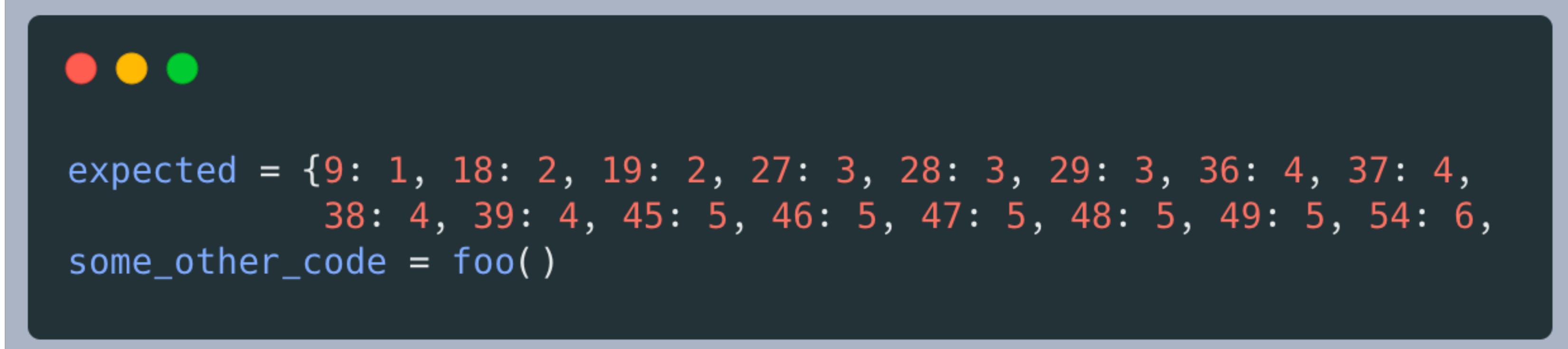
```
File "example.py", line 3
    some_other_code = foo()
                  ^
```

```
SyntaxError: invalid syntax
```

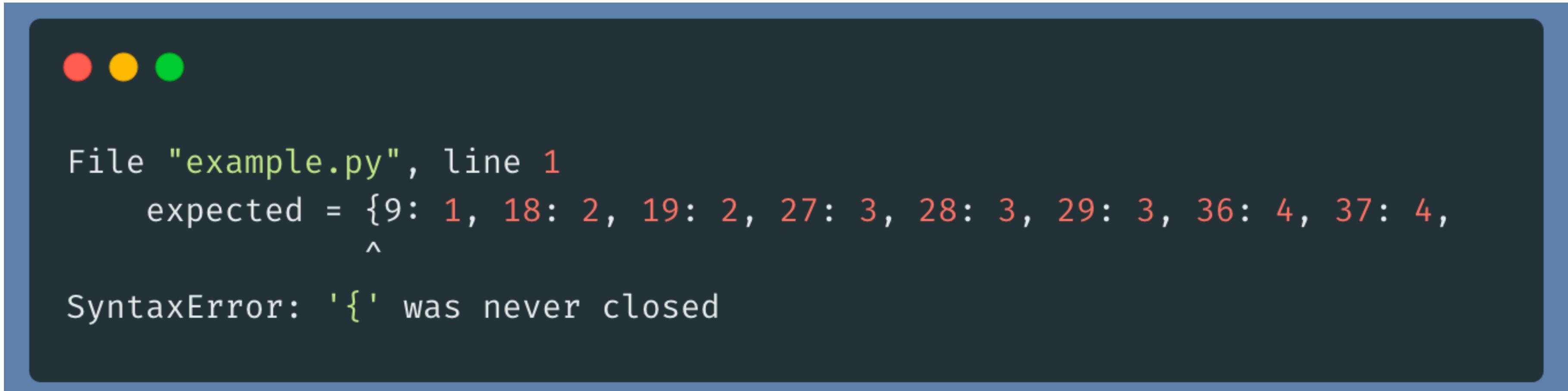
Les nouveautés de Python 3.10

Better error messages

Après



```
expected = {9: 1, 18: 2, 19: 2, 27: 3, 28: 3, 29: 3, 36: 4, 37: 4,
            38: 4, 39: 4, 45: 5, 46: 5, 47: 5, 48: 5, 49: 5, 54: 6,
some_other_code = foo()
```



```
File "example.py", line 1
    expected = {9: 1, 18: 2, 19: 2, 27: 3, 28: 3, 29: 3, 36: 4, 37: 4,
                 ^
SyntaxError: '{' was never closed
```

Les nouveautés de Python 3.10

<https://www.python.org/downloads/release/python-3100/>

Les variables

Affectation de variable

Une variable est un nom attaché à un objet particulier.

Une variable Python est un nom symbolique qui est une référence ou un pointeur vers un objet.

Une fois qu'un objet est affecté à une variable, vous pouvez faire référence à l'objet par ce nom. Mais les données elles-mêmes sont toujours contenues dans l'objet.

Affectation de variable

En Python, tout est traité comme un objet. Chaque objet possède ces trois attributs :

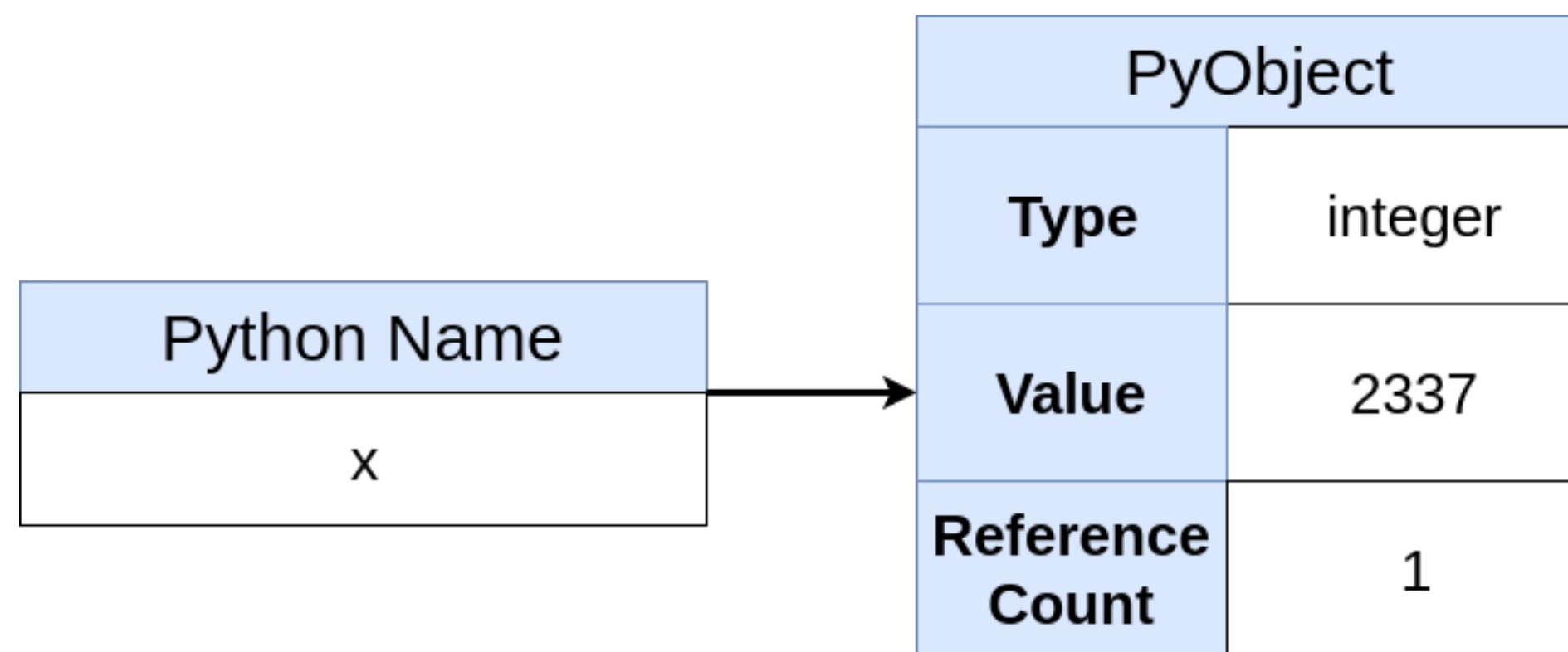
Identité (id): Il s'agit de l'adresse à laquelle l'objet se réfère dans la mémoire de l'ordinateur.

Type : Il s'agit du type d'objet qui est créé. Par exemple, un nombre entier, une liste, une chaîne de caractères, etc.

Valeur : Il s'agit de la valeur stockée par l'objet. Par exemple, List=[1,2,3] contient les nombres 1, 2 et 3.

Alors que l'ID et le type ne peuvent pas être modifiés une fois l'objet créé, **les valeurs peuvent être modifiées pour les objets mutables**.

Affectation de variable



Mutable, Immutable

Mutable: désigne la capacité des objets à modifier leurs valeurs. Ce sont souvent les objets qui stockent une collection de données.

Immutable: si la valeur d'un objet ne peut pas être modifiée dans le temps, alors il est dit immuable. Une fois créé, la valeur de ces objets est permanente.

TP

Les structures de données

Définition des slices

Le slicing est une fonctionnalité qui permet d'accéder à des parties de séquences comme les chaînes de caractères, les tuples et les listes.

Il est également possible de les utiliser pour modifier ou supprimer les éléments de séquences mutables telles que les listes.

Définition des slices

Lecture de slices

```
● ● ●  
word[0:2] # characters from position 0 (included) to 2 (excluded)  
  
word[2:5] # characters from position 2 (included) to 5 (excluded)
```

Définition des slices

Ecriture de slice

```
● ● ●  
  
letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g']  
  
# replace some values  
letters[2:5] = ['C', 'D', 'E']  
# ['a', 'b', 'C', 'D', 'E', 'f', 'g']
```

Travail avec les slices

slice notation

```
● ● ●

a[start:stop]    # items start through stop-1
a[start:]        # items start through the rest of the array
a[:stop]          # items from the beginning through stop-1
a[:]              # a copy of the whole array

a[-1]             # last item in the array
a[-2:]            # last two items in the array
a[:-2]             # everything except the last two items

a[::-1]            # all items in the array, reversed
a[1::-1]           # the first two items, reversed
a[:-3:-1]          # the last two items, reversed
a[-3::-1]           # everything except the last two items, reversed
```

Travail avec les slices

La méthode `__getitem__`

`__getitem__()` est une méthode magique.

Lorsqu'elle est utilisée dans une classe, elle permet à ses instances d'utiliser les opérateurs `[]`.

Si `x` est une instance de cette classe, alors `x[i]` est équivalent à
`type(x).__getitem__(x, i)`

Travail avec les slices

La méthode `__getitem__`

```
● ● ●

class UneSequence:
    def __getitem__(self, index):
        print(index)

seq = UneSequence()
print(seq[1:3])

# slice(1, 3, None)
# None
```

Structure de données avancées

ChainMap

ChainMap est fournie pour relier rapidement un certain nombre de dictionnaires afin qu'ils puissent être traités comme une seule unité.

C'est souvent beaucoup plus rapide que de créer un nouveau dictionnaire et d'exécuter plusieurs appels à update().

Structure de données avancées

ChainMap

```
#!/usr/bin/env python
import collections
from pprint import pprint

baseline = {'music': 'bach', 'art': 'rembrandt'}
adjustments = {'art': 'van gogh', 'opera': 'carmen'}
cm = collections.ChainMap(adjustments, baseline)
pprint(cm)
pprint(dict(cm))

# ChainMap({'art': 'van gogh', 'opera': 'carmen'},
#           {'art': 'rembrandt', 'music': 'bach'})
# {'art': 'van gogh', 'music': 'bach', 'opera': 'carmen'}
```

Structure de données avancées

Counter

Un **Counter** est une sous-classe de dict pour compter les objets hashables.

Il s'agit d'une collection où les éléments sont stockés sous forme de clés de dictionnaire et où leur nombre est stocké sous forme de valeurs de dictionnaire. Les nombres peuvent être des valeurs entières, y compris des nombres nuls ou négatifs. La classe Counter est similaire aux bags ou aux multisets dans d'autres langages.

Les éléments sont comptés à partir d'un itérable ou initialisés à partir d'un autre mapping.

Structure de données avancées

Counter

```
● ● ●

import collections
from pprint import pprint

# a new, empty counter
c = collections.Counter()

# a new counter from an iterable
c = collections.Counter('Python Perfectionnement')
pprint(c)
print(c['e'])

# a new counter from a mapping
c = collections.Counter({'python': len('python'), 'perfectionnement':
len('perfectionnement')})
pprint(c)
print(c['python'])

# a new counter from keyword args
c = collections.Counter(python=len('python'),
perfectionnement=len('perfectionnement')) # a new counter
from keyword args
pprint(c)
print(c['python'])
```

Structure de données avancées

Collections

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

Les structures de contrôles

For...else

Les boucles **for** ont une clause **else**.

La clause **else** s'exécute après que la boucle se termine **normalement**.

Cela signifie que la boucle n'a pas rencontré d'instruction **break**.

For...else



```
values = [i for i in range(1,10,2)]
print(values)

for v in values:
    if v % 2 == 0:
        print(v)
        break
    else:
        print("pas de valeur pair")

# [1, 3, 5, 7, 9]
# pas de valeur pair
```

While...else

Les boucles **while** ont une clause **else**.

La clause **else** s'exécute après que la boucle se termine normalement.

Cela signifie que la boucle n'a pas rencontré d'instruction **break**.

While...else

```
x = 1
while x < 10:
    print(x,end=", ")
    x +=2
    if x % 2 == 0:
        print(x)
        break
else:
    print("Fin")

# 1, 3, 5, 7, 9, Fin
```

Structural Pattern Matching

PEP 634

Le **pattern matching process** prend en entrée un motif (**case**) et une valeur de sujet (**match**).

Les expressions suivantes décrivent le processus :

« le motif est comparé à (ou contre) la valeur du sujet" et "nous comparons le motif à (ou avec) la valeur du sujet".

Structural Pattern Matching

PEP 634



```
value = 12

match value:
    case 1:
        print("Un")
    case 2:
        print("Deux")
    case 3:
        print("Trois")
    case _:
        print("Autre")

# Autre
```

try ... except ... else ... finally

Le bloc **try** permet de tester un bloc de code pour détecter les erreurs.

Le bloc **except** vous permet de traiter l'erreur.

Le bloc **finally** vous permet d'exécuter le code, quel que soit le résultat des blocs try et except.

try ... except ... else ... finally

```
try:  
    b=0  
    a = int(input("Please enter a number: "))  
    c= call_division(b,a)  
    print(c)  
except ZeroDivisionError as e:  
    print("ZeroDivisionError",e)  
except ValueError as e:  
    print("ValueError",e)  
except Exception as e:  
    print("Exception",e.__class__.__name__,e)  
finally:  
    print("OK")# 2
```

Les fonctions

Les Fonctions

Introduction

En Python, les fonctions sont des « **first-class citizens** ».

Cela signifie qu'elles sont sur un pied d'égalité avec tout autre objet, comme les nombres, les chaînes de caractères, les listes, les tuples, les modules, etc.

Vous pouvez les créer ou les détruire dynamiquement, les stocker dans des structures de données, les passer comme arguments à d'autres fonctions, les utiliser comme valeurs de retour, etc.

Les différents types de passage de paramètres

Valeur et référence

Un paramètre d'entrée est une donnée fournie par le code appelant au code appelé. Cette donnée peut être transmise de deux façons :

passage par copie (aussi appelé par **valeur**) : le code appelé dispose d'une copie de la valeur qu'il peut modifier sans affecter l'information initiale dans le code appelant

passage par adresse (aussi appelé par **référence**) : le code appelé dispose d'une information lui permettant d'accéder en mémoire à la valeur que le code appelant cherche à lui transmettre. Il peut alors modifier cette valeur là où elle se trouve ; le code appelant aura accès aux modifications faites sur la valeur. Dans ce cas, le paramètre peut aussi être utilisé comme un paramètre de sortie.

Les différents types de passage de paramètres

Passage par valeur

Un paramètre d'entrée est une donnée fournie par le code appelant au code appelé. Cette donnée peut être transmise de deux façons :

passage par copie (aussi appelé par **valeur**) : le code appelé dispose d'une copie de la valeur qu'il peut modifier sans affecter l'information initiale dans le code appelant.

passage par adresse (aussi appelé par **référence**) : le code appelé dispose d'une information lui permettant d'accéder en mémoire à la valeur que le code appelant cherche à lui transmettre. Il peut alors modifier cette valeur là où elle se trouve ; le code appelant aura accès aux modifications faites sur la valeur. Dans ce cas, le paramètre peut aussi être utilisé comme un paramètre de sortie.

Les différents types de passage de paramètres

Passage par référence

Une référence est une valeur qui est un moyen d'accéder en lecture et/ou écriture à une donnée.

Une référence n'est pas la donnée elle-même mais seulement une information sur sa localisation.

Les références sont souvent vues comme des noms identifiant des données. Plusieurs noms peuvent référencer la même donnée (aliasing), ce qui peut provoquer des effets de bord indésirés.

L'effet le plus classique consiste à libérer l'objet par l'une de ses références sans invalider les autres références, ce qui entraîne une erreur de segmentation, lors de l'utilisation des autres références non libérées et non invalidées.

Les différents types de passage de paramètres

Passage par référence

CPython utilise aujourd'hui un mécanisme de compteur de références.

del x n'appelle pas directement **x.__del__()**: la première décrémente le compteur de références de **x**. La seconde n'est appelée que quand le compteur de références de **x** atteint zéro.

Les différents types de passage de paramètres

Notion d'Affectation

Une **affectation**, (ou **assignation**), est une structure qui permet d'attribuer une valeur à une variable.

Il s'agit d'une structure particulièrement courante en programmation impérative, et dispose souvent pour cette raison d'une notation courte comme `x = expr` ou `x := expr`.

Dans certains langages, le symbole est considéré comme un opérateur d'affectation, et la structure entière peut alors être utilisée comme une expression. D'autres langages considèrent une affectation comme une instruction et ne permettent pas cet usage.

Portée des variables

local,nonlocal,global : règle LEGB

La notion de portée régit la manière dont les variables et les noms sont recherchés dans votre code.

Elle détermine la visibilité d'une variable dans le code. La portée d'un nom ou d'une variable dépend de l'endroit où vous créez cette variable.

Le concept de portée Python est généralement présenté à l'aide d'une règle connue sous le nom de règle LEGB.

Portée des variables

La règle LEGB

La règle LEGB est une procédure de recherche de noms, qui détermine l'ordre dans lequel Python recherche les noms.

Local (function) scope : bloc de code ou le corps de toute fonction Python ou expression lambda.

Enclosing (nonlocal) scope : portée spéciale qui n'existe que pour les fonctions imbriquées.

Global (module) scope : portée la plus élevée dans un programme, un script ou un module Python. Cette portée Python contient tous les noms que vous définissez au niveau supérieur d'un programme ou d'un module

Built-in scope : portée Python spéciale qui est créée ou chargée lorsque vous exécutez un script ou ouvrez une session interactive

Fonction d'ordre supérieur (Higher-order function)

Définition

Les fonctions d'ordre supérieur sont des fonctions qui ont au moins une des propriétés suivantes :

- elles prennent une ou plusieurs fonctions en entrée ;
- elles renvoient une fonction.

La fonction map présente dans de nombreux langages de programmation fonctionnelle est un exemple de fonction d'ordre supérieur : elle prend une fonction f comme argument, et retourne une nouvelle fonction qui prend une liste comme argument et applique f à chaque élément.

Un autre exemple très courant est celui d'une fonction de tri qui prend en argument une fonction de comparaison ; on sépare ainsi l'algorithme de tri de la comparaison des éléments à trier.

Les Closures (Fermetures)

Définition

Les fermetures sont des fonctions créées dynamiquement qui sont retournées par d'autres fonctions.

Leur principale caractéristique est qu'elles ont un accès complet aux variables et aux noms définis dans l'espace de noms local où la fermeture a été créée, même si la fonction englobante est retournée et a fini de s'exécuter.

En Python, lorsque vous renvoyez un objet de fonction interne, l'interpréteur emballle la fonction avec son environnement ou sa fermeture. L'objet fonction conserve un instantané de toutes les variables et de tous les noms définis dans son environnement contenant. Pour définir une fermeture, vous devez suivre trois étapes :

- Créer une fonction interne
- Référencer les variables de la fonction de fermeture
- Retourner la fonction interne

L'introspection

L'introspection

Définition

La réflexion est la capacité d'un programme à examiner, et éventuellement à modifier, ses propres structures internes lors de son exécution.

On distingue deux techniques utilisées par les systèmes réflexifs :

- **l'introspection** : la capacité d'un programme à examiner son propre état ;
- **l'intercession** : la capacité d'un programme à modifier son propre état d'exécution ou d'altérer sa propre interprétation ou signification.

Ce principe permet de considérer des modules et des fonctions en mémoire comme des objets, d'obtenir des informations de leur part et de les manipuler.

L'introspection

Le module inspect

Le module `inspect` fournit des fonctions permettant de connaître les objets en cours d'exécution, notamment les modules, les classes, les instances, les fonctions et les méthodes.

Il est possible d'utiliser les fonctions de ce module pour récupérer le code source original d'une fonction, examiner les arguments d'une méthode sur la pile et extraire le type d'informations utiles pour produire une documentation de bibliothèque pour votre code source.

Les Générateurs

Les Générateurs (PEP 255)

Introduction

Les fonctions génératrices sont un type particulier de fonctions qui renvoient un itérateur paresseux (**lazy iterator**).

Un itérateur est un objet (et un design pattern) sur lequel vous pouvez boucler comme une liste.

Cependant, contrairement aux listes, les **lazy iterator** ne stockent pas leur contenu en mémoire.

Generator expression (Generator comprehension)

```
● ● ●  
csv_gen = (row for row in open(file_name))
```

Yield

yield est une instruction assez simple. Son rôle principal est de contrôler le flux d'une fonction de générateur d'une manière similaire aux instructions return.

Pour rappel :

Lorsque vous appelez une fonction de générateur ou que vous utilisez une expression de générateur, vous renvoyez un itérateur spécial appelé générateur. **Lorsque vousappelez des méthodes spéciales sur le générateur, comme next(), le code de la fonction est exécuté jusqu'à yield.**

Lorsque l'instruction Python yield est atteinte, le programme suspend l'exécution de la fonction et renvoie la valeur cédée à l'appelant.

Lorsqu'une fonction est suspendue, l'état de cette fonction est sauvegardé.

Cela inclut toute liaison de variable locale au générateur, le pointeur d'instruction, la pile interne et toute gestion d'exception.

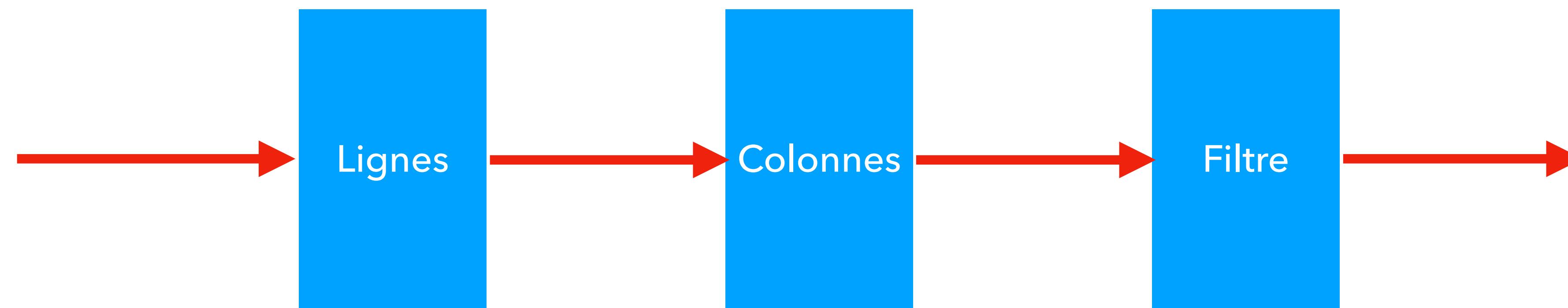
Cela vous permet de reprendre l'exécution de la fonction chaque fois que vous appelez l'une des méthodes du générateur.

De cette façon, toute l'évaluation de la fonction reprend immédiatement après le **yield**.

Pipeline de donnée avec les générateurs

Définition

Les pipelines de données vous permettent d'enchaîner du code pour traiter de grands ensembles de données ou des flux de données sans saturer la mémoire de votre machine.



Programmation Orientée Objet

Méthode de classe et d'instances

Méthodes d'instance

C'est le type de méthode de base. La méthode prend un paramètre, **self**, qui pointe vers une instance de UneClass lorsque la méthode est appelée. Grâce au paramètre self, les méthodes d'instance peuvent accéder librement aux attributs et aux autres méthodes du même objet (self !).

Non seulement elles peuvent modifier l'état de l'objet, mais les méthodes d'instance peuvent également accéder à la classe elle-même grâce à l'attribut `self.__class__`. Cela signifie que les méthodes d'instance peuvent également modifier l'état de la classe.

Méthodes de classe

Au lieu d'accepter un paramètre **self**, les méthodes de classe prennent un paramètre **cls** qui pointe vers la classe.

Comme la méthode de classe n'a accès qu'à cet argument **cls**, elle ne peut pas modifier l'état de l'instance de l'objet. Cela nécessiterait un accès à **self**. Cependant, les méthodes de classe peuvent toujours modifier l'état de la classe qui s'applique à toutes les instances de la classe.

Méthodes statiques

Ce type de méthode ne prend **ni un paramètre self ni un paramètre cls**.

Par conséquent, une méthode statique ne peut modifier ni l'état de l'objet ni celui de la classe. Les méthodes statiques sont limitées dans les données auxquelles elles peuvent accéder - et elles sont principalement un moyen de créer des fonctions utilitaires.

Méthode de classe et d'instances

```
class UneClass:
    def methodeInstance(self):
        return "methode d'instance", self

    @classmethod
    def methodeDeClass(cls):
        return 'méthode de class', cls

    @staticmethod
    def methodeStatic():
        return 'méthode static'
```

Les propriétés (property)

En programmation orientée objet, une **propriété** est un élément de description d'un objet.

Cela se traduit par un champ du point de vue de l'interface d'une classe, mais auquel sont adjoints une ou plusieurs méthodes, dites **accesseurs** et **mutateurs**, vouées à lire (get / accesseur) et modifier (set / mutateur) la valeur du champ de l'instance.

Le champ de la propriété peut selon le langage se traduire par un attribut, éventuellement homonyme des accesseurs (getter ou setter).

Les propriétés (property)

```
● ● ●

class Pen(object):
    def __init__(self):
        self._color = 0 # "private" variable

    @property
    def color(self):
        return self._color

    @color.setter
    def color(self, color):
        self._color = color
```

Les itérateurs

Les itérateurs

L'itérateur est un patron de conception (design pattern) comportemental.

Un itérateur est un objet qui permet de parcourir tous les éléments contenus dans un autre objet, le plus souvent un conteneur (liste, arbre, etc).

Un itérateur ressemble à un pointeur disposant essentiellement de deux primitives : **accéder à l'élément pointé** en cours (dans le conteneur), et se **déplacer pour pointer vers l'élément suivant**

Les itérateurs

En Python, un itérateur est un objet qui prend en charge les méthodes `__iter__()` et `__next__()`.

Ces méthodes fonctionnent automatiquement avec les boucles `for-in`.

Les classes et méthodes abstraites (ABC)

Classes de base abstraites

ABC (Abstract Base Class)

Les classes de base abstraites complètent le duck-typing en fournissant un moyen de définir des interfaces.

Les ABC introduisent des sous-classes virtuelles qui n'héritent pas d'une classe mais qui sont quand même reconnues par `isinstance()` ou `issubclass()`.

Python contient de nombreuses ABC :

les structures de données (dans le module `collections.abc`), les nombres (dans le module `numbers`),

les flux (dans le module `io`)

les finders et loaders du système d'importation (dans le module `importlib.abc`).

Les Interfaces avec les ABC

Une interface agit comme un modèle pour la conception de classes.

Comme les classes, les interfaces définissent des méthodes.

Contrairement aux classes, ces méthodes sont abstraites : elles sont simplement définies, pas implémentées

Le travail d'implémentation est effectué par la classe, qui implémente ensuite l'interface et donne une signification concrète aux méthodes abstraites de l'interface.

L'approche de Python en matière de conception d'interfaces est quelque peu différente de celle de langages comme Java, Go et C++. Ces langages ont tous un mot-clé interface, alors que Python n'en a pas.

Python s'écarte également des autres langages sur un autre aspect. Il n'exige pas que la classe qui implémente l'interface définisse toutes les méthodes abstraites de l'interface.

Les métaclasses

La Métaprogrammation

Le terme métaprogrammation fait référence à la possibilité pour un programme d'avoir des connaissances sur lui-même ou de le manipuler.

Python supporte une forme de métaprogrammation pour les classes appelées métaclasses.

Les métaclasses sont un concept ésotérique de la POO, qui se cache derrière pratiquement tout le code Python.

Vous les utilisez, que vous en soyez conscient ou non. Dans la plupart des cas, vous n'avez pas besoin d'en être conscient.

La Métaprogrammation

L'utilisation de métaclasses personnalisées est quelque peu controversée, comme le suggère la citation suivante de Tim Peters, le gourou de Python qui a écrit le Zen of Python :

"Les métaclasses sont de la magie plus profonde que ce dont 99% des utilisateurs ne devraient jamais s'inquiéter. Si vous vous demandez si vous en avez besoin, c'est que vous n'en avez pas besoin (les personnes qui en ont réellement besoin savent avec certitude qu'elles en ont besoin, et n'ont pas besoin qu'on leur explique pourquoi)."

- Tim Peters

Les décorateurs et Design Patterns

Les décorateurs

Un décorateur est une fonction qui prend une autre fonction et étend le comportement de cette dernière sans la modifier explicitement.

Ils enveloppent une fonction, modifiant son comportement.

Pour rappel, les fonctions sont des **objets de première classe**.

Elles peuvent être transmises et utilisées comme arguments, comme n'importe quel autre objet (chaîne de caractères, nombre entier, flottant, liste, etc.).

Les décorateurs

Décorer une fonction

```
def log_around(func):  
    def wrapper():  
        print('Log before')  
        func()  
        print('Log after')  
    return wrapper  
  
def say_hello():  
    print("Hello")  
  
say_hello = log_around(say_hello)  
  
say_hello()
```

Les décorateurs

Décorer une fonction

```
● ● ●

def log_around(func):
    def wrapper():
        print('Log before')
        func()
        print('Log after')
    return wrapper

@log_around
def say_hello():
    print("Hello")

# say_hello = log_this(say_hello)

say_hello()
```

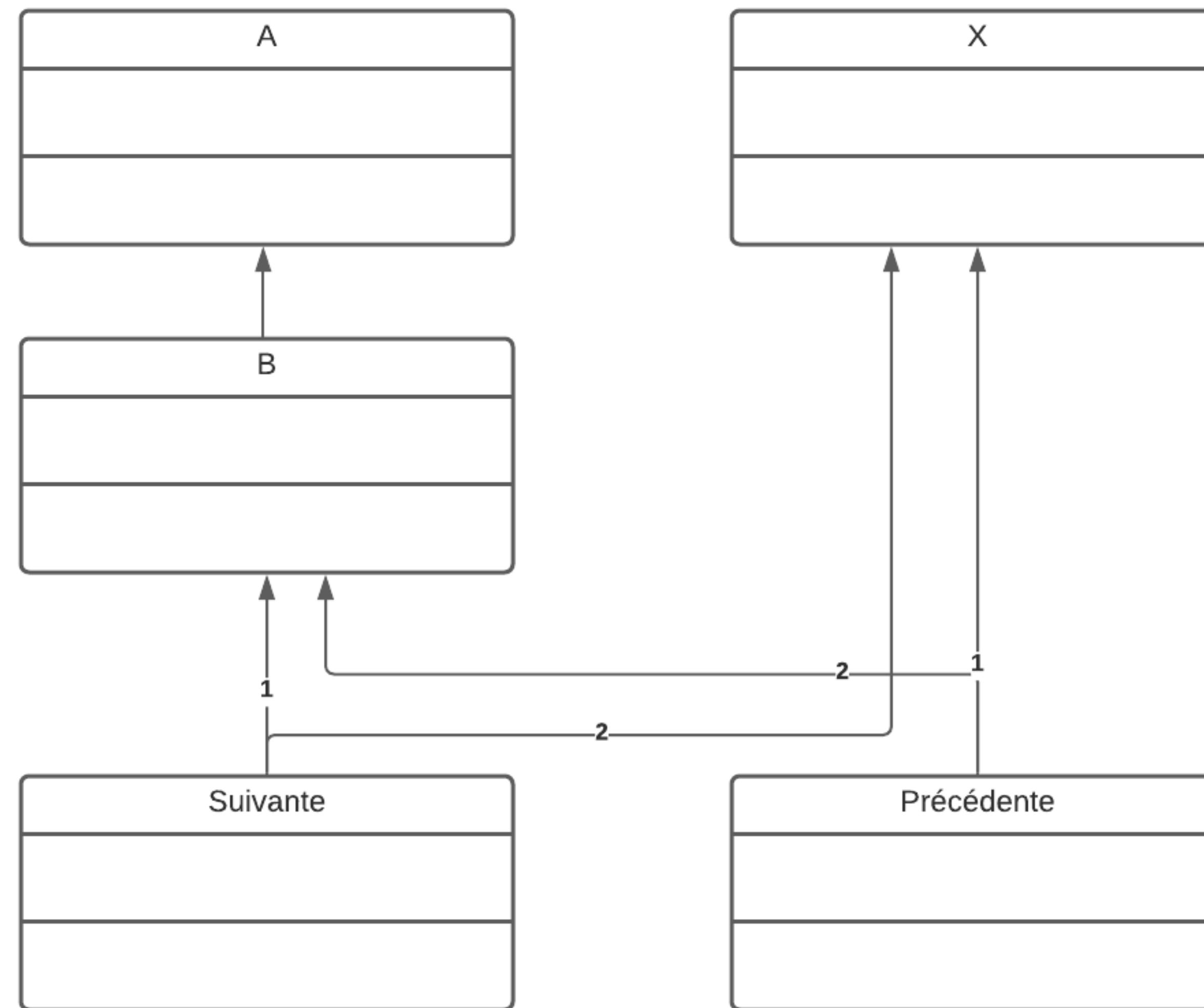
L'héritage multiple

En Python, il est possible d'hériter de plusieurs classes.

L'ordre de résolution des méthodes (**MRO**) détermine où Python recherche une méthode lorsqu'il existe une hiérarchie de classes. L'utilisation de `super()` permet d'accéder à la classe suivante dans le MRO.

L'héritage multiple

MRO



L'héritage multiple

MRO

```
● ● ●

class A:
    def __init__(self):
        print('A')
        super().__init__()

class B(A):
    def __init__(self):
        print('B')
        super().__init__()

class X:
    def __init__(self):
        print('X')
        super().__init__()

class Suivante(B, X):
    def __init__(self):
        print('Suivante')
        super().__init__()

class Precedente(X, B):
    def __init__(self):
        print('Precedente')
        super().__init__()

s = Suivante()
p = Precedente()
```

L'héritage multiple

MRO



```
~/local_dev/formations/prep/python-perf » python tp_herit_mult.py
```

Suivante

B

A

X

Precedente

X

B

A

L'héritage multiple

Problème du diamant

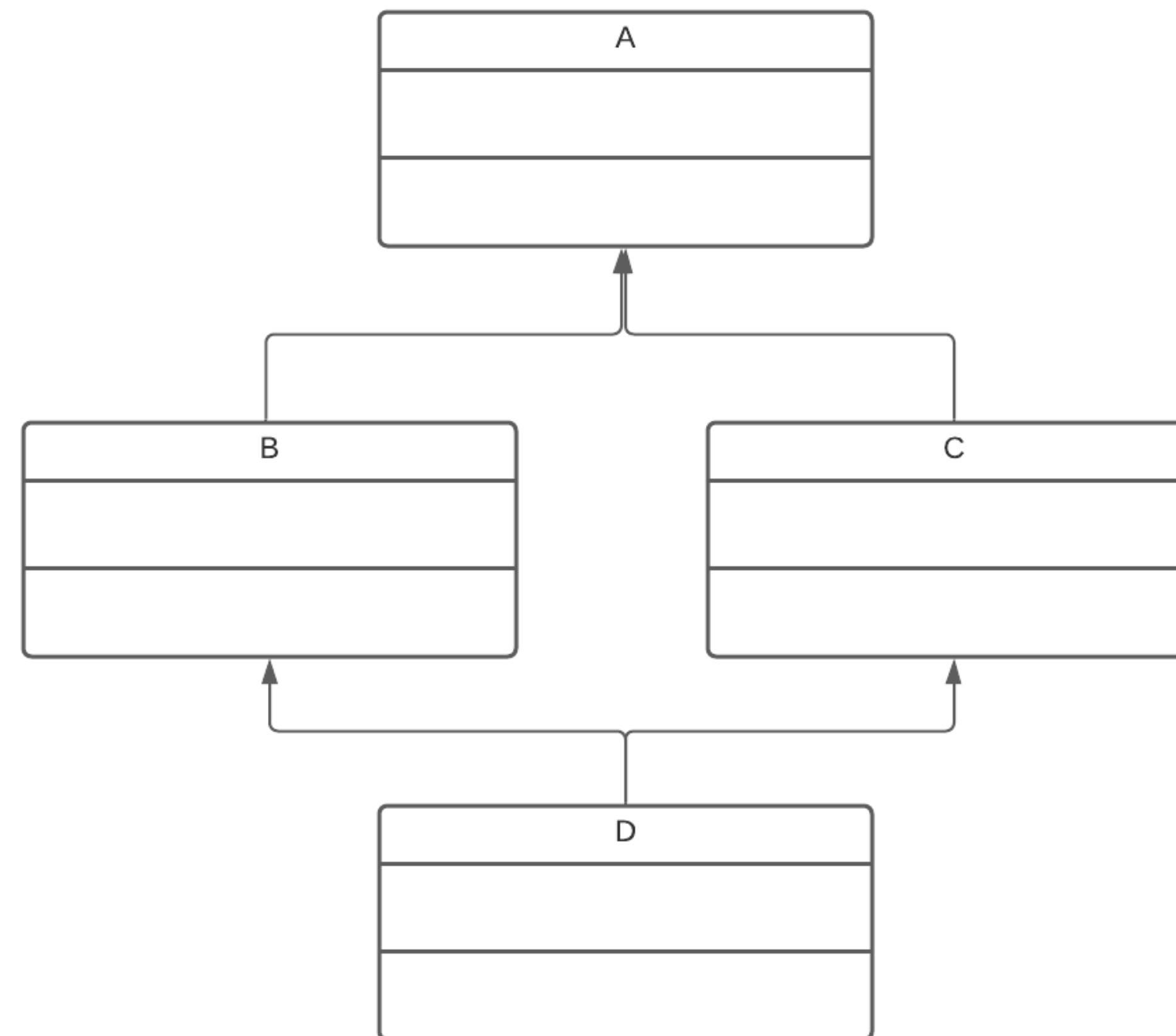
En informatique, le problème du diamant arrive principalement en programmation orientée objet, lorsque le langage permet l'héritage multiple.

Si une classe D hérite de deux classes B et C, elles-mêmes filles d'une même classe A, se pose un problème de conflit lorsque des fonctions ou des champs des classes B et C portent le même nom.

Le nom de ce problème provient de la forme du schéma d'héritage des classes A, B, C et D dans ce cas.

L'héritage multiple

Problème du diamant



L'héritage multiple

Résolution : Problème du diamant

Comment résoudre le problème du diamant ?

- en interdisant les homonymies
- **en définissant un ordre de priorité**
- en réservant l'héritage multiple aux interfaces

Déploiement et qualité

Les environnements virtuels

pip est arrivé en 2008, comme une alternative à **easy_install**, bien qu'il soit toujours largement construit sur les composants de **setuptools**. Il a introduit l'idée des fichiers de **requirements**, qui permettent aux utilisateurs de reproduire facilement des environnements.

```
● ● ●

# installation de l'environnement
pip install -m venv .venv

# activation de l'environnement Windows
.venv\Scripts\activate.bat

# activation de l'environnement Linux/macOS
.venv\bin\activate
```

Installer des dépendances



```
# installation d'une dépendance  
pip install requests
```

Sauvegarder la liste des dépendances

```
● ● ●  
#Liste des dépendances  
pip freeze  
  
# Sauvegarde des dépendances  
pip freeze > requirements.txt
```

Restaurer des dépendances



```
#Restaurer des dépendances  
pip install -r requirements.txt
```

Le Python Package Index



Le Python Package Index (PyPI) est un référentiel de logiciels pour Python.

PyPI vous aide à trouver et à installer des logiciels développés et partagés par la communauté Python.

Les auteurs de paquets utilisent PyPI pour distribuer leurs logiciels.

Wheels

Les Wheels (.whl) sont un format de distribution construit introduit par PEP 427, qui est destiné à remplacer le format Egg. Wheel est actuellement supporté par pip.

Les roues sont un composant de l'écosystème Python qui aide à faire fonctionner les installations de paquets. Elles permettent des installations plus rapides et une plus grande stabilité dans le processus de distribution des paquets.

Un fichier .whl est essentiellement une archive ZIP (.zip) avec un nom de fichier spécialement conçu qui indique aux installateurs quelles versions de Python et quelles plates-formes la roue supportera.

Une wheel est un type de distribution construite. Dans ce cas, "built" signifie que la roue est fournie dans un format prêt à l'installation et vous permet de sauter l'étape de construction requise avec les distributions sources.

Un nom de fichier de roue est décomposé en plusieurs parties séparées par des traits d'union :

```
{dist}-{version}(-{build})?-{python}-{abi}-{platform}.whl
```

Les types de Wheel

Universal wheel

Prend en charge Python 2 et Python 3 sur tous les systèmes d'exploitation et toutes les plates-formes. La majorité des roues répertoriées sur le site Python Wheels sont des roues universelles.

Pure-Python wheel

Supporte soit Python 3, soit Python 2, mais pas les deux.

Platform wheel

Elle contient des segments indiquant une version spécifique de Python, l'ABI, le système d'exploitation ou l'architecture.

Création et distribution de packages

Setuptools



```
$ python -m venv env && source ./env/bin/activate  
$ python -m pip install -U pip wheel setuptools
```

Déployer un environnement autonome

Le fichier setup.py

```
#Fichier setup.py

from setuptools import setup

setup()
```

```
$ python setup.py -h

Common commands: (see '--help-commands' for more)

  setup.py build      will build the package underneath 'build/'
  setup.py install    will install the package

Global options:
  --verbose (-v)      run verbosely (default)
  --quiet (-q)        run quietly (turns verbosity off)
  --dry-run (-n)      don't actually do anything
  --help (-h)         show detailed help message
  --no-user-cfg       ignore pydistutils.cfg in your home directory
  --command-packages  list of packages that provide distutils commands
# ...

usage: setup.py [global_opts] cmd1 [cmd1_opts] [cmd2 [cmd2_opts] ...]
or: setup.py --help [cmd1 cmd2 ...]
or: setup.py --help-commands
or: setup.py cmd --help
```

Déployer un environnement autonome

Distribution au format wheel



```
$ python setup.py bdist_wheel
```

Profilez vos programmes avec Timeit et cProfile

Timeit

La bibliothèque standard **timeit** est conçue pour mesurer le temps d'exécution de petits bouts de code.

Bien que vous puissiez importer et appeler `timeit.timeit()` depuis Python comme une fonction ordinaire, il est généralement plus pratique d'utiliser l'interface en ligne de commande.

Profilez vos programmes avec Timeit et cProfile

Timeit

timeit appelle automatiquement votre code de nombreuses fois pour faire une moyenne des mesures.

```
$ python -m timeit --setup "nums = [7, 6, 1, 4, 1, 8, 0, 6]" "set(nums)"  
2000000 loops, best of 5: 163 nsec per loop  
  
$ python -m timeit --setup "nums = [7, 6, 1, 4, 1, 8, 0, 6]" "{*nums}"  
2000000 loops, best of 5: 121 nsec per loop
```

Profilez vos programmes avec Timeit et cProfile

cProfile et **profile** fournissent un profilage déterministe des programmes Python. Un profil est un ensemble de statistiques qui décrit la fréquence et la durée d'exécution des différentes parties du programme.

La bibliothèque standard Python fournit deux implémentations différentes de la même interface de profilage :

cProfile est recommandé pour la plupart des utilisateurs ; il s'agit d'une extension C avec une surcharge raisonnable qui la rend appropriée pour le profilage de programmes longs.

profile, un module Python pur dont l'interface est imitée par cProfile, mais qui ajoute une surcharge significative aux programmes profilés.

Profilez vos programmes avec Timeit et cProfile

cProfile

```
● ● ●

import cProfile
import re

def main():
    # Output stdout
    cProfile.run('re.compile("foo|bar")')

    # Output to restats file
    cProfile.run('re.compile("foo|bar")', 'restats')

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

Le parallélisme

Multithreading et Multiprocessing

Définition

Un **thread** est similaire à un **processus**, car tous deux représentent l'exécution d'un ensemble d'instructions du langage machine d'un processeur.

Du point de vue de l'utilisateur, ces exécutions semblent se dérouler en parallèle.

Toutefois, là où chaque processus possède sa propre mémoire virtuelle, les threads d'un même processus se partagent sa mémoire virtuelle.

Par contre, tous les threads possèdent leur propre pile d'exécution.

Deux processus sont totalement indépendants et isolés l'un de l'autre.

Ils ne peuvent interagir qu'à travers une API fournie par le système, telle qu'IPC, tandis que les threads partagent une information sur l'état du processus, des zones de mémoires, ainsi que d'autres ressources.

Multithreading et Multiprocessing

Multiprocessing

```
● ● ●  
from multiprocessing import Pool  
  
def f(x):  
    return x*x  
  
if __name__ == '__main__':  
    with Pool() as p:  
        print(p.map(f, [1, 2, 3]))
```

Celery

Docker

Docker est une plateforme permettant de lancer certaines applications dans des conteneurs logiciels.

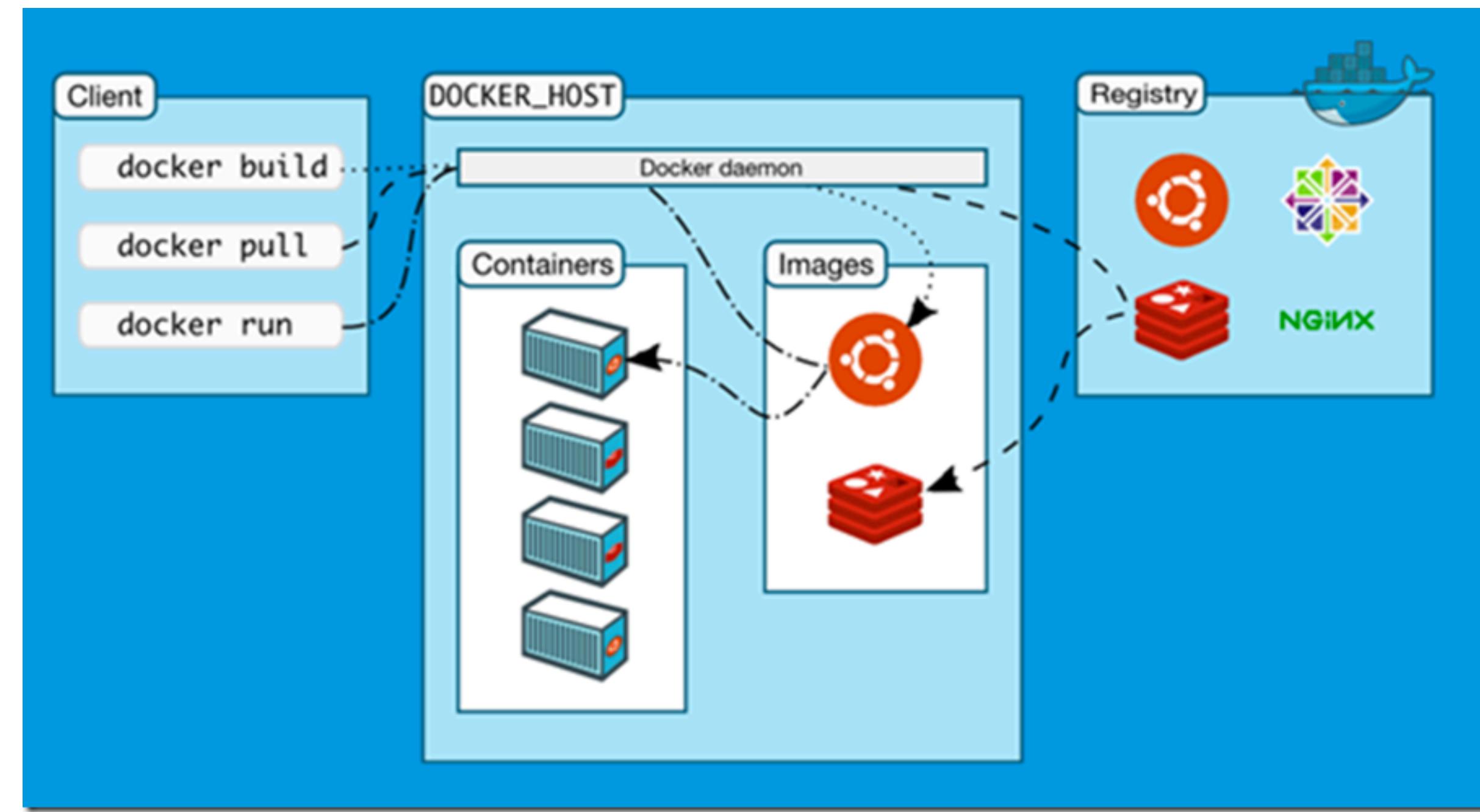
Selon la firme de recherche sur l'industrie 451 Research, « Docker est un outil qui peut empaqueter une application et ses dépendances dans un conteneur isolé, qui pourra être exécuté sur n'importe quel serveur ».

Il ne s'agit pas de virtualisation, mais de conteneurisation, une forme plus légère qui s'appuie sur certaines parties de la machine hôte pour son fonctionnement.

Techniquement, Docker étend le format de conteneur Linux standard, LXC, avec une API de haut niveau fournissant une solution pratique de virtualisation qui exécute les processus de façon isolée². Pour arriver à ses fins, Docker utilise entre autres LXC, cgroups et le noyau Linux lui-même.

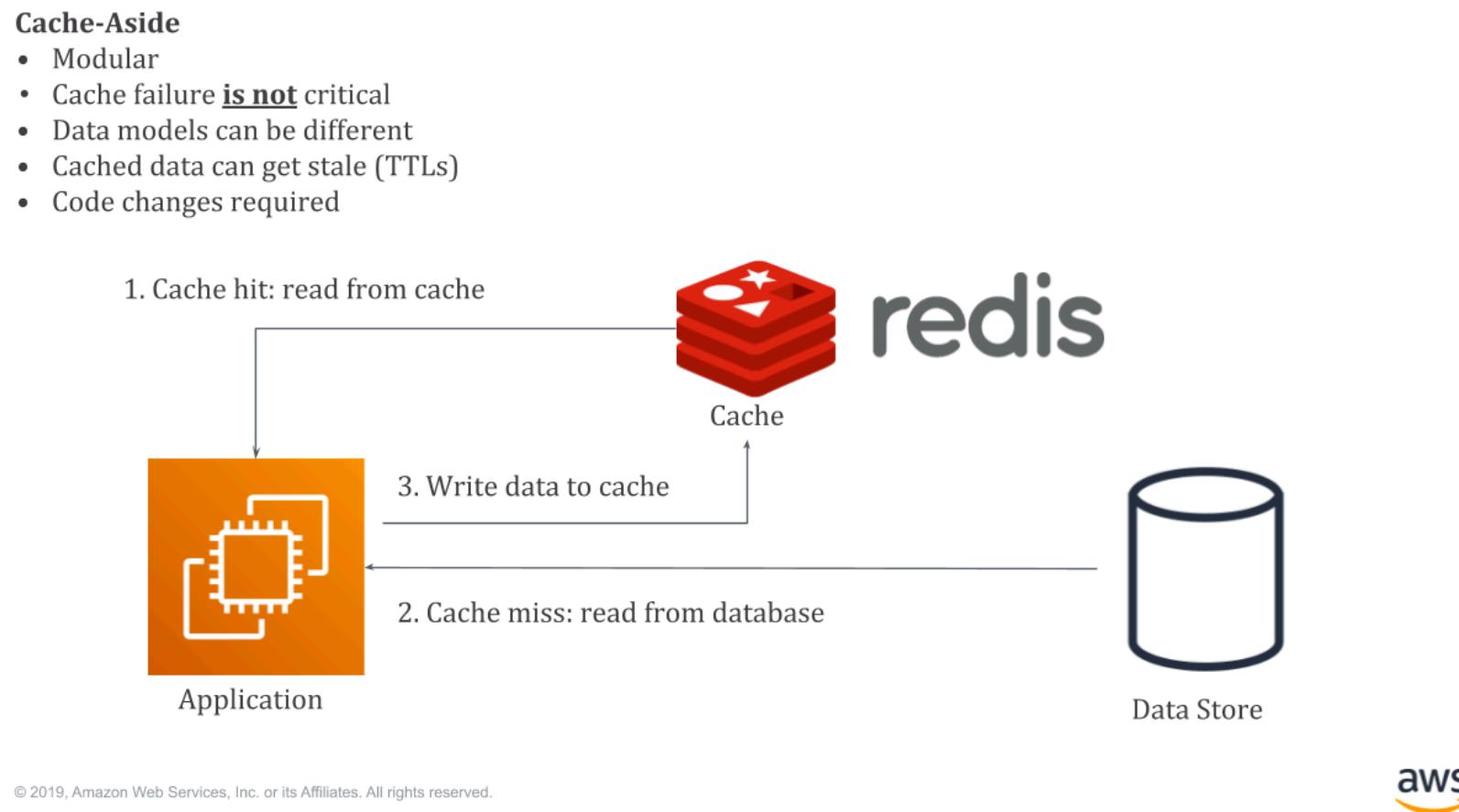
Contrairement aux machines virtuelles traditionnelles, un conteneur Docker n'inclut pas de système d'exploitation, mais s'appuie au contraire sur les fonctionnalités du système d'exploitation fournies par la machine hôte.

Docker



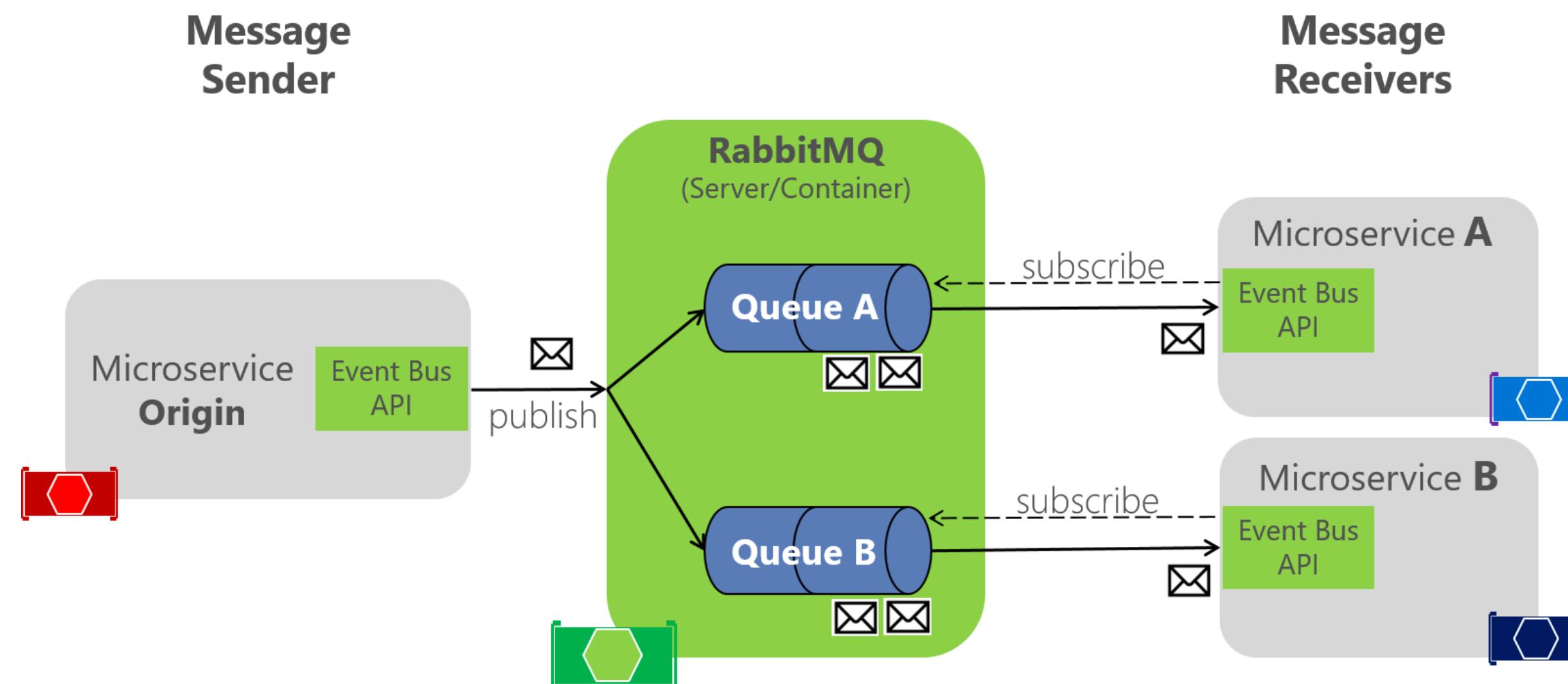
RabbitMQ

Redis (de l'anglais REmote DIctionary Server) est un système de gestion de base de données clé-valeur extensible, très hautes performances, écrit en C ANSI et distribué sous licence BSD. Il fait partie de la mouvance NoSQL et vise à fournir les performances les plus élevées possibles.

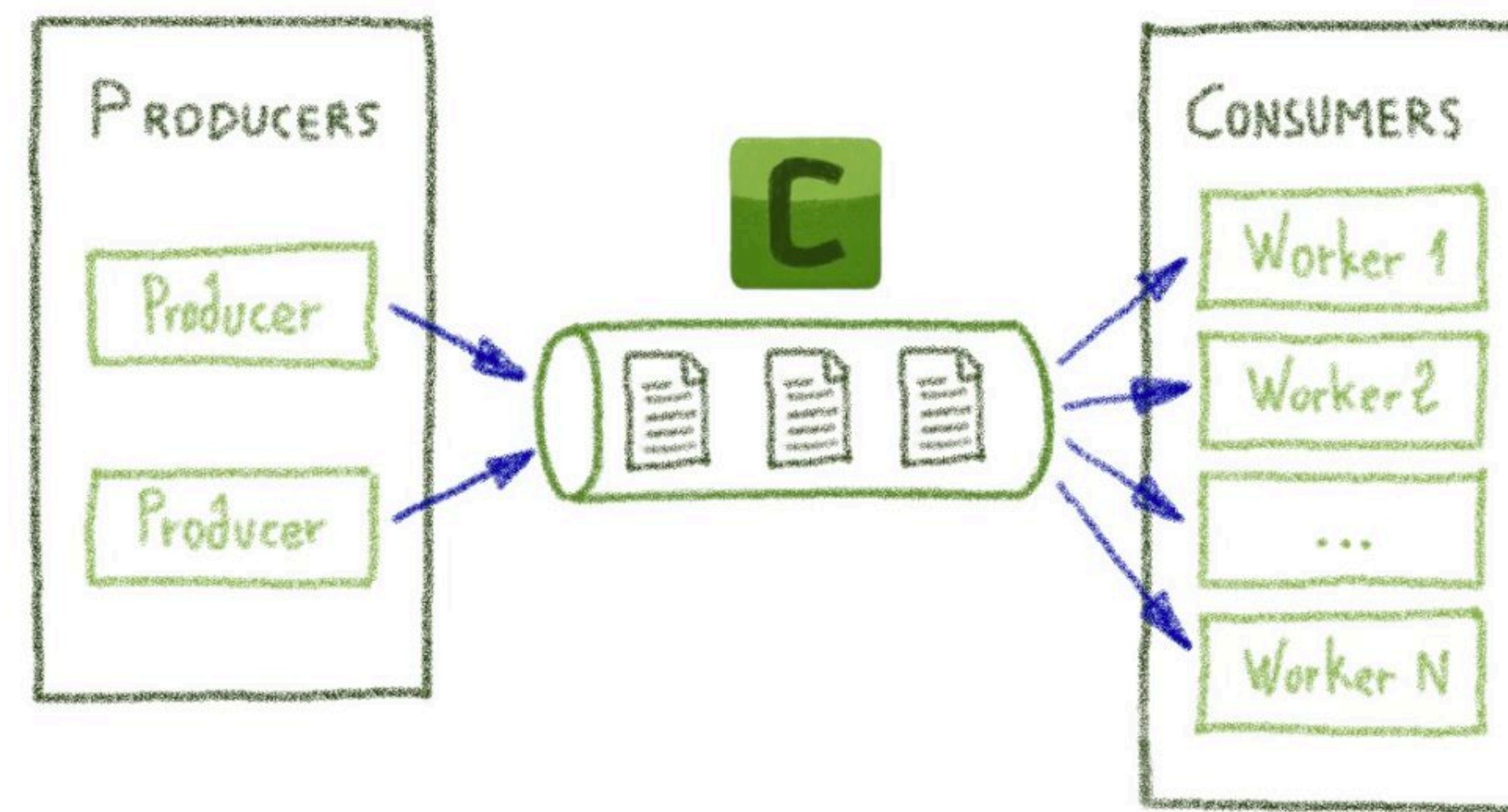


Redis

RabbitMQ est un logiciel d'agent de messages open source qui implémente le protocole Advanced Message Queuing (AMQP), mais aussi avec des plugins Streaming Text Oriented Messaging Protocol (STOMP) et Message Queuing Telemetry Transport (MQTT).



Celery



Celery

Celery est un système distribué simple, flexible et fiable pour traiter de grandes quantités de messages, tout en fournissant aux opérations les outils nécessaires à la maintenance d'un tel système.

Il s'agit d'une file d'attente de tâches axée sur le traitement en temps réel, tout en prenant également en charge l'ordonnancement des tâches.

Celery possède une communauté importante et diversifiée d'utilisateurs et de contributeurs, vous devriez venir nous rejoindre sur IRC ou sur notre liste de diffusion.

Celery

```
# tasks.py
from celery import Celery

app = Celery("tasks", broker="redis://127.0.0.1:6379/0")

@app.task
def add(x, y):
    return x + y
```

Calcul scientifique et statistique

Numpy

NumPy est une bibliothèque pour langage de programmation Python, destinée à manipuler des matrices ou tableaux multidimensionnels ainsi que des fonctions mathématiques opérant sur ces tableaux.

Plus précisément, cette bibliothèque logicielle libre et open source fournit de multiples fonctions permettant notamment de créer directement un tableau depuis un fichier ou au contraire de sauvegarder un tableau dans un fichier, et manipuler des vecteurs, matrices et polynômes.

NumPy est la base de SciPy, regroupement de bibliothèques Python autour du calcul scientifique.

Pandas

Pandas est une bibliothèque permettant la manipulation et l'analyse des données. Elle propose en particulier des structures de données et des opérations de manipulation de tableaux numériques et de séries temporelles.

Pandas est un logiciel libre sous licence BSD. Son nom est dérivé du terme "données de panel", un terme d'économétrie pour les jeux de données qui comprennent des observations sur plusieurs périodes de temps pour les mêmes individus. Son nom est également un jeu de mots sur l'expression "analyse de données Python".

Scipy

SciPy est un projet visant à unifier et fédérer un ensemble de bibliothèques Python à usage scientifique. Scipy utilise les tableaux et matrices du module NumPy.

Il offre des possibilités avancées de visualisation grâce au module matplotlib.

Matplotlib

Matplotlib est une bibliothèque du langage de programmation Python destinée à tracer et visualiser des données sous formes de graphiques. Elle peut être combinée avec les bibliothèques python de calcul scientifique NumPy et SciPy.

Plusieurs points rendent cette bibliothèque intéressante :

- Export possible en de nombreux formats matriciels (PNG, JPEG...) et vectoriels (PDF, SVG...)
- Documentation en ligne en quantité, nombreux exemples disponibles sur internet
- Forte communauté très active
- ...

Algorithmes d'apprentissage (IA)

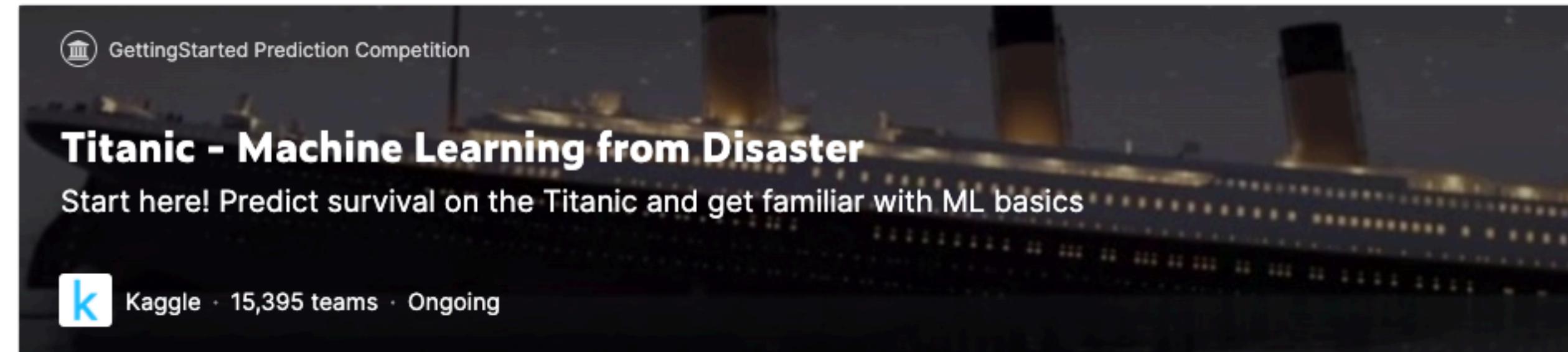
Scikit-learn

Scikit-learn est une bibliothèque libre Python destinée à l'apprentissage automatique. Elle est développée par de nombreux contributeurs notamment dans le monde académique par des instituts français d'enseignement supérieur et de recherche comme Inria.

Elle propose dans son framework de nombreuses bibliothèques d'algorithmes à implémenter, clé en main. Ces bibliothèques sont à disposition notamment des data scientists. Elle comprend notamment des fonctions pour estimer des forêts aléatoires, des régressions logistiques, des algorithmes de classification, et les machines à vecteurs de support.

Elle est conçue pour s'harmoniser avec d'autres bibliothèques libres Python, notamment NumPy et SciPy.

Kaggle et le Titanic



Traitements XML

W3C SAX

Simple API For XML

SAX ou Simple API for XML est une interface de programmation pour de nombreux langages permettant de lire et de traiter des documents XML.

L'API SAX est originellement spécifique au langage de programmation Java qui l'utilise pour lire une portion ou la totalité d'un document XML. Elle a pu par la suite être adoptée par la plupart des langages de programmation actuels.

W3C DOM

Document Object Model

Le Document Object Model (DOM) est une interface de programmation normalisée par le W3C, qui permet à des scripts d'examiner et de modifier le contenu du navigateur web.

Par le DOM, la composition d'un document HTML ou XML est représentée sous forme d'un jeu d'objets reliés selon une structure en arbre. À l'aide du DOM, un script peut modifier le document présent dans le navigateur en ajoutant ou en supprimant des nœuds de l'arbre.

ElementTree

```
● ● ●  
<?xml version="1.0"?>  
<data>  
    <country name="Liechtenstein">  
        <rank>1</rank>  
        <year>2008</year>  
        <gdppc>141100</gdppc>  
        <neighbor name="Austria" direction="E"/>  
        <neighbor name="Switzerland" direction="W"/>  
    </country>  
    <country name="Singapore">  
        <rank>4</rank>  
        <year>2011</year>  
        <gdppc>59900</gdppc>  
        <neighbor name="Malaysia" direction="N"/>  
    </country>  
    <country name="Panama">  
        <rank>68</rank>  
        <year>2011</year>  
        <gdppc>13600</gdppc>  
        <neighbor name="Costa Rica" direction="W"/>  
        <neighbor name="Colombia" direction="E"/>  
    </country>  
</data>
```

```
● ● ●  
import xml.etree.ElementTree as ET  
  
def main():  
    tree = ET.parse('country_data.xml')  
    root = tree.getroot()  
    print(root.tag)  
    for child in root:  
        print(child.tag, child.attrib)  
  
if __name__ == '__main__':  
    main()
```

```
● ● ●  
$ python tp_xml.py  
  
data  
country {'name': 'Liechtenstein'}  
country {'name': 'Singapore'}  
country {'name': 'Panama'}
```

Programmation Réseau

Twisted

Twisted est un moteur de mise en réseau piloté par événements.

Twisted

Serveur Web

Twisted est



```
from twisted.web import server, resource
from twisted.internet import reactor, endpoints

class Counter(resource.Resource):
    isLeaf = True
    numberRequests = 0

    def render_GET(self, request):
        self.numberRequests += 1
        request.setHeader(b"content-type", b"text/plain")
        content = u"I am request #{}\n".format(self.numberRequests)
        return content.encode("ascii")

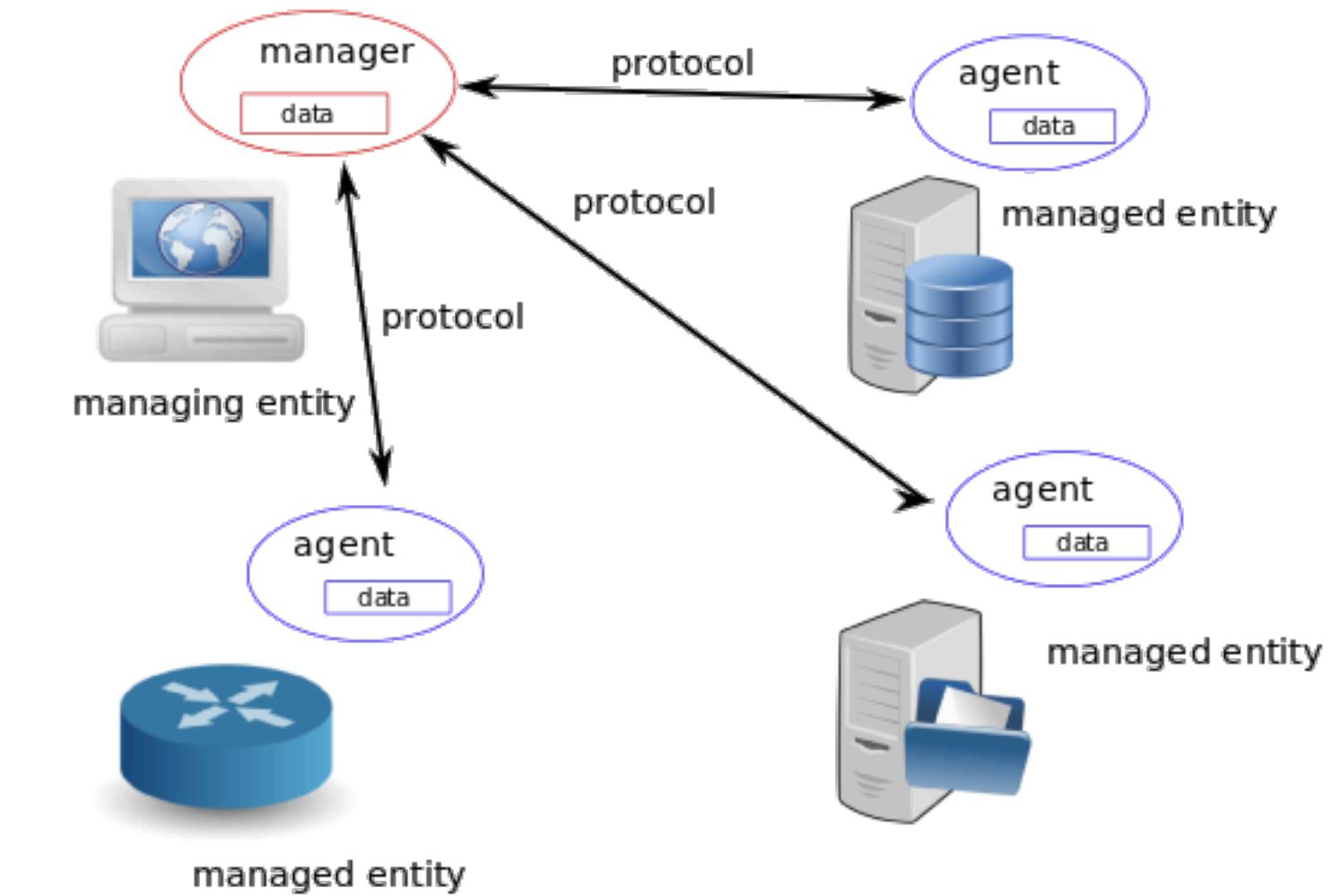
endpoints.serverFromString(reactor,
    "tcp:8080").listen(server.Site(Counter()))
reactor.run()
```

ments.

PySNMP

SNMP

Simple Network Management Protocol (abrégé SNMP), en français « protocole simple de gestion de réseau », est un protocole de communication qui permet aux administrateurs réseau de gérer les équipements du réseau, de superviser et de diagnostiquer des problèmes réseaux et matériels à distance.



Conclusion