wevioo

VOYAGE AU CŒUR DE PYTHON

UN GUIDE PRATIQUE POUR FAIRE PASSER VOS COMPÉTENCES PYTHON AU NIVEAU SUPÉRIEUR

TABLE DES MATIERES

I. INTRODUCTION

Selon le dernier index de la communauté de programmation ***TIOBE,*** Python est l'un des 10 langages de programmation les plus populaires de 2021. Classé en seconde position, Python a remporté, en mai 2021, le prix du langage de programmation qui a gagné le plus de popularité en un an. Ce titre lui est décerné pour la quatrième fois de l'histoire, ce qui est un record.

May 2021	May 2020	Charge	Programming Language	Ratings	Change
1	1		C	13,38%	-3,68%
2	3	▲	Python	11,87%	+2,75%
3	2	▼	Java	11,74%	-4,54%
4	4		C++	7,81%	+1,69%
5	5		C#	4,41%	+0,12%

(Source : https://www.tiobe.com/tiobe-index/)

Selon ***TIOBE***, Python apparait partout. À l'heure actuelle, c'est le langage de programmation privilégié dans les domaines de la science de données et de l'apprentissage automatique; mais, il est également utilisé pour la programmation backend et le développement Web et même dans les systèmes embarqués.

Il existe un certain nombre d'arguments qui penchent en faveur de Python. La facilité d'apprentissage, la richesse et la robustesse de ses bibliothèques sont parmi les qualités essentielles qui ont contribué à son adoption massive et à le placer en favori des langages de programmation.

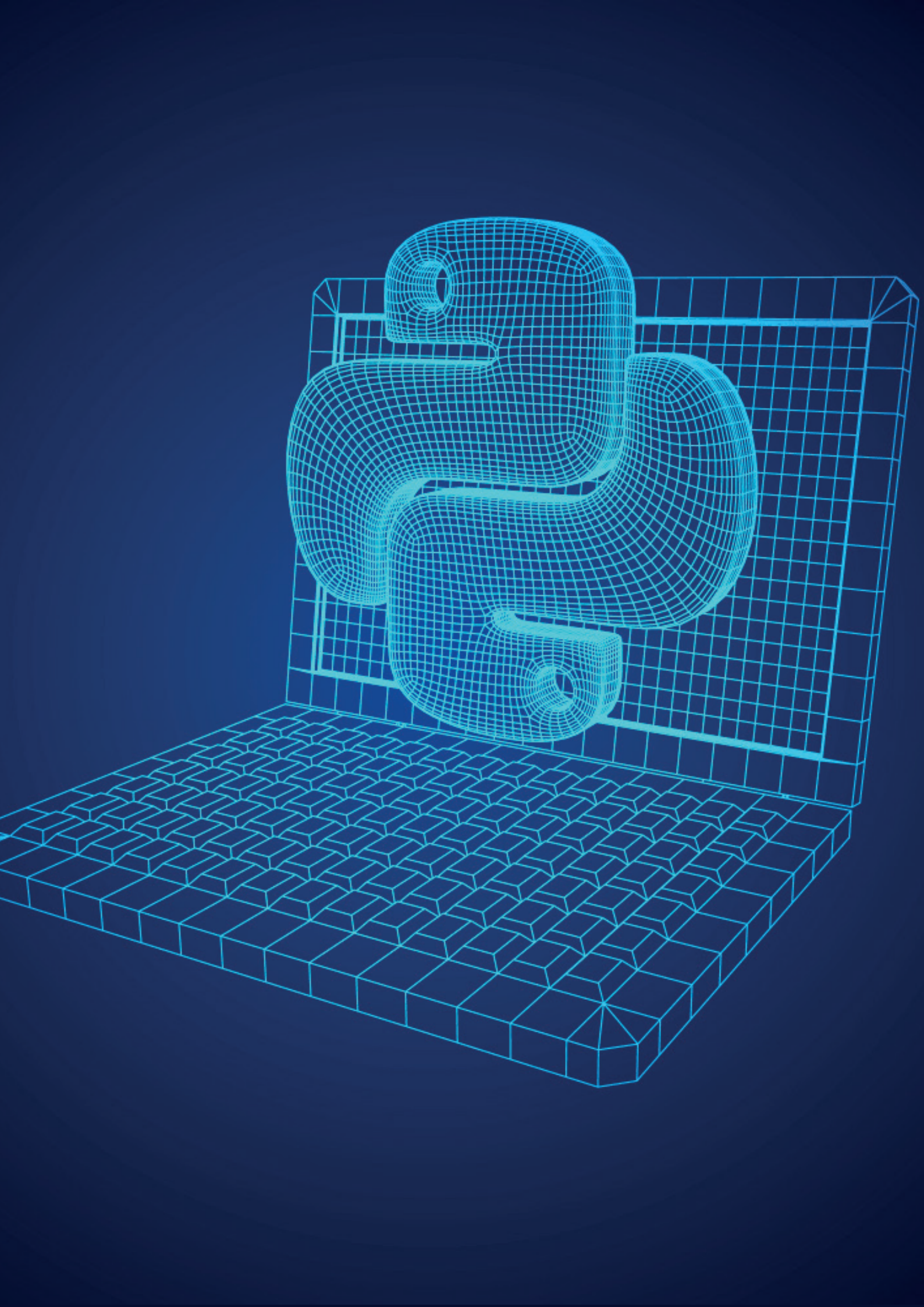
Outre cela, étant un langage sous licence libre, Python est compatible avec les principales plateformes et systèmes d'exploitation, notamment Windows, Unix, Mac OS, Android et iOS. Il offre également des outils de programmation très performants et simples à utiliser dans le but d'augmenter la productivité des développeurs utilisant ce langage.

Comme pour presque tous les langages de programmation, il existe un ensemble de directives et de règles stylistiques et conventionnelles acceptées par la communauté Python pour produire des applications unifiées, maintenables et concises.

Nous présentons, dans ce livre blanc, les meilleures pratiques et les conseils les plus précieux de Python afin d'améliorer la qualité de votre programme et de vous permettre de développer du code propre, lisible et « pythonique[1] ».

Que vous utilisiez Python depuis un certain temps ou que vous débutiez, nous allons vous faire bénéficier d'astuces qui vont vous permettre de faire passer votre code au niveau supérieur.

[1] **Pythonique** signifie un code qui n'obtient pas seulement la syntaxe correcte, mais qui suit les conventions de la communauté Python et utilise le langage de la manière dont il est destiné à être utilisé.



II. À QUOI RESSEMBLE PYTHON ?

Un peu d'histoire

En février 1991, Guido Van Rossum a publié la première édition du langage de programmation Python. Progressivement, la popularité de Python a évolué de façon constante. 30 ans après, le langage est absolument partout. Avec une base d'utilisateurs en croissance rapide et un large éventail de capacités, Python pourrait sembler destiné à devenir la **lingua franca**[2] de la programmation.

Python a été conçu pour être facile à apprendre, facile à comprendre et surtout facile à lire. La lisibilité et la réutilisation étant au cœur de la conception du langage, nous pourrons nous attendre à ce que la maîtrise du langage soit plus facile.

A. À quoi ressemble Python vu d'en haut ?

- **Un langage de programmation facile et lisible**

Comparé aux autres langages de programmation, Python est de loin le plus facile à étudier. Il se caractérise par une syntaxe claire, facile à lire et à comprendre. En milieu professionnel, ceci est considérablement important étant donné qu'un programme peut changer de main plusieurs fois et doit être maintenu sur le long terme.

- **Un langage polyvalent et à usage générique**

Python est un langage interactif, orienté objet et de haut niveau. Il a également la particularité d'être multiplateforme et interopérable. Il fonctionne sur toutes les plateformes connues : Windows, Linux, iOS y compris les plateformes mobiles telles que Maemo ou Android. Ceci rend Python un langage privilégié dans presque tous les domaines de l'Informatique, particulièrement le développement Web, le cloud computing, l'automatisation, les tests de logiciels, le Big Data, la science des données, l'intelligence artificielle, etc.

[2] Une lingua franca ou langue franque est une langue couramment utilisée pour les communications entre des personnes qui ne partagent pas la même langue maternelle.

- **Offre un débogueur intégré**

Python offre également un débogueur intégré pour aider les développeurs à analyser les bugs d'un programme. Il permet à n'importe quel programme Python d'être interrompu et exécuté pas-à-pas. Le débogueur facilite aussi l'affichage du contenu et du type des variables à tout moment de l'exécution grâce à la pose de point d'arrêt sur des lignes du programme.

- **Programmation avec moins de lignes de code**

Une autre particularité de Python est sa capacité à créer des fonctions et à résoudre des problèmes avec moins de lignes de code. Sa syntaxe est plus concise que les autres langages de programmation, ce qui facilite la mise en route et le test des programmes à la volée rapidement et facilement.

Exemple : Python vs Java

Étant un langage interactif, ce dernier donne la main au développeur d'exécuter à la demande toute commande Python valide. Dans ce mode interactif, Python peut aussi être utilisé comme une calculatrice programmable .Saisissez sur le clavier 2+4, l'ordinateur vous affiche 6.

Python — Possibilité 1

```
>>> 2 + 4
6
```

Python — Possibilité 2

```
>>> print(2 + 4)
6
```

En revanche, Java n'a pas d'interpréteur de commandes (CLI). Donc, pour imprimer 6 comme nous l'avons fait ci-dessus, nous devons écrire un programme complet puis le compiler avant de l'exécuter.

Java

```
public class Print6 {
        public static void main(String[] args) {
         System.out.println("2+4=" + (Integer.toString(2+4)));
        }
}
```

- **Un large catalogue de bibliothèques et de frameworks spécialisés**

Python dispose de l'un des gestionnaires de paquets les plus matures : PyPI (de l'anglais « Python Package Index »). PyPI est le dépôt officiel tiers de Python, doté de plus de 200 000 modules Python contenant des fonctions prêtes à l'emploi.

Son objectif est de doter la communauté des développeurs Python d'un large catalogue de paquets Python libres. Ces bibliothèques fournissent des fonctionnalités permettant de résoudre divers problèmes relatifs notamment à la création de services Web RESTful, à l'analyse des données, à l'apprentissage supervisé et autres.

- **Modernité et multiparadigme**

Au-delà d'une syntaxe très lisible, l'un des points positifs du langage est le gain en productivité pendant le développement par rapport à d'autres langages à typage statique comme Java ou C++.

Python possède un typage dynamique, une compilation automatique en bytecode, un garbage collector, une gestion des exceptions, de l'Unicode, de la programmation multithread et multiprocessus ainsi que d'autres caractéristiques qui en font un langage moderne et de plus en plus utilisé.

Python permet, entre autres, de programmer selon différents paradigmes, à savoir la programmation impérative et procédurale, la programmation orienté-objet et la programmation fonctionnelle grâce à l'utilisation des opérations de bases ***map***, ***filter*** et ***reduce*** et des fonctions anonymes ***lambda***.

B. Python en un mot

Python est un langage de programmation robuste qui fournit aux développeurs un environnement flexible et dynamique. Sa puissance et sa facilité d'utilisation l'ont propulsé à devenir l'un des langages de programmation le plus polyvalent et bientôt le plus utiliser. Au cours des dernières années, Python a réussi à développer une forte communauté aussi bien dans le développement traditionnel que l'Intelligence Artificielle.

Dans ce qui suit, nous allons voir comment exploiter le vaste potentiel de Python pour développer des programmes robustes et maintenables.


key, value
typeOfFID
if(typeOfFID
d =
dataCal
FidAndValue
else:FidAndValu

III. LES STRUCTURES DE DONNEES LES PLUS COURANTES DE PYTHON

Les structures de données sont les fondements autour desquelles les programmes sont construits. Chaque structure de données fournit une manière particulière d'organiser les données afin qu'elles soient accessibles efficacement, tant en termes d'espace que de temps.
Python est livré avec un ensemble complet de structures de données dans sa bibliothèque standard. Dans ce chapitre, nous allons parcourir les types de données integrés dans Python et apprendre comment les mettre en pratique dans divers algorithmes.

A. Les dictionnaires

Les dictionnaires ou ***dict³*** en abrégé sont l'une des structures de données les plus importantes et les plus utilisées en Python. Les dictionnaires sont aussi appelés **maps, hashmaps, tables de recherche ou tableaux associatifs.**

Un dictionnaire se compose d'une collection de paires clé-valeur. Chaque paire clé-valeur mappe la clé unique à sa valeur associée. Les dictionnaires stockent un nombre arbitraire d'objets et permettent la recherche, l'insertion et la suppression efficaces de tout objet associé à une clé donnée.

Pour illustrer son fonctionnement, nous allons assimiler le dictionnaire à un annuaire téléphonique. Imaginez que vous vouliez trouver le numéro d'une certaine personne. Au lieu d'avoir à lire tout l'annuaire recto verso, le dictionnaire vous permet d'accéder directement au nom de la personne et de récupérer les informations associées.
Python propose une implémentation de dictionnaire robuste qui est intégrée directement dans le langage de base: le type de données ***dict.***

dict³ : Le dictionnaire de référence

Python fournit un **sucre syntaxique**[4] utile pour travailler avec les objets dictionnaires.
Pour définir un nouveau dictionnaire, il est possible d'utiliser la syntaxe d'expression de dictionnaire avec les accolades bouclées ({}) ou les dictionnaires en compréhension.

Python

```
>>> dict1 = {'one': 1, 'two': 2, 'three': 3}
>>> dict1
{'one': 1, 'two': 2, 'three': 3}

>>> # Create a dict using dictionary comprehension
>>> dict2 = {x: x * x for x in range(6)}
>>> dict2
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25}

>>> # Another way to create a dict using dictionary comprehension
>>> keys = ['one', 'two', 'three']
>>> values = [1, 2, 3]
>>> dict3 = {key: value for key, value in zip(keys, keys)}
>>> dict3
{'one': 1, 'two': 2, 'three': 3}
```

Ici, ***zip***() reçoit en arguments deux itérables (keys et values) et crée des couples qui agrègent les éléments de chaque itérables. Ces couples sont ensuite décompressés en clé et valeur utilisés pour créer le nouveau dictionnaire.

Les dictionnaires Python sont basés sur une implémentation de table de hachage bien testée et finement réglée qui fournit les caractéristiques de performances attendues: O (1) complexité temporelle pour les opérations de recherche, d'insertion, de mise à jour et de suppression dans le cas moyen.

Outre les objets ***dict***, Python comprend un certain nombre d'implémentation de dictionnaires spécialisés. Ils sont tous basés sur la classe ***dict*** intégrée et partagent ses caractéristiques de performances, mais incluent d'autres fonctionnalités supplémentaires.

[4] Le **sucre syntaxique** exprime le fait de donner au programmeur des possibilités d'écriture plus succinctes ou plus proches d'une notation usuelle

▸ *collections.ChainMap: Fusion de dictionnaires*

Python comprend une sous-classe de ***dict*** appelée ***collections.ChainMap*** permettant de regrouper plusieurs dictionnaires en un seul mappage. Les recherches se font sur les mapping sous-jacents un par un jusqu'à ce qu'une clé soit trouvée.

> **Remarque :** ***collections.ChainMap*** ne fait pas partie intégrante du langage de base et doit être importé du module ***collections***.

Seul le premier dictionnaire ajouté à la chaine est affecté par les insertions, les mises à jour et les suppressions.

Python

```
>>> from collections import ChainMap
>>> dict1 = {"a": 1, "b": 2}
>>> dict2 = {"c": 3, "d": 4}
>>> chain = ChainMap(dict1, dict2)

>>> chain
ChainMap({'a': 1, 'b': 2}, {'c': 3, 'd': 4})

>>> # ChainMap searches each collection in the chain
>>> # from left to right until it finds the key (or fails):
>>> chain["c"]
3
>>> chain["a"]
1
>>> chain["f"]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'f'
```

▸ *types.MappingProxyType: Dictionnaire en lecture seule*

MappingProxyType fournit un accès au dictionnaire en mode lecture seule. Cette classe peut être utile si vous souhaitez mettre en place des restrictions limitant la manipulation du dictionnaire ; c'est-à-dire que si vous souhaitez passer un dictionnaire à une autre fonction mais sans qu'elle ne puisse le changer, ***MappingProxyType*** agit alors comme un proxy en lecture seule.

En revanche, si le dictionnaire initial change, le proxy reflète aussi ces modifications.

Python
```
>>> from types import MappingProxyType
>>> d = {'a': 1, 'b': 2}
>>> m = MappingProxyType(d)
>>> m['a']
1
>>> m['a'] = 5
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'mappingproxy' object does not support item assignment
>>> d['a'] = 42
>>> m['a']
42
>>> for i in m.items():
...     print(i)
('a', 42)
('b', 2)
```

B. Les tableaux

Un tableau est une structure de données fondamentales disponible dans la plupart des langages de programmation, et il a un large éventail d'utilisations à travers différents algorithmes.
Les tableaux se composent d'enregistrements de données de taille fixe qui permettent à chaque élément d'être localisé efficacement en fonction de son index.

Étant donné que les tableaux stockent des informations dans des blocs de mémoire adjacents, ils sont considérés comme des structures de données contiguës (par opposition à une structure de données liée comme une liste liée, par exemple).

Dans cette section, nous allons uniquement étudier une sélection des implémentations de tableaux les plus utilisées et incluses dans la bibliothèque standard de Python.

▸ *list : Tableaux dynamiques modifiables*

Les ***listes*** sont un type de données utile en Python. Malgré leur nom, les listes de Python sont implémentées, dans les coulisses, sous forme de tableaux dynamiques modifiables. Cela signifie qu'il est possible d'ajouter ou de supprimer des éléments de la liste.

Les ***listes*** peuvent contenir des éléments arbitraires : un entier, une chaîne de caractères, une autre liste, une fonction Python et bien plus encore. Cela peut être une fonctionnalité puissante, mais l'inconvénient est que la prise en charge de plusieurs types de données en même temps signifie que les données sont généralement moins compressées. En conséquence, toute la structure prend plus de place.

Python
```
>>> list1 = ["one", 2, "three", 4]
>>> list1 [0]
'one'

>>> list1 [1] = "two"
>>> list1
["one", "two", "three", 4]

>>> del list1 [1]
>>> list1
['one', 'three', 4]

>>> list1.append([5, 6])
>>> list1
['one', 'three', 4, [5, 6]]
```

▸ *tuple : Conteneurs immuables*

Tout comme les ***listes***, les ***tuples*** (ou en français n-uplets) font partis du langage de base de Python. Cependant, les tuples sont immuables[5] ; c'est-à-dire que les éléments ne peuvent être ni ajoutés ni supprimés dynamiquement - tous les éléments d'un tuple doivent être définis au moment de la création.
Les tuples ont la même flexibilité que les listes et peuvent contenir tout type de données.

Python
```
>>> tuple1 = ("one", 2, "three")
>>> tuple1[0]
'one'

>>> # Tuples are immutable:
>>> tuple1[1] = "hello"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment

>>> del tuple1[1]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object doesn't support item deletion
```

[5]Qu'on ne peut pas modifier

▸ *str : Tableau immuable de chaine de caractères*

La structure de données ***str*** permet de stocker des données textuelles sous forme de séquences immuables de caractères ***Unicode***.

Contrairement aux ***listes*** et ***tuples***, les objets ***str*** sont peut encombrants vu qu'ils sont très compactes et ne comportent qu'un seul type de données.

Étant donné que les chaînes sont immuables en Python, la modification d'une chaîne nécessite la création d'une copie modifiée.

Python

```
>>> str1 = "hello"
>>> str1[1]
'e'

>>> # Strings are immutable:
>>> str1[1] = "a"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment

>>> del str1[1]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object doesn't support item deletion
```

C. Les ensembles

Une autre structure de données fondamentale est l'***ensemble*** (ou en anglais ***set***). L'ensemble est une collection d'objets non ordonnés et uniques ; c'est-à-dire que contrairement aux autres types de données tels que les ***listes*** et les ***tuples***, les ***sets*** n'autorisent pas les duplications d'éléments.

D'une manière générale, les ***sets*** sont utilisés pour insérer ou supprimer de nouvelles valeurs d'un ensemble et pour calculer l'union ou l'intersection de deux ensembles.

Tout comme les ***dictionnaires***, les ***sets*** reçoivent un traitement spécial en Python et contiennent du sucre syntaxique qui les rend faciles à créer. La syntaxe d'accolades bouclées ({}) ou les ensembles en compréhension permettent de définir facilement de nouvelles instances de ***set***.

Python
```
>>> vowels = {"a", "e", "i", "o", "u"}
>>> "e" in vowels
True

>>> letters = set("hello")
>>> letters.intersection(vowels)
{'e', 'o'}

>>> vowels.add("y")
>>> vowels
{'i', 'a', 'u', 'o', 'y', 'e'}

>>> len(vowels)
6
```

Astuce : Si vous souhaitez **supprimer les doublons d'une *liste***, utilisez le type de données ***set***. Il suffit de convertir la liste en ensemble. Les éléments en double seront automatiquement supprimés. Vous devez ensuite convertir l'ensemble résultant en ***liste*** pour retrouver le type de données initial. Tout ceci peut se faire en une ligne de code.

Python
```
>>> list1 = ["hello", "alice", "car", "alice", "ben", "hello"]
>>> list1
["hello", "alice", "car", "alice", "ben", "hello"]

>>> # Remove duplicates
>>> list1 = list(set(list1))
>>> list1
["ben", "car", "alice", "hello"]
```

D. Les structures de données en un mot

Python comporte une liste importante de type de données et il est livré avec plusieurs implémentations pour chaque structure. Tous ont des caractéristiques légèrement différentes ainsi que des compromis en termes de performances et d'utilisation. Il est donc important de bien choisir les structures de données qui conviennent parfaitement à votre algorithme ou à votre cas d'utilisation spécifique.

Nous avons ici présenté quelques structures de données de base les plus utilisées en programmation Python. Nous vous invitons à vous référer à la documentation officielle de Python pour découvrir les implémentations et les différentes variantes de ces types de données.


check_catch(self):
""" Check if catch balls.
for ball in self.overlappi
self.score.value += 10
self.score.right = games
ball.handle_caught()
""" Change game level.
if self.score.value ==
self.level.value +=
self.level.left =
""" Next level
level_message

IV. COMMENT UTILISER LES FONCTIONS ANONYMES EFFICACEMENT ?

Les fonctions de Python sont des objets de première classe[6] . Vous pouvez les affecter à des variables, les stocker dans des structures de données, les transmettre en tant qu'arguments à d'autres fonctions et même les renvoyer en tant que valeurs d'une autre fonction.
Comme tout langage de programmation, Python est livré avec des fonctions anonymes soumises à une syntaxe plus restrictive mais plus concise que les fonctions Python classiques.

A. La fonction lambda

Il existe des fonctions dans Python qu'on appelle fonction anonyme ou ***lambda***. Ces fonctions sont des méthodes sans nom, c'est-à-dire non liées à un identifiant comme lorsque l'on définit une méthode en utilisant ***def function***.

En pratique, les méthodes lambda fonctionnent de la même façon que tout autre méthode en Python, à un détail près : elles sont définies sur une ligne de code sans nom.

```
lambda arguments: expression
```

Ci-dessous la définition d'une fonction Python :

Python
```
>>> def add(x,y):
...     return x + y
>>> add(2, 4)
6
```

Elle peut être traduite en :

Python
```
>>> add = lambda x, y: x + y
>>> add(2, 4)
6
```

[6] En informatique, un objet de première classe (ou en anglais first-class object) est toute entité qui peut être utilisée sans restriction

Les ***lambdas*** différent des méthodes Python normales car elles ne peuvent avoir qu'une seule expression, ne peuvent contenir aucune instruction et leur type de retour est un object ***function***. Ainsi, la ligne de code `lambda x, y: x + y` ne renvoie pas exactement la valeur x + y mais la fonction qui calcule x + y.

Les expressions ***lambda*** sont souvent utilisées pour optimiser le code et résoudre des problèmes complexes avec le moins de lignes de code possible.

Par exemple, pour trier une liste de dictionnaires par valeur, vous pouvez soit développer l'algorithme de tri par sélection à la main en utilisant les boucles ***for*** comme suit:

Python

```
>>> list1 = [{"Name": "Jeremy", "Age": 25, "Color": "Blue"},
...          {"Name": "Ally", "Age": 41, "Color": "Magenta"},
...          {"Name": "Jasmine", "Age": 29, "Color": "Aqua"}]
>>>
>>> # Sort by age
>>> size = len(list1)
>>> for i in range(size):
...     min_index = i
...     for j in range(i + 1, size):
...         if list1[min_index]["Age"] > list1[j]["Age"]:
...             min_index = j
...     temp = list1[i]
...     list1[i] = list1[min_index]
...     list1[min_index] = temp
...
>>> list1
[{'Name': 'Jeremy', 'Age': 25, 'Color': 'Blue'}, {'Name': 'Jasmine', 'Age':
29, 'Color': 'Aqua'}, {'Name': 'Ally', 'Age': 41, 'Color': 'Magenta'}]
```

Vous pouvez aussi utiliser l'expression lambda. Ceci se traduit comme suit :

Python

```
>>> # Sort by Age
>>> list1 = sorted(list1, key=lambda item: item["Age"])
>>> list1
[{'Name': 'Jeremy', 'Age': 25, 'Color': 'Blue'}, {'Name': 'Jasmine', 'Age':
29, 'Color': 'Aqua'}, {'Name': 'Ally', 'Age': 41, 'Color': 'Magenta'}]
```

B. Les fonctions map, filter et reduce

Les fonctions ***map()***, ***filter()***et ***reduce()*** sont des paradigmes de programmation fonctionnelle. Ces trois fonctions sont des fonctions pratiques qui peuvent être remplacées par des listes en compréhension ou des boucles ***for***, mais qui offrent une approche plus élégante et plus courte.

▸ *La fonction map()*

La fonction ***map()*** parcourt tous les éléments d'une liste donnée et exécute la fonction ***function*** passée en arguments. Comparé à la boucle ***for***, la fonction ***map()*** a un temps d'exécution beaucoup plus rapide.

Sa syntaxe est la suivante :

```
map(function, iterable(s))
```

Vous pouvez passer en argument autant d'objets itérables que vous le souhaitez après avoir définit la fonction ***function***.

Dans ce qui suit, nous allons comparer le temps d'exécution de la fonction ***map()***, de la boucle ***for*** et des listes en compréhension.

Python

```
>>> # import datetime to compute execution time
>>>  from datetime import datetime

>>> # function to implement the Fibonacci Sequence
>>> def fib(n):
...     if n < 2:
...         return n
...     return fib(n-2) + fib(n-1)

>>> list1 = [1, 2 , 3, 4, 5, 6, 7, 8, 9, 10]

>>> # using 'map' to call the function 'fib' for all the elements of 'list1'
>>> start = datetime.now()
>>> list2 = map(fib, list1)
>>> end = datetime.now()
>>> print(end - start)
0:00:00.067818

>>> # map function returns a map object
>>> print(list2)
<map object at 0x0000022825E7E6C8>

>>> # convert map to list
>>> print(list(list2))
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55]

>>> # alternative way to execucte fib using 'comprehensive'
>>> start = datetime.now()
>>> list3 = [fib(i) for i in list1]
>>> end = datetime.now()
>>> print(end - start)
0:00:00.113697
>>> print(list3)
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55]

>>> # alternative way to execucte fib using 'for loop'
>>> start = datetime.now()
>>> list4 = []
>>>
>>> for i in list1 :
...     list4.append(fib(i))
>>> end = datetime.now()
>>> print(end - start)
0:00:00.126665
```

Comme vous pouvez le remarquer la fonction ***map()*** génère un objet de type ***map***. Il faudrait donc le convertir en une ***liste*** pour pouvoir l'utiliser.

Ici, les temps d'exécution sont assez proches mais ceci n'est pas le cas si le nombre d'éléments d'une liste devient assez important.

Python

```
>>> list1 = range(0, 45)

>>> # using 'map' to call the function 'fib' for all the elements of 'list1'
>>> start = datetime.now()
>>> list2 = map(fib, list1)
>>> end = datetime.now()
>>> print(end - start)
0:24:00.655769

>>> # alternative way to execucte fib using 'comprehensive'
>>> start = datetime.now()
>>> list3 = [fib(i) for i in list1]
>>> end = datetime.now()
>>> print(end - start)
0:24:31.179224

>>> # alternative way to execucte fib using 'for loop'
>>> start = datetime.now()
>>> list4 = []
>>>
>>> for i in list1 :
...     list4.append(fib(i))
>>> end = datetime.now()
>>> print(end - start)
0:55:51.138050
```

Ici ***range*** va générer une liste d'entiers allant de 0 à 44.

Comme nous pouvons le constater, la fonction ***map()*** est nettement plus rapide que les boucles ***for*** et les listes en compréhension.

▸ *La fonction filter()*

Similaire à ***map()***, la fonction ***filter()*** prend en argument une fonction et un ou plusieurs objets itérables.

Comme son nom l'indique, la fonction ***filtre()*** retourne une nouvelle liste qui ne contient que les éléments qui satisfont une certaine condition définie dans la fonction. Cela signifie que la fonction ***filtre()*** n'accepte en arguments que les fonctions qui renvoie des valeurs booléennes (vrai ou faux), puis passe chaque élément de l'itérable à travers la fonction et filtre ceux qui sont faux.

Elle a la syntaxe suivante :

```
filter(function, iterable(s))
```

Nous allons utiliser l'exemple précédent pour illustrer le fonctionnement de ***filter()***. La nouvelle liste ne contiendra que les éléments pairs de la liste.

Python
```
>>> def is_even(n):
...     return n % 2 == 0
...
>>> list1 = [1, 2 , 3, 4, 5, 6, 7, 8, 9, 10]
>>> filter_object = filter(is_even, list1)
>>> print(list(filter_object))
[2, 4, 6, 8, 10]
```

Nous pouvons également réécrire ce code en utilisant l'expression ***lambda*** :

Python
```
>>> list1 = [1, 2 , 3, 4, 5, 6, 7, 8, 9, 10]
>>> filter_object = filter(lambda n: n % 2 == 0, list1)
>>> print(list(filter_object))
[2, 4, 6, 8, 10]
```

▸ *La fonction reduce()*

Contrairement à ***map()*** et ***filter()***, la fonction ***reduce()*** ne renvoie pas une liste mais plutôt une valeur unique.

reduce() applique une fonction de **deux arguments de manière cumulative** aux éléments d'un itérable, en commençant éventuellement par un argument initial. Il a la syntaxe suivante:

```
reduce(function, iterable[, initial])
```

L'argument initial est optionnel. S'il est défini, il sera utilisé au début de l'exécution avec le premier élément de l'objet itérable.

Remarque : Dans Python 3 ***reduce()*** n'est plus une fonction intégrée, et il faut l'importer du module ***functools***.

Dans cette partie, nous garderons l'exemple précédent pour illustrer le fonctionnement de ***reduce()***.

Python
```
>>> from functools import reduce
>>> def add(x, y):
...     return x + y
...
>>> list1 = [1, 2 , 3, 4, 5, 6, 7, 8, 9, 10]
>>> print(reduce(add, list1))
55
```

Nous pouvons également réécrire ce code en utilisant l'expression ***lambda*** :

Python
```
>>> from functools import reduce
>>>
>>> list1 = [1, 2 , 3, 4, 5, 6, 7, 8, 9, 10]
>>> print(reduce(lambda x, y: x + y, list1))
55
```

C. Les fonctions anonymes en un mot

Les fonctions anonymes fournissent parfois de la valeur au langage Python et aux développeurs. Les exemples précédents illustrent des scénarios dans lesquels l'utilisation des fonctions lambda est non seulement appropriée mais encouragée dans le code Python. Elles apportent une forme d'élégance et de légèreté au code et optimise efficacement la consommation des ressources.

Les fonctions anonymes peuvent se placer partout, là où une fonction est attendue et nous pouvons ne pas l'affecter à une variable. C'est là où réside la simplicité des fonctions anonymes et de Python.


python

V. LES 10 BONNES PRATIQUES POUR UN MEILLEUR CODE

La lisibilité et la syntaxe simpliste sont au cœur de Python. Comme pour presque tous les langages de programmation, il existe certaines directives stylistiques et conventionnelles acceptées par la communauté Python pour promouvoir des applications maintenables et concises.
Ici, nous aborderons quelques règles de style qui vont vous permettre de créer un code unifié et lisible.

A. Les bonnes pratiques de Python

• Règle 1 : Suivre les directives de style

Le PEP (Python Enhancement Proposals) définit des directives et des normes de développement communes entre les développeurs Python.
Bien que Python soit un langage flexible, le PEP permet de s'assurer que tout le code a la même apparence et cohérence.

Parmi les directives proposées, nous citons :

• Utiliser les conventions de dénomination appropriées pour les variables, les fonctions, les méthodes, etc : ***this_is_a_variable***
• Nommer les classes et les exceptions avec la convention ***CapitalWords***
• Méthodes protégées et fonctions internes: ***_single_leading_underscore***
• Méthodes privées: ***__double_leading_underscore***
• Constantes: ***CAPS_WITH_UNDERSCORES***
• Utiliser 4 espaces pour l'indentation

• Règle 2 : Implémenter le contrôle de version

Lorsque vous démarrez un nouveau projet, il est recommandé de créer un nouveau répertoire et d'implémenter le contrôle de gestion. Vous pouvez soit utiliser GitHub ou un autre fournisseur.
Pour structurer votre projet Python, il est important d'inclure les éléments suivants :

• Licence, sous le répertoire racine (exemples : Licence publique Mozilla 2.0, Licence Apache 2.0, Licence MIT, etc)
• Un fichier README, également sous le fichier racine : il doit contenir une description du projet et de ce qu'il fait.
• Le code du module, qui peut être soit sous le répertoire racine ou à l'intérieur d'un sous-répertoire
• Le fichier requirements.txt, sous le répertoire racine. Ce fichier contient les modules et les dépendances du projet
• Une documentation

- **Règle 3 : Créer une documentation lisible**

La documentation lisible est l'une des meilleures pratiques Python. Ceci peut être fastidieux à rédiger, mais cela permet de créer un code propre, lisible et maintenable.
Vous pouvez utiliser un des outils de la liste suivante :

• ***Markdown*** et ***reStructuredText*** sont des langages de balisage avec une syntaxe de mise en forme de texte brut pour faciliter le balisage du texte et le convertir dans un format tel que HTML ou PDF.
• ***Sphinx*** est un outil pour créer facilement une documentation intelligente et belle. Il vous permet de générer des documentations Python à partir de ***reStructuredText*** existant et d'exporter la documentation dans des formats tels que HTML.

• Les ***docstrings*** sont des chaînes de documentation à mettre au début de chaque module, classe ou méthode (fortement recommandé).

- **Règle 4 : Isoler les dépendances des projets**

L'objectif principal des environnements virtuels Python est de créer un environnement isolé pour les projets Python . Cela signifie que chaque projet peut avoir ses propres dépendances, quelles que soient les dépendances de tous les autres projets.

Pour cela, il faudrait simplement créer, à l'aide de l'outil ***virtualenv***, un environnement virtuel séparé pour chacun des projets. Il n'y a aucune limite au nombre d'environnements. Cela signifie que vous pouvez avoir autant d'environnements virtuels que de projets (ou plus).

- **Règle 5 : Corriger immédiatement les bugs**

Il est impératif de corriger immédiatement les erreurs. Si vous laissez les bugs traîner, cela risque d'entrainer des problèmes plus graves plus tard.

Optez plutôt pour une méthodologie « zéro défaut » : limiter les erreurs et les retours client, réduire les opérations de retouches. L'effort de correction peut être particulièrement conséquent.

- **Règle 6 : Gérer les exceptions**

Les exceptions sont des erreurs que Python signale en cas d'erreurs syntaxiques ou lorsqu'une situation exceptionnelle se produit, par exemple diviser un entier par zéro. Si une erreur se produit, Python la signale et met fin au programme. Ce n'est généralement pas ce que vous voulez.

Pour éviter cela, vous pouvez gérer les exceptions en plaçant vos instructions entre les expressions ***try-except*** ou ***assert***.

Il est recommandé de spécifier le type de chaque exception et d'éviter les déclarations génériques.

Python

```
# Not recommended
try:
        <instructions>
except:
        <solutions>

# Recommended
try:
  z = x / y
except NameError:
  print("Variable x is not defined")
except ZeroDivisionError:
  print("Something else went wrong")
```

- **Règle 7 : Déboguer**

Le débogage du code est très important pour anticiper et résoudre les erreurs. Il permet de signaler immédiatement les erreurs et d'éviter les codes de test à usage unique.

Le débogage permet aussi de fournir des informations sur le contenu et la structure des données pour faciliter l'interprétation du code. Grâce au débogage, les développeurs peuvent gagner du temps et de l'énergie. Cela rend le développement moins stressant et plus efficace.

- **Règle 8 : Utiliser PyPI**

L'une des raisons de la popularité de Python est la richesse de ces bibliothèques. Le PyPI vous permet de bénéficier des librairies et frameworks partagées par la communauté de développeurs Python.

Vous pouvez les installer à l'aide de ***pip*** au lieu de les écrire vous-même. Cela vous fait gagner du temps, vous demande moins d'efforts et vous permet de vous concentrer sur des tâches beaucoup plus importantes.

- **Règle 9 : Privilégier une conception orientée objet**

Python est un langage orienté objet. Tout tourne autour des objets. Il est vivement recommandé d'utiliser le paradigme orienté objet. Cela permet la réutilisabilité, la modularité, le polymorphisme, l'encapsulation des données et l'héritage.

Il est aussi recommandé d'utiliser les classes abstraites. Cela peut apporter de la clarté au code. Une autre raison pour commencer à rédiger des classes abstraites est si plusieurs personnes travaillent sur le même projet et que tout le monde commence à faire des méthodes différentes, cela peut créer un chaos improductif.

- **Règle 10 : Utiliser les patrons de conception (Design pattern)**

En Python, rien ne vous oblige à écrire des classes ou à instancier des objets. Vous pouvez tout aussi bien écrire des fonctions ou mieux encore, vous pouvez écrire un code plat sans structurer le code.

Toutefois, bien qu'il soit si dynamique et flexible, Python est un langage orienté objet par excellence. De ce fait, nous avons besoin de certaines règles pour la conception de notre programme. Les patrons de conception (souvent appelés design pattern) sont un excellent moyen d'exploiter son vaste potentiel. Vous pouvez utiliser les patrons de création, structurels et comportementaux.

B. Les bonnes pratiques de python en un mot

Le poème écrit par Tim Peters résume toutes les bonnes pratiques à suivre pour une meilleure efficacité de programmation. Vous pouvez l'obtenir en exécutant la commande `import this` dans l'interpréteur de commande de Python.

Python
```
>>> 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!
```


SYSTEM
SAFETY
DATA BREACH
CYBER ATTACK
SYSTEM PROTECTION
SECURUTY BREACH

VI. PYTHON DANS LA SCIENCE DES DONNÉES ET L'INTELLIGENCE ARTIFICIELLE

Python, à côté du langage R, est le langage le plus utilisé dans la science moderne des données et l'Intelligence Artificielle. Cependant, les projets d'IA diffèrent d'un projet logiciel traditionnel, il est donc nécessaire d'approfondir le sujet.

A. Pourquoi utiliser python ?

Plusieurs raisons ont fait que Python soit le langage fondamental de l'Intelligence Artificielle.

- **Une riche sélection de bibliothèques**

Une sélection extraordinaire de bibliothèques et de frameworks est l'une des principales raisons pour lesquelles Python est le plus utilisé en IA. Les bibliothèques Python fournissent des éléments de base, de sorte que les concepteurs n'ont pas nécessairement besoin de les coder de zéro.

Voici quelques-unes des bibliothèques les plus répandues que vous pouvez utiliser dans l'IA:
- ***SciPy*** - informatique avancée
- ***Keras***, ***PyTorch***, ***Tensorflow*** - modèles d'apprentissage automatique et d'apprentissage en profondeur
- ***Scikit-learn*** - Modélisation de données
- ***NumPy*** - nettoyage et manipulation des données
- ***Seaborn, Matplotlib*** - visualisation de données
- ***Caffe***, ***OpenCV*** - traitement d'images
- ***Pandas*** - utilisation générale pour l'analyse des données

Dans le Dépôt PyPI, vous pouvez découvrir et comparer plus de bibliothèques Python.

- **Une facilité d'apprentissage et flexibilité**

La faible barrière d'entrée permet à davantage de scientifiques des données d'apprendre rapidement Python et de commencer à l'utiliser pour le développement de l'IA sans perdre trop d'efforts pour apprendre le langage.

De plus, la flexibilité de Python permet aux développeurs de choisir les styles de programmation avec lesquels ils sont parfaitement à l'aise ou même de les combiner pour résoudre différents types de problèmes de la manière la plus efficace.

Les développeurs peuvent tout aussi bien utiliser le style fonctionnel, procédural ou orienté objet, utiliser des éditeurs de code ou bien des notebooks pour exécuter séparément des bouts de code.

- **Indépendance de la plateforme**

Comme nous l'avons déjà expliqué, Python est non seulement facile à utiliser mais aussi très polyvalent. Cela signifie que les frameworks et les bibliothèques existants ainsi que les modules d'apprentissage développés peuvent être exécutés sur la plupart des systèmes d'exploitation sans même avoir besoin d'un interpréteur Python.

- **Bonnes options de visualisation**

Il est important de souligner qu'en matière de science des données et d'Intelligence Artificielle, il est essentiel de pouvoir représenter les données dans un format lisible par l'Homme.

Python fournit un large catalogue de bibliothèques tels que ***Matplotlib*** et ***Seaborn*** permettant aux scientifiques des données de créer des graphiques et des histogrammes pour une meilleure compréhension et une visualisation efficace des données.
Différentes interfaces de programmation tels que ***Jupyter notebook*** et ***Google Colab*** simplifient également le processus de visualisation et facilitent la création de rapports clairs.

- **Soutien de la communauté**

Python bénéficie d'un fort soutien communautaire. Python est un langage libre source, ce qui signifie qu'il existe un large éventail de ressources disponibles pour les développeurs, qu'ils soient débutants ou experts.

Une grande partie de la documentation Python est disponible en ligne ainsi que dans les forums de collaboration, où les développeurs discutent des erreurs et proposent les solutions adéquates.

B. Les pratiques d'apprentissage automatique utiles pour les développeurs python

Comme pour la programmation traditionnelle, il existe quelques autres bonnes pratiques pour vous aider à programmer efficacement.

Ajoutées aux bonnes pratiques du chapitre précédent, nous vous proposons quelques règles utiles pour la manipulation des données et le développement des solutions intelligentes.

- **Règle 1 : Chronométrer les fonctions**

Toutes les fonctions ne sont pas créées égales.

Même si tout le code fonctionne, cela ne signifie pas que vous avez écrit du bon code. Certains bogues logiciels peuvent ralentir l'exécution du programme et il est nécessaire de les trouver. Vous pouvez utiliser les décorateurs pour calculer le temps d'exécution des fonctions.

- **Règle 2 : Accélérer le traitement de pandas**

Dans certains cas, la bibliothèque ***pandas*** peut s'avérer être très lente, en particulier quand vous gérez de grands ensembles de données. Vous pouvez donc utiliser ***modin*** comme solution pour l'accélérer.

```
import modin.pandas as pd
```

Cette bibliothèque Python peut être utilisée pour remplacer ***pandas***, en particulier lors du traitement d'énormes ensembles de données. ***modin*** est capable d'**accélérer vos programmes *pandas* jusqu'à 4x.**

- **Règle 3 : Définir les options d'affichage des DataFrame**

Il s'agit de changer la façon dont pandas affiche les ***DataFrame*** dans ***Jupyter notebook***. Il est important de placer ces lignes de code avant l'instruction d'importation des données.

Python
```
pd.set_option('max_colwidth', 1000) # Show up to 1000 characters within each
cell
pd.set_option('max_rows', 20) # Show up to 20 dataframe rows
pd.set_option('max_columns', 1000) # Show up to 1000 columns
```

Ainsi, vous n'avez plus à vous soucier de la taille de votre DataFrame. Vous pouvez défiler vers la droite et la gauche pour lire entièrement le contenu des cellules.

- **Règle 4 : Créer des APIs robustes**

Vous pouvez développer le plus innovant des modèles d'apprentissage supervisé, vous pouvez quand même créer un énorme chaos au moment du déploiement. Il est donc important de bien choisir votre framework pour la création d'API Python.

Vous pouvez opter pour ***Fastapi*** pour sa rapidité et sa haute performance.

C. Python au service de l'IA en un mot

La puissance et la facilité d'utilisation de Python l'ont aidé à devenir l'un des langages de base pour fournir des solutions d'Intelligence Artificielle. C'est l'outil fondamental pour servir le domaine multidisciplinaire de la science des données. Il permet aux développeurs de manipuler les données provenant de sources diverses et variées, de les transformer en application intelligente et robuste et de les déployer sur n'importe quelle infrastructure. Sa flexibilité, son large choix de bibliothèques et son soutien communautaire ont fait de lui le langage de prédilection des experts en IA et en science de données.

VII. CONCLUSION

Cela conclut votre visite du langage de programmation Python. Les 30 ans d'âge de la plateforme lui ont apporté une forte maturité et ont engendré un environnement et un écosystème très varié.

La notoriété de Python a été acquise grâce à son agilité, sa polyvalence et son efficacité. Peu de langages de programmation offrent une telle simplicité et flexibilité.

Pour garantir une meilleure performance, il est recommandé de suivre les directives conventionnelles proposées par la communauté de Python. Dans ce livre blanc, nous avons présenté quelques règles à suivre pour faire passer votre code au niveau supérieur et optimiser votre programme. Nous vous invitons à exploiter le vaste potentiel de Python pour développer du code « Pythonique » et propre.

VIII. A PROPOS DE WEVIOO

Crée en 1998, Wevioo est un groupe international de conseil et de services numériques. Wevioo accompagne ses clients dans leurs projets de transformation digitale. Customer-oriented, le groupe apporte son expertise et son savoir-faire dans 3 domaines : le Consulting, le Digital et les Solutions Embarquées.

En partenaire engagé Wevioo apporte à ses clients des solutions d'innovation digitale parfaitement adaptées à leurs enjeux d'agilité, de performance et de développement à l'international.

Avec une culture de l'innovation au cœur de son ADN, Wevioo investit dans la R&D pour apporter à ses clients et partenaires des expertises à la pointe des technologies les plus récentes et des solutions les plus innovantes.

Avec **23 années** d'innovation technologique au service de ses clients, Wevioo emploie plus de 350 talents et experts métier. Présent à Paris, Dubaï, Tunis, Alger et Abidjan, Wevioo est fort de plusieurs centaines de projets exigeants menés dans plus de 30 pays d'Europe, d'Amérique du Nord, d'Afrique et du Moyen-Orient.

Pour en savoir plus : **www.wevioo.com**

IX. AUTEURS DU LIVRE BLANC #6

Khaled Ben Driss

Chief Technology Officer

Lilia Zghal

Ingénieur IA