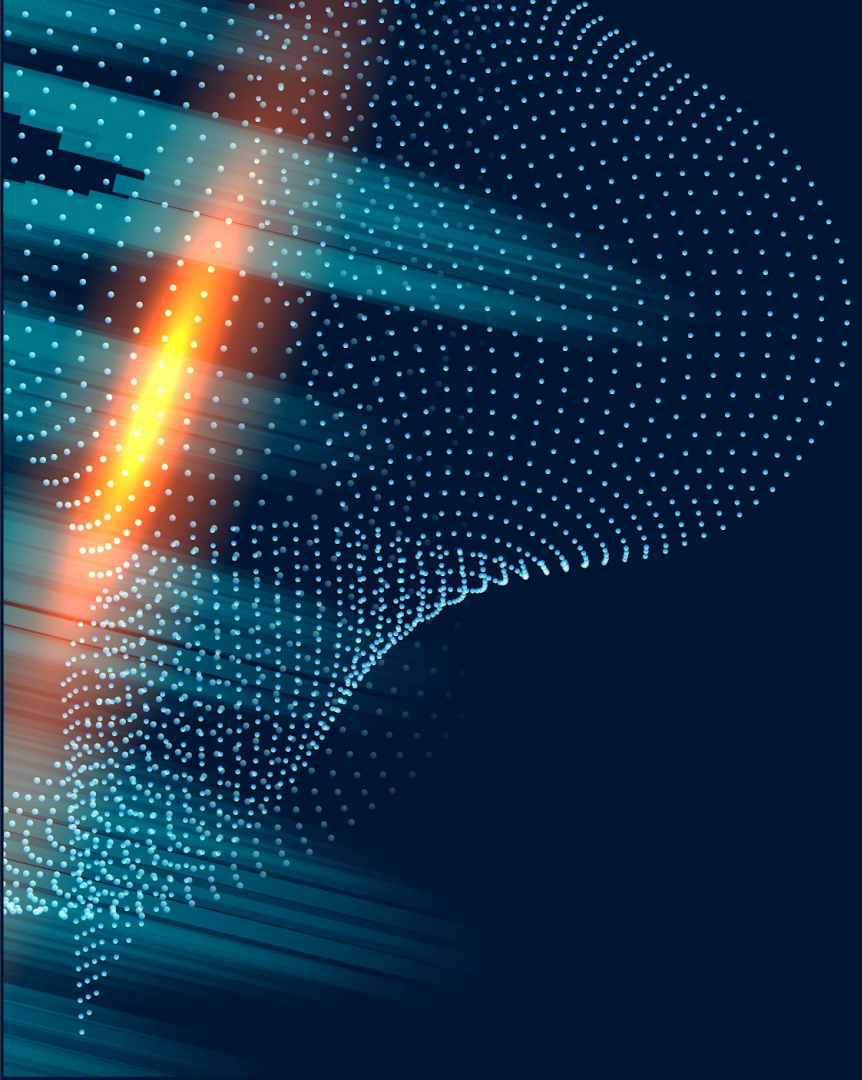
PYTHON Workshop

Advanced concepts II

“Ce qui ne se partage pas, se perd”



Hello!

I’m Khouas Aymen Rayane, Current President of OpenMindsClub, and Artificial Intelligence (SII) student.
Contact : aymenkhouas@gmail.com

01

Iterators

02

Decorators

03

Files

04

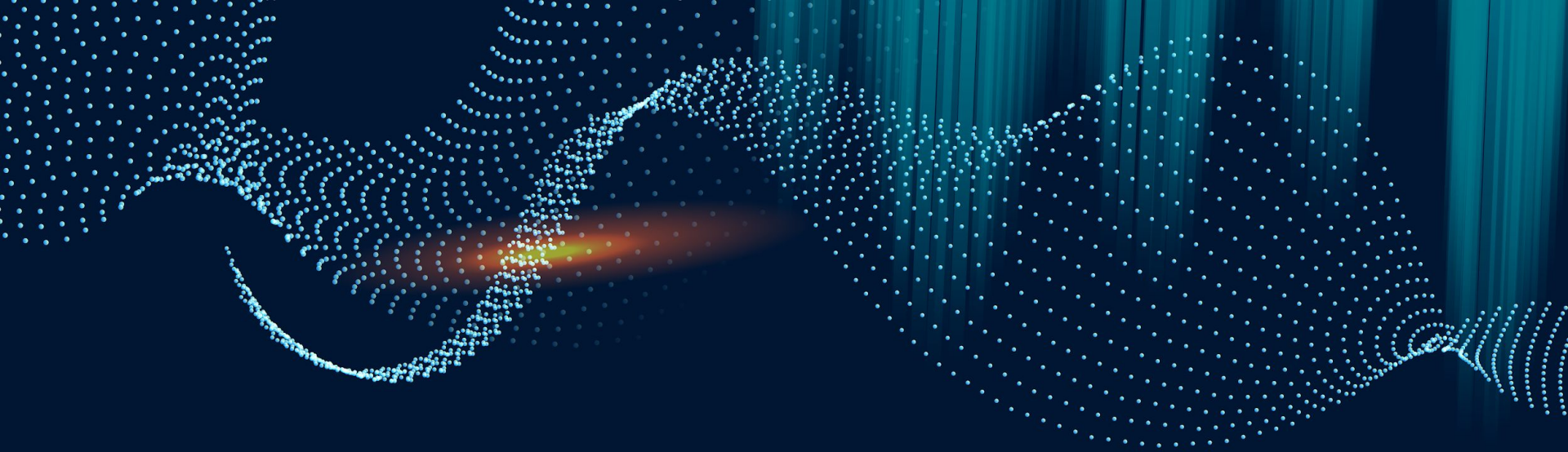
References

05

Use PIP

06

Write good code in python



01 Iterators

You want to control the world?
First, build your own iterator

Itérateurs

Méthode iter()

Initialiser l’itérateur

Méthode next()

Passé à la valeur suivante

Exemple

```
>>> string = "hello world"
>>> my_iterator = iter(string)
>>> next(my_iterator)
'h'
>>> next(my_iterator)
'e'
>>> next(my_iterator)
'l'
>>> next(my_iterator)
'l'
>>> next(my_iterator)
'o'
>>> next(my_iterator)
' '
>>> next(my_iterator)
'w'
>>> next(my_iterator)
'o'
>>> next(my_iterator)
'r'
>>> next(my_iterator)
'l'
>>> next(my_iterator)
'd'
>>> next(my_iterator)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
>>>
```

Creating an itterator

On crée un itérateur avec les methode speciales __iter__ et __next__

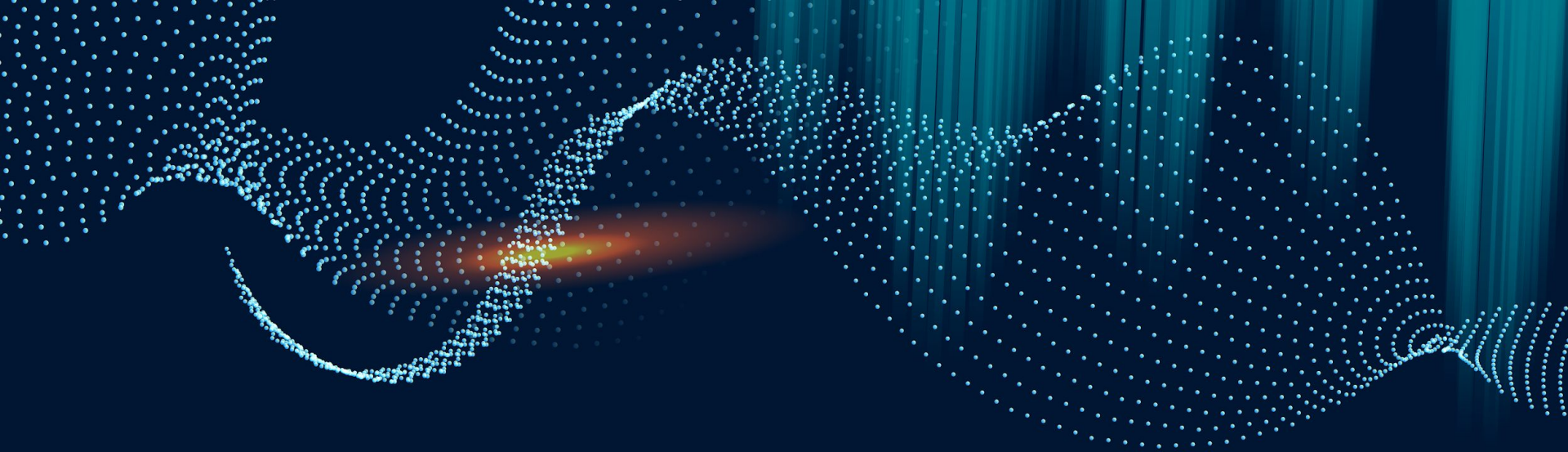
```python
class Reverse:
    def __init__(self, chaine_a_parcourir):
        """On se positionne à la fin de la chaîne"""
        self.chaine_a_parcourir = chaine_a_parcourir

    def __iter__(self):
        self.position = len(self.chaine_a_parcourir)
        return self

    def __next__(self):
        if self.position == 0: # Fin du parcours
            raise StopIteration
        self.position -= 1 # On décrémente la position
        return self.chaine_a_parcourir[self.position]
```

Générateurs

```
>>> def simple_gen():
...     yield 1
...     yield 2
...     yield 3
...
>>> iterator = iter(simple_gen())
>>> next(iterator)
1
>>> next(iterator)
2
>>> next(iterator)
3
>>> next(iterator)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
>>>
```

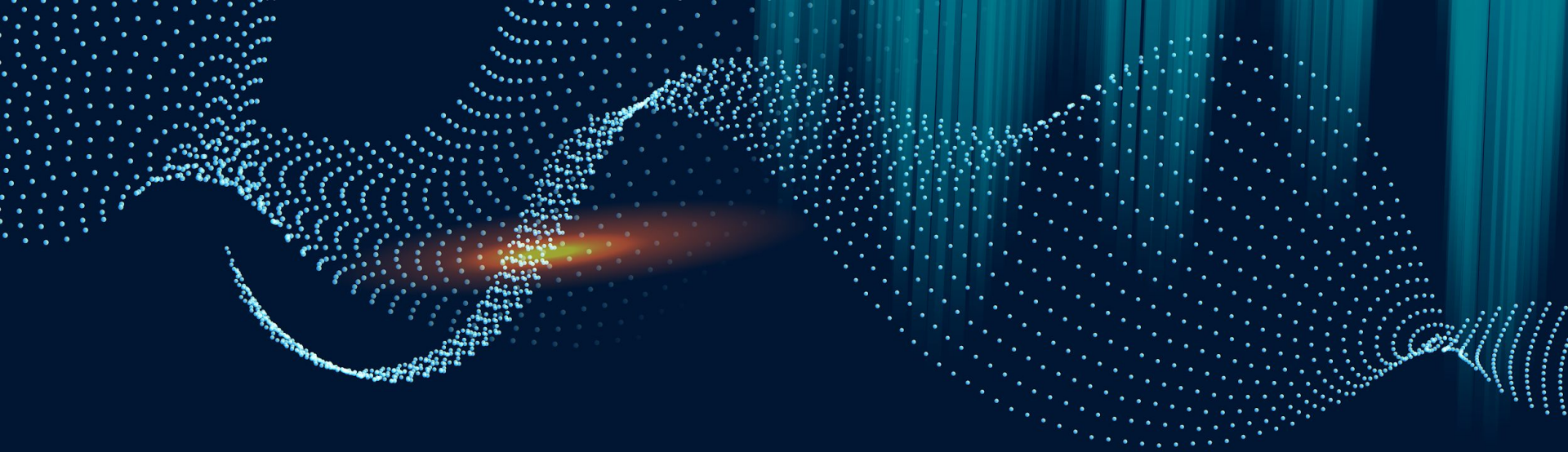
Générateurs

```
>>> def simple_gen():
...     yield 1
...     yield 2
...     yield 3
...
>>> for nombre in simple_gen(): # on exécute la fonction
...     print(nombre)
...
1
2
3
>>>
```



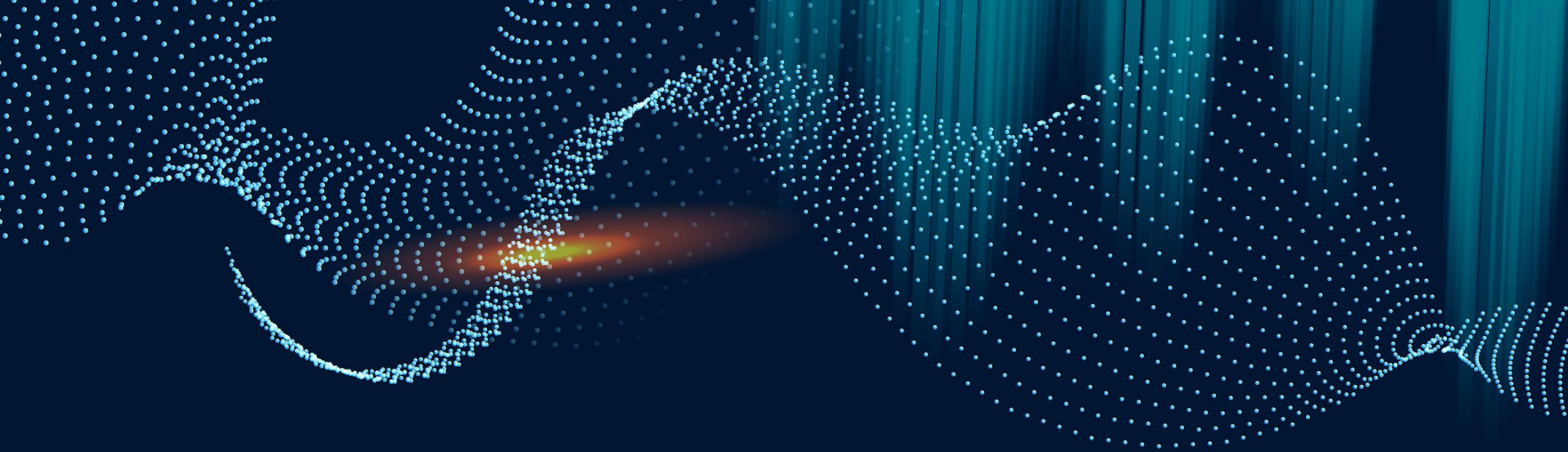
Exercice

Réécrire la fonction range avec un itérateur et générateur



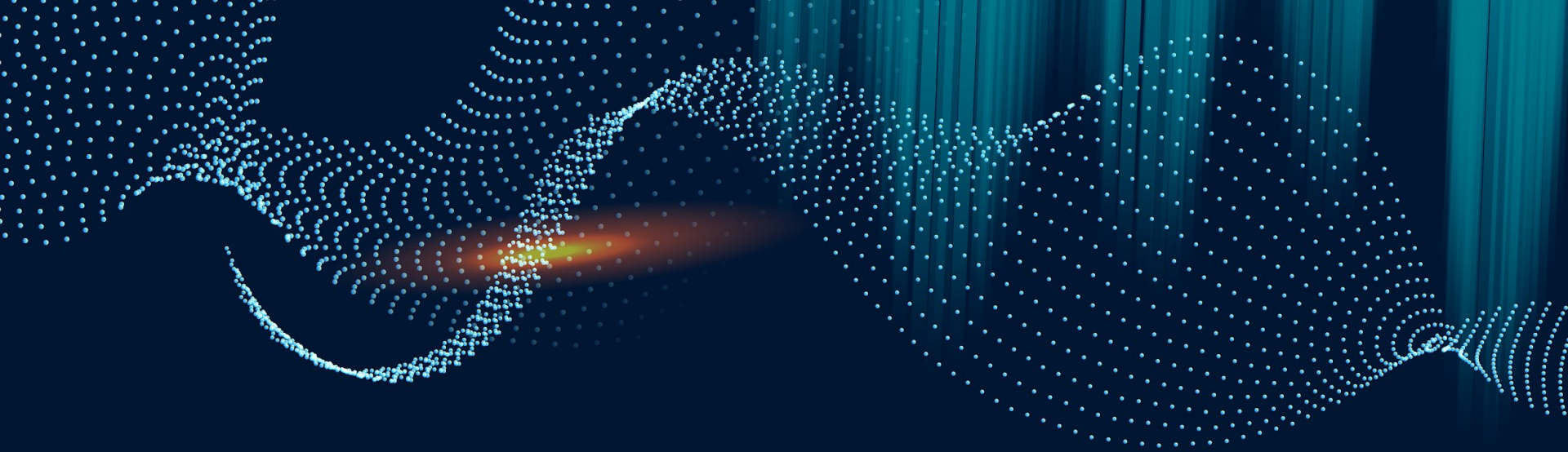
02 Decorators

You can enter a subtitle here if you need it



Exercice

Écrire un décorateur qui permet de calculer le temps d'exécution d'une fonction



03 Files

Chemins relatifs et absolus

Chemin absolu

Chemin du fichier static dans L'OS par exemple /home/... ou C:\....

Chemin relatif

Chemin du fichier à partir de l'emplacement du script

Mode d’ouverture d’un fichier

‘r’

Pour lire un fichier, renvoie une erreur si le fichier n'existe pas

‘w’

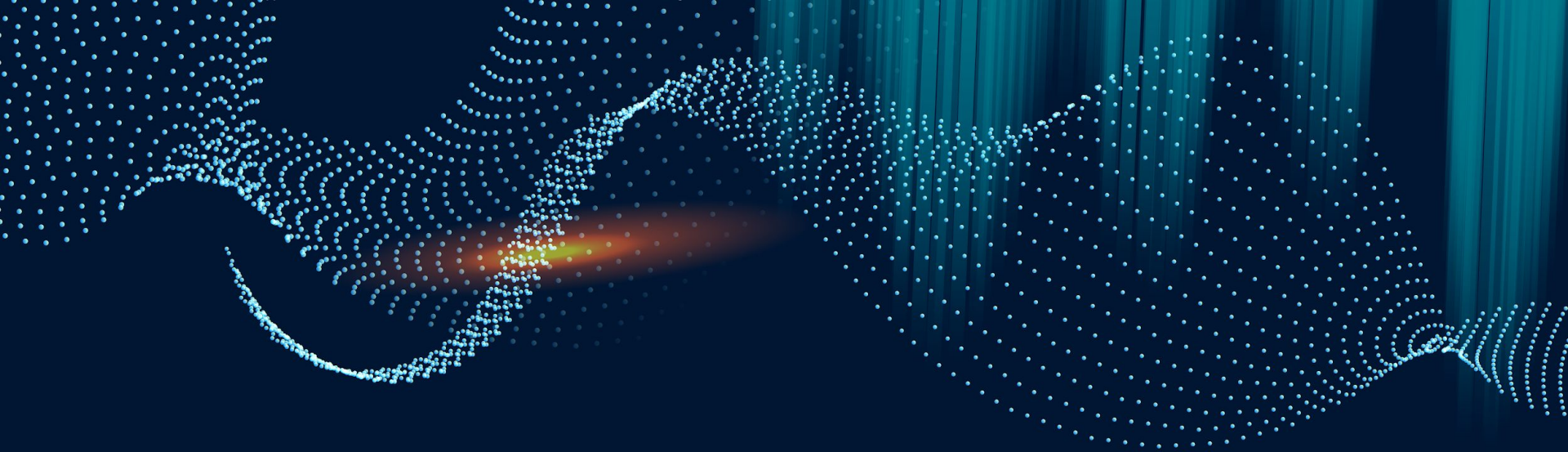
Pour écrire dans un fichier, Si le fichier n'existe pas, il est créé.

‘a’

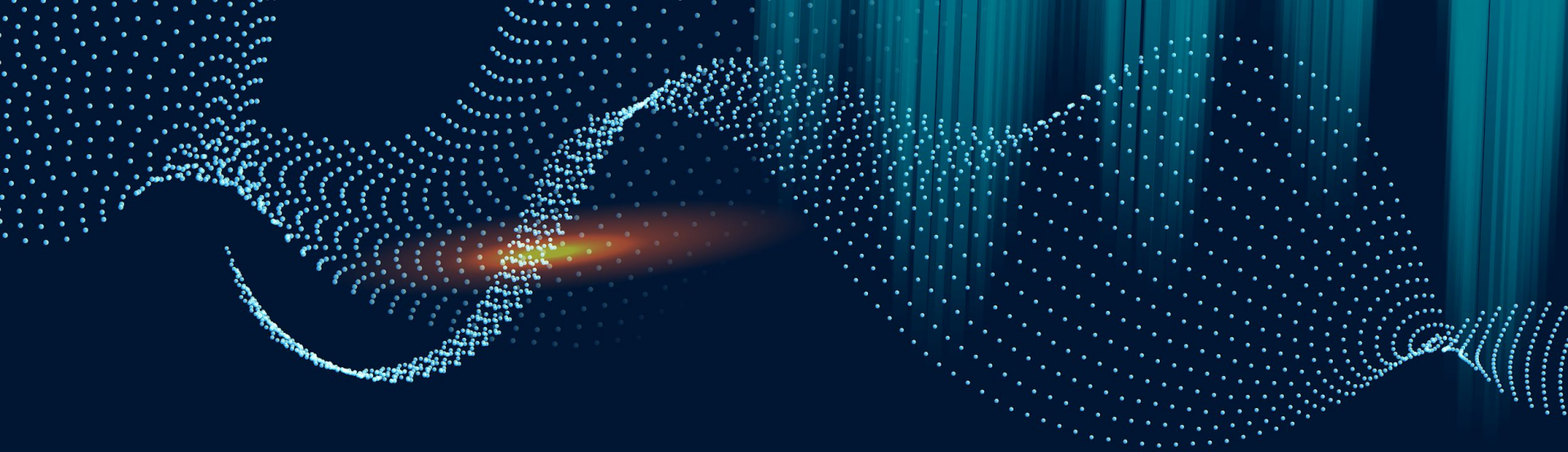
Ecrire a la fin du fichier (comme un append)

Syntaxe

```
with open('test_file.txt', 'r') as file:
    string = file.read()
    print(string)

with open('test_file.txt', 'w') as file:
    file.write(string + "\nCe fichier a etait modifer")
```

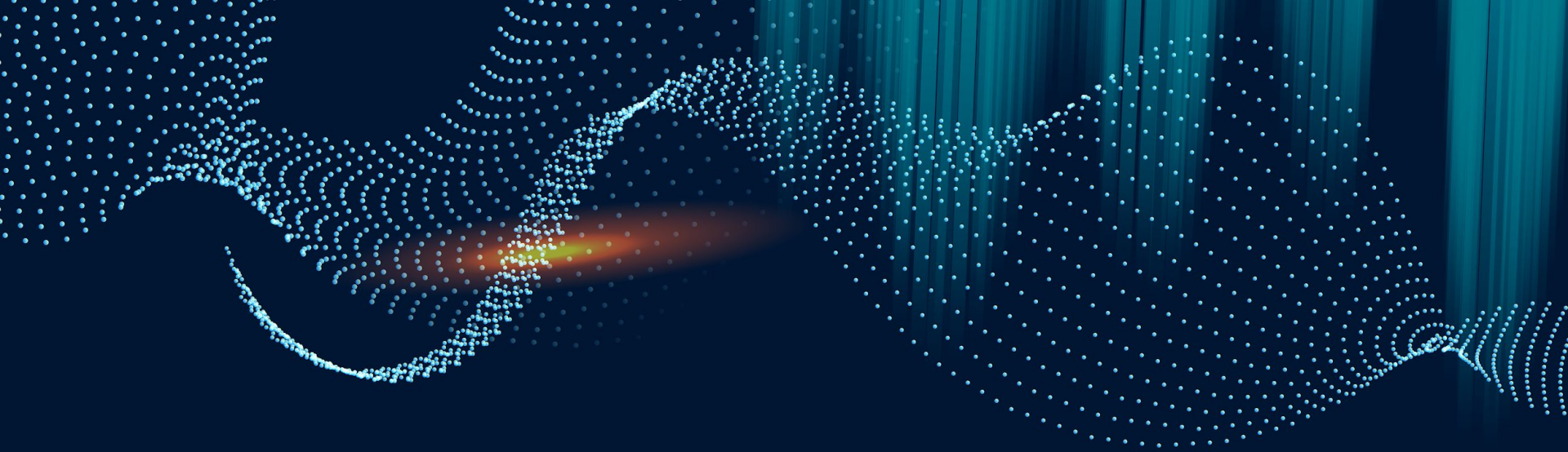


Serialisation



Exercice

Essayer de programmer un jeux du pendu

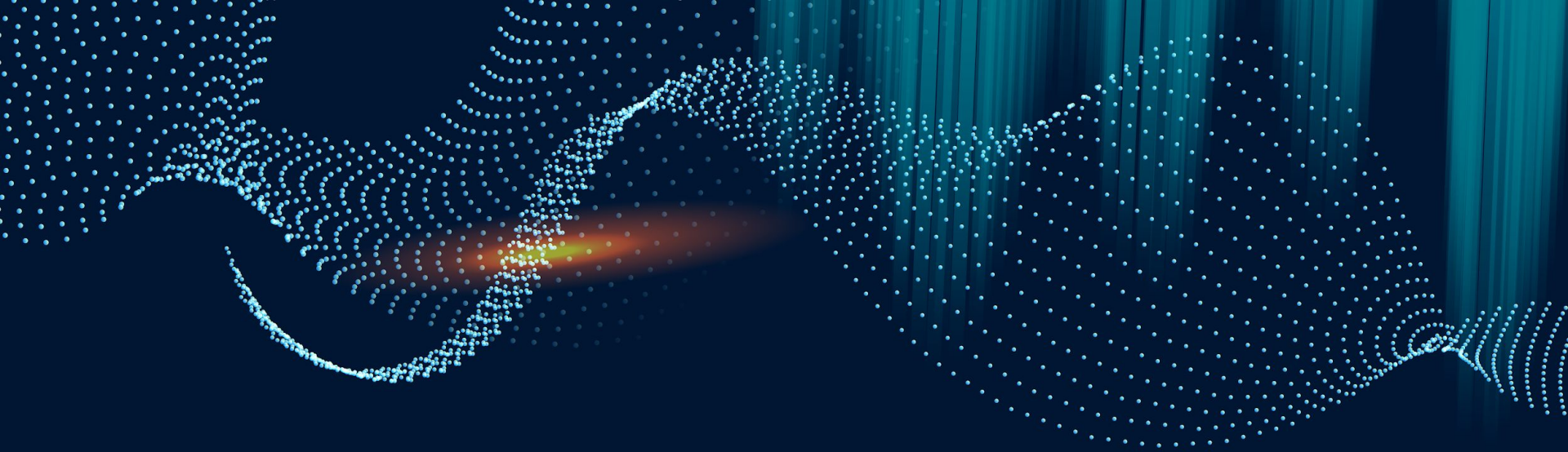


04 Python reference

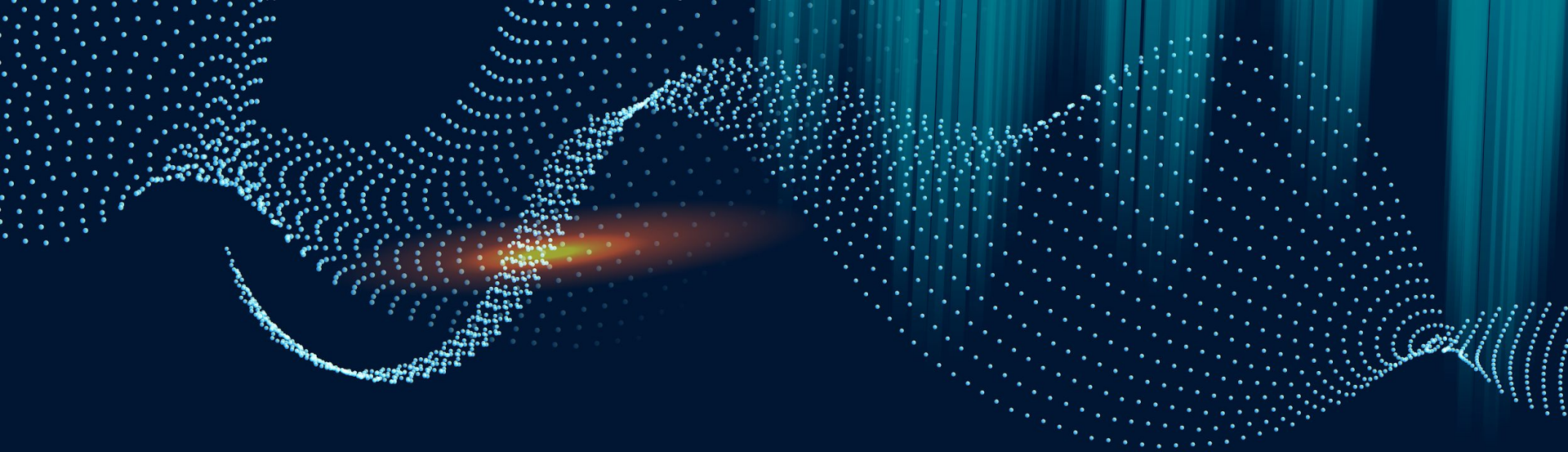
C’est quoi une référence

- En python tous est objet
- Les variables en python sont des références vers les objets
- Les références sont proches des pointeurs en C ou C++ mais en plus haut niveau.

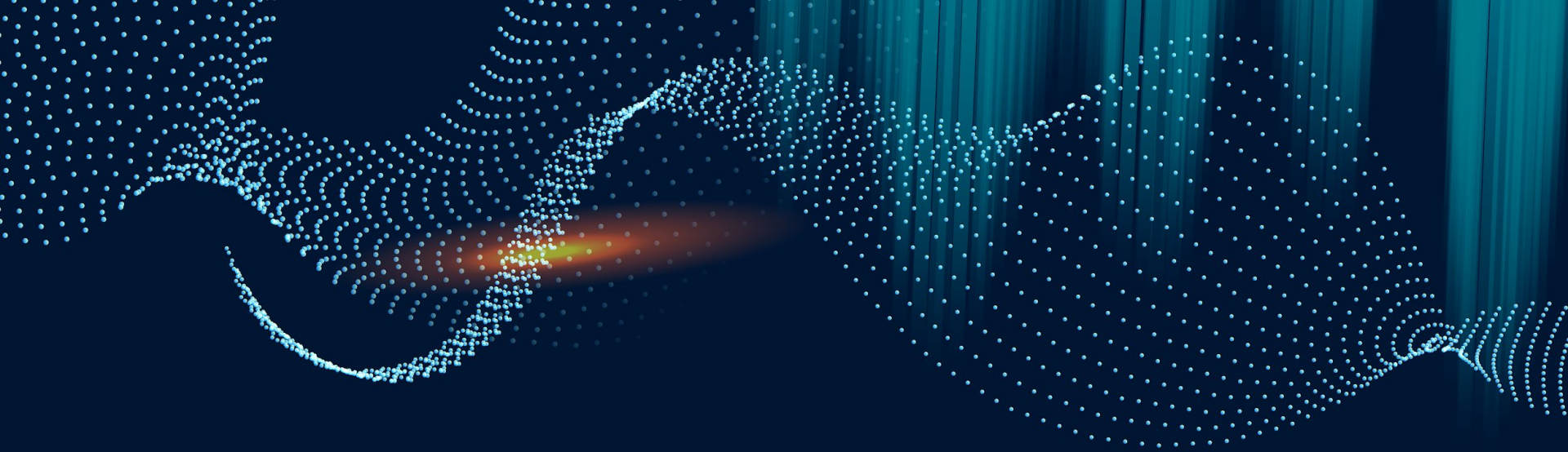
```
>>> my_list = [1,2,3]
>>> new_list = my_list
>>> new_list.append(4)
>>> new_list
[1, 2, 3, 4]
>>> my_list
[1, 2, 3, 4]
>>>
```



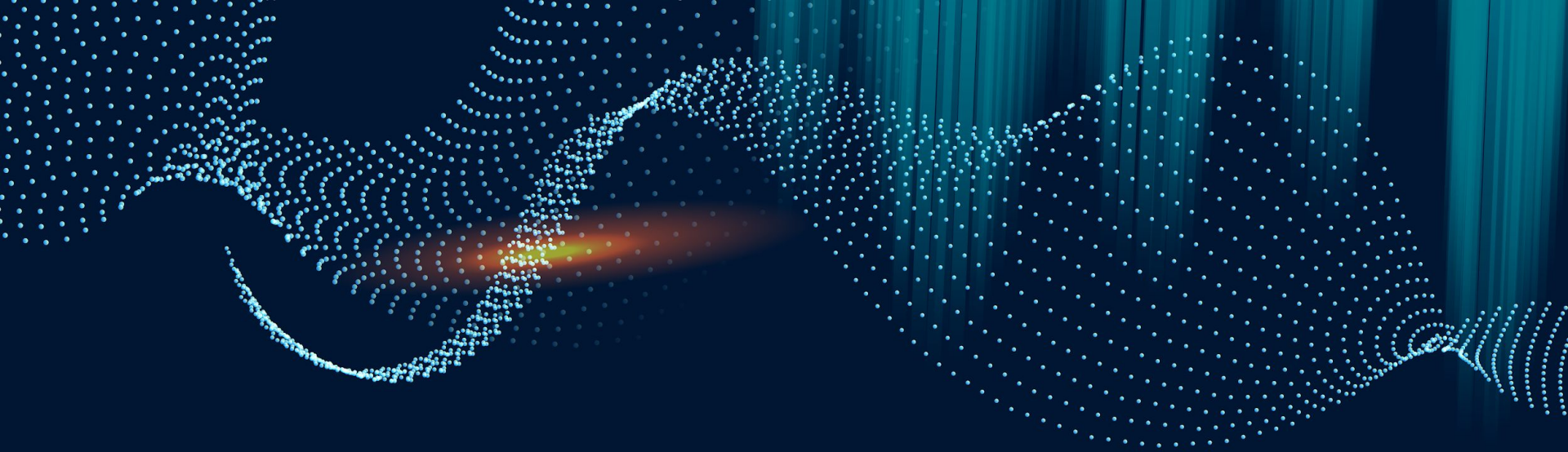
0X | Bonus



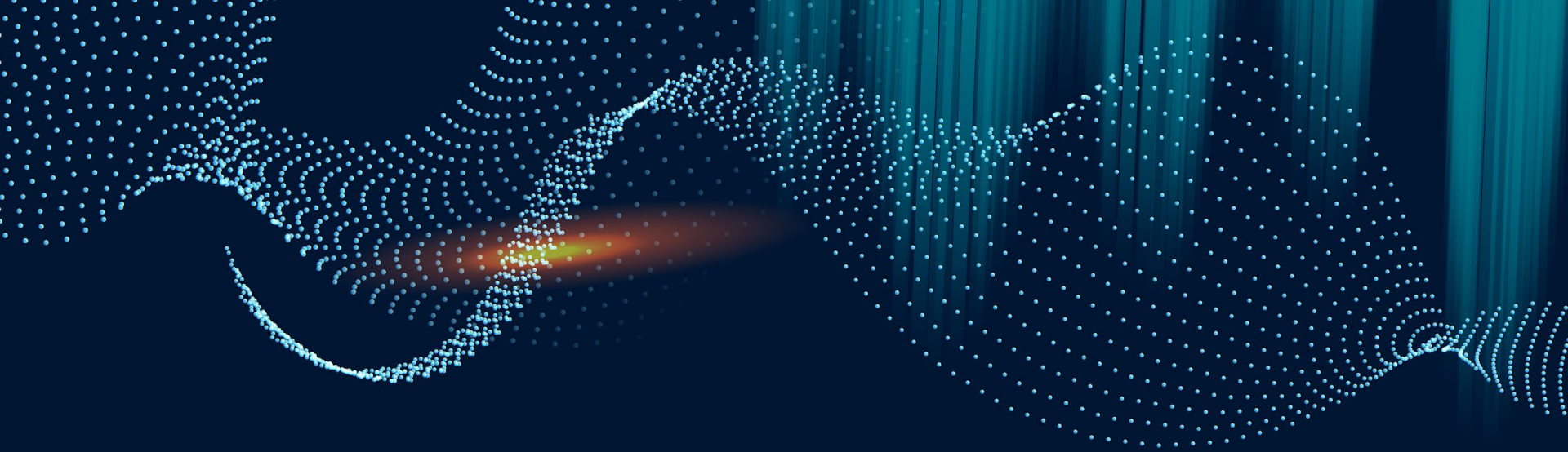
Sets



Lambda Functions



05 Use PIP



06 Write good code in python

PEP20 : The Zen of Python

- Une liste de recommandation pour écrire du code python propre

- On retrouve par exemple :
  - *“Beautiful is better than ugly.”*
  - *“Explicit is better than implicit.”*
  - *“Complex is better than complicated.”*
  - *“Errors should never pass silently.”*
  - *“If the implementation is hard to explain, it's a bad idea.”*

- *On peut y avoir acces dans l’interpreteur avec “import this”*

- Accessible via https://www.python.org/dev/peps/pep-0020/

PEP8 : des conventions plus precisent

- On retrouve dans la PEP8 des conventions plus precisent sur comment nommer les variables, les espaces à laisser entre les fonctions...etc

- Par exemple
  - Utiliser 4 espace pour une indentation
  - Limit all lines to a maximum of 79 characters.
  - Line break before operator
  - Imports should usually be on separate lines

- Accessible via https://www.python.org/dev/peps/pep-0008/

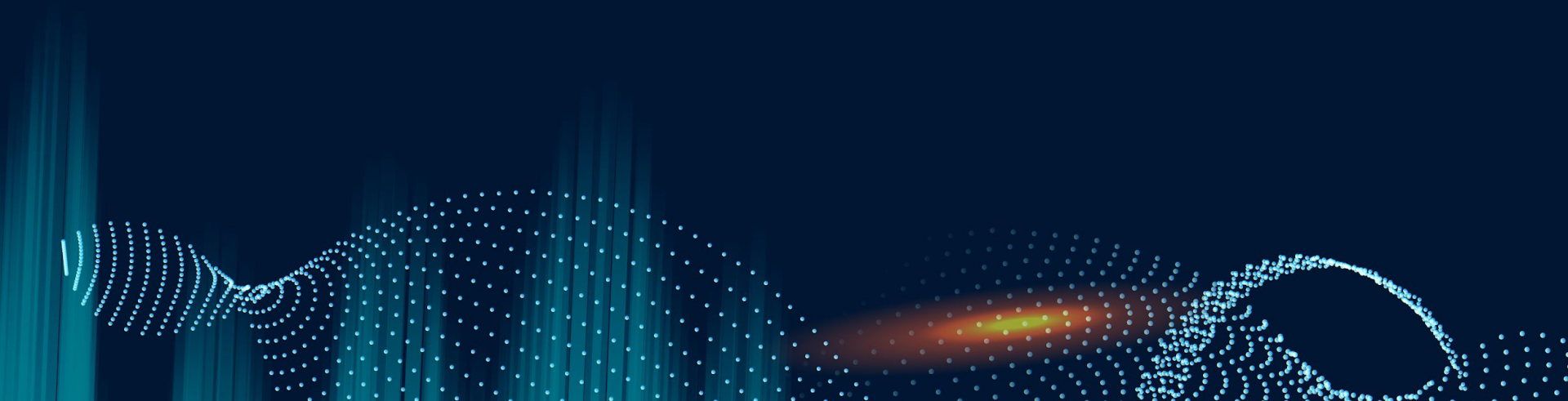
PEP8 : Comment écrire des docstring

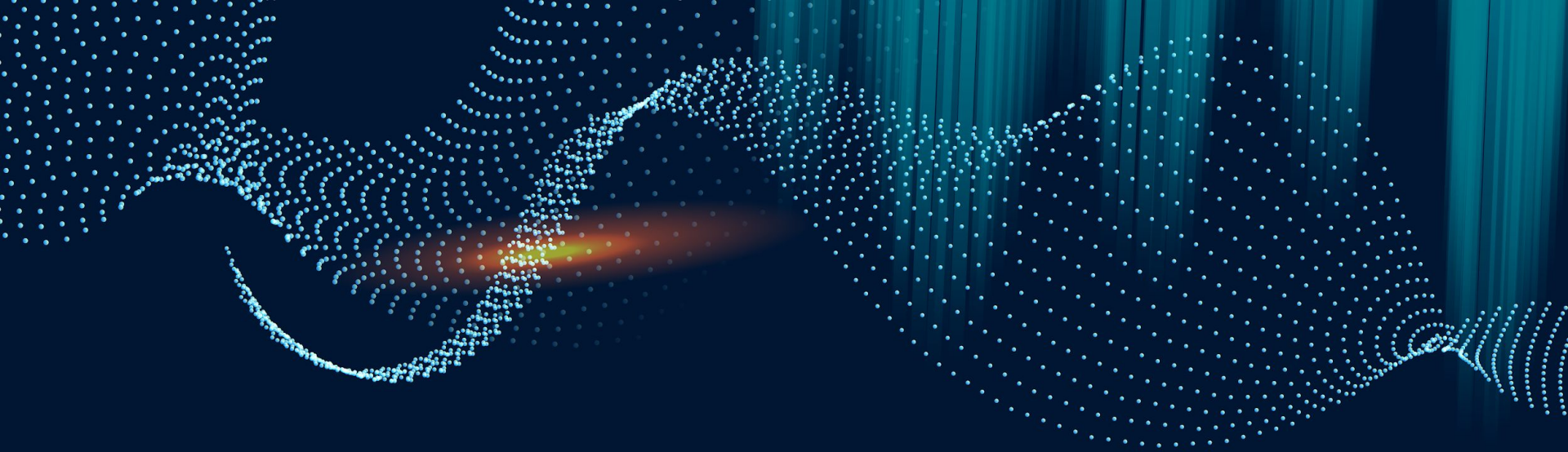
- Contient des directives pour écrire de bonne docstrings et documentation
- Accessible via https://www.python.org/dev/peps/pep-0257/

Pylint

- Une librairy qui permet de “noter” la lisibilité et le respect des conventions de notre code
- Accessible via https://www.pylint.org/
- Installable via ‘pip’ : pip install pylint
- Executer : pylint <module name>
- Peut être ajouté directement à un éditeur ou ide

ANY Question?





Movie night

THANKS!

Do you have any questions?
You can ask it in the discord server

By OpenMindsClub