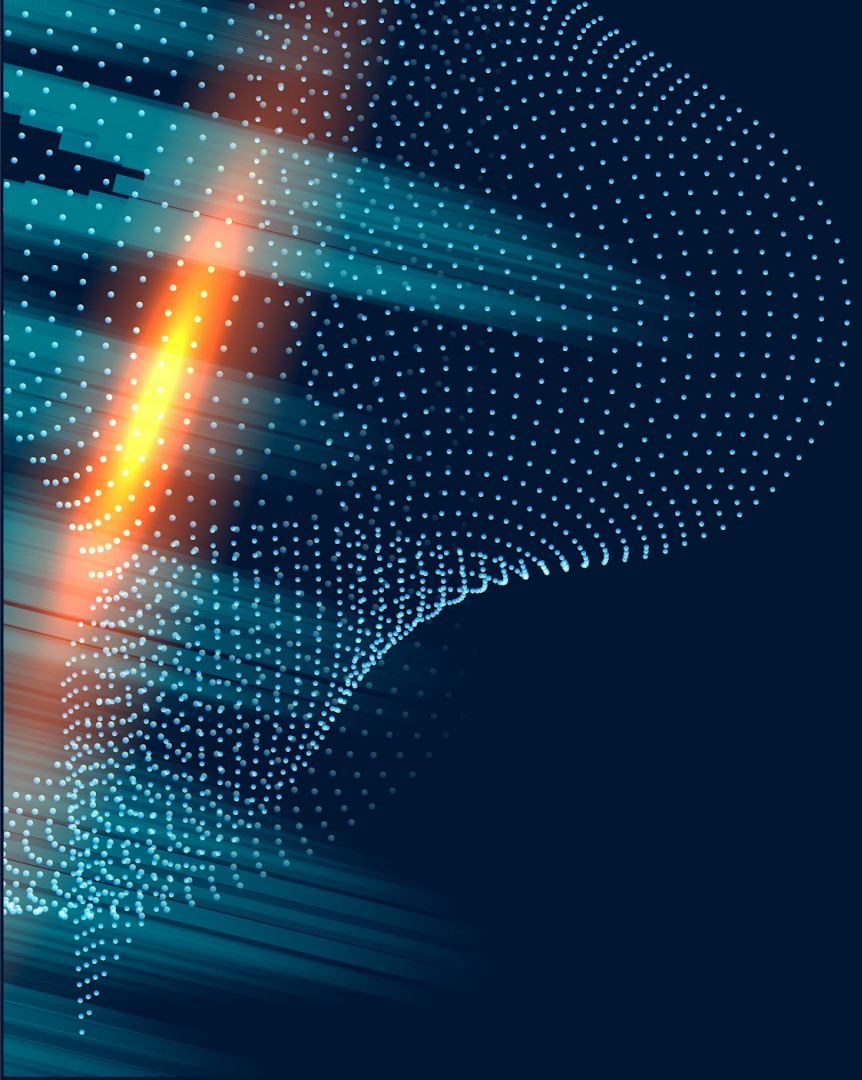
PYTHON Workshop II

Data Structures and OOP

“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

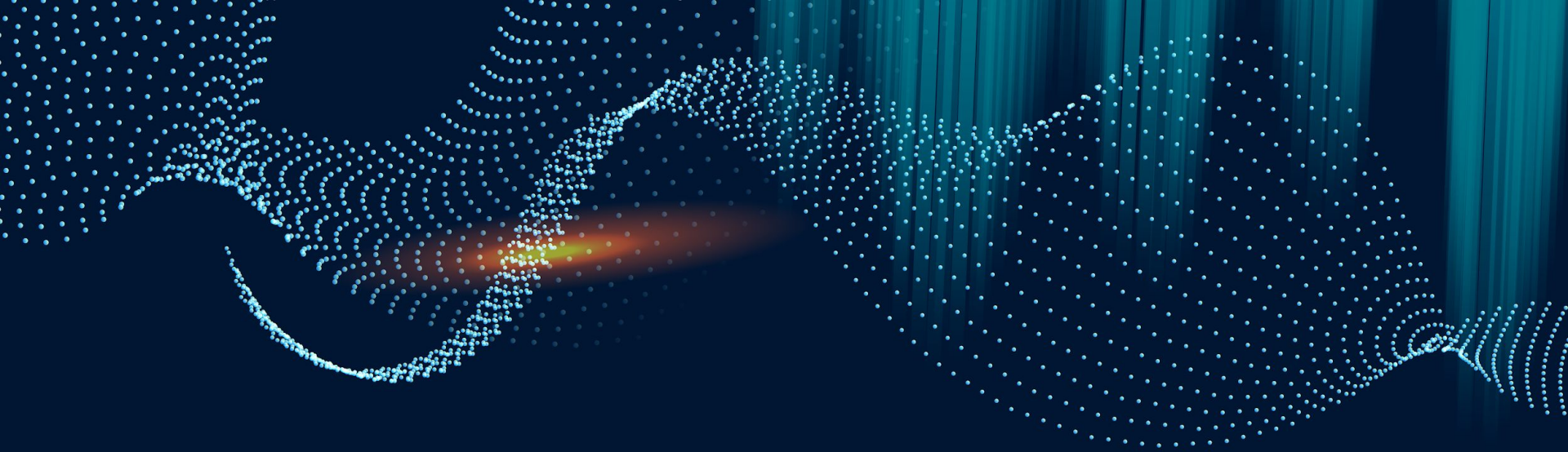
DATA Structures in Python

02

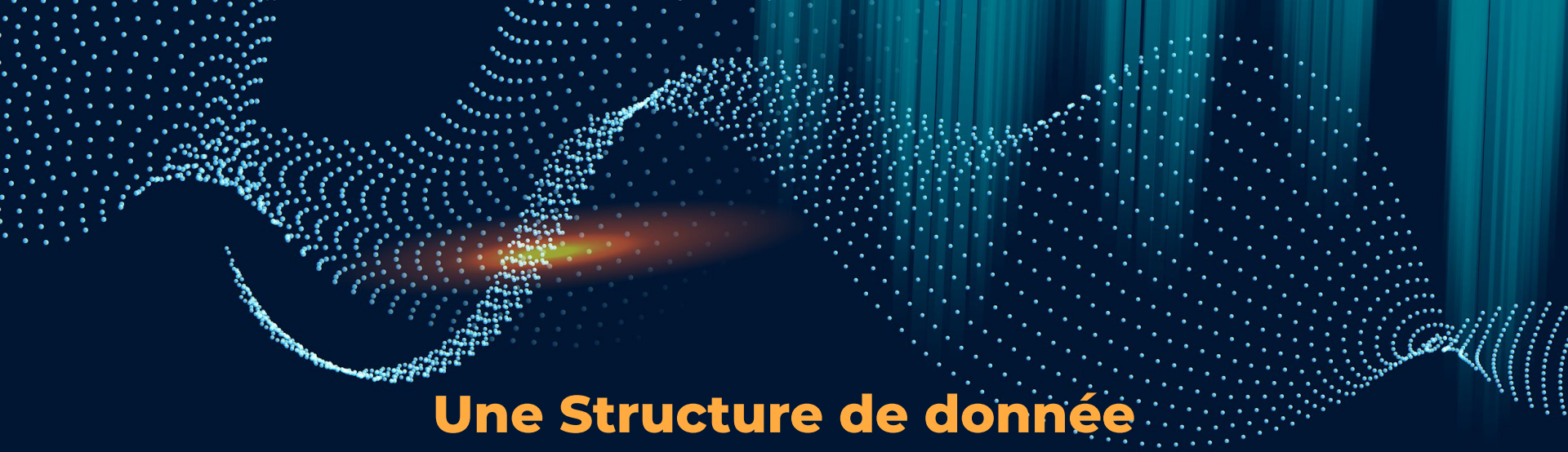
Oriented Object Programming

03

Exploring DATA Structures Methods



01 DATA Structures in Python



Une Structure de donnée c’est quoi?

Objet mutable vs non mutable

Mutable

Les données à l'intérieur des objets peuvent changer au fil de l'exécution

Immuable

Les données à l'intérieur des objets ne peuvent pas changer au fil de l'exécution

Les symboles des différentes SD

Structures de données	Symbole pour l'initialisation	Initialisation
String	“” ou ‘’	My_str = “”
list	[]	My_list = []
tuple	()	My_tuple = ()
dict	{}	My_dict = {}

Initiez-vous aux listes

- Structure mutable qui permet de “ranger” des données.

5	15	5.4	“This is a string”	85
0	1	2	3	4

Initialiser une liste

```python
# initialiser une liste vide
my_list = []

# autre maniere d'initialiser initialiser une liste vide
my_list = list()

# initialiser une liste
my_list = [2, 5, "python"]
```

Opérations sur les listes

- Ajouter un element
  my_list.append(element)
- Accéder à un élément
  my_list[i]
- Modifier un élément
  my_list[i] = x
- Supprimmer un element
  del (my_list[i])
- Avoir le nombre d'element
  len (my_list)

```
>>> my_list
[2, 5, 'python']
>>> my_list[0]
2
>>> my_list.append(5)
>>> my_list
[2, 5, 'python', 5]
>>> my_list[0] = 3
>>> my_list
[3, 5, 'python', 5]
>>> del(my_list[1])
>>> my_list
[3, 'python', 5]
>>> len(my_list)
3
```

Accéder a des sous listes

```
>>> my_list = [2, 5, "python", 6]
>>> my_list[:2]
[2, 5]
>>> my_list[2:]
['python', 6]
```

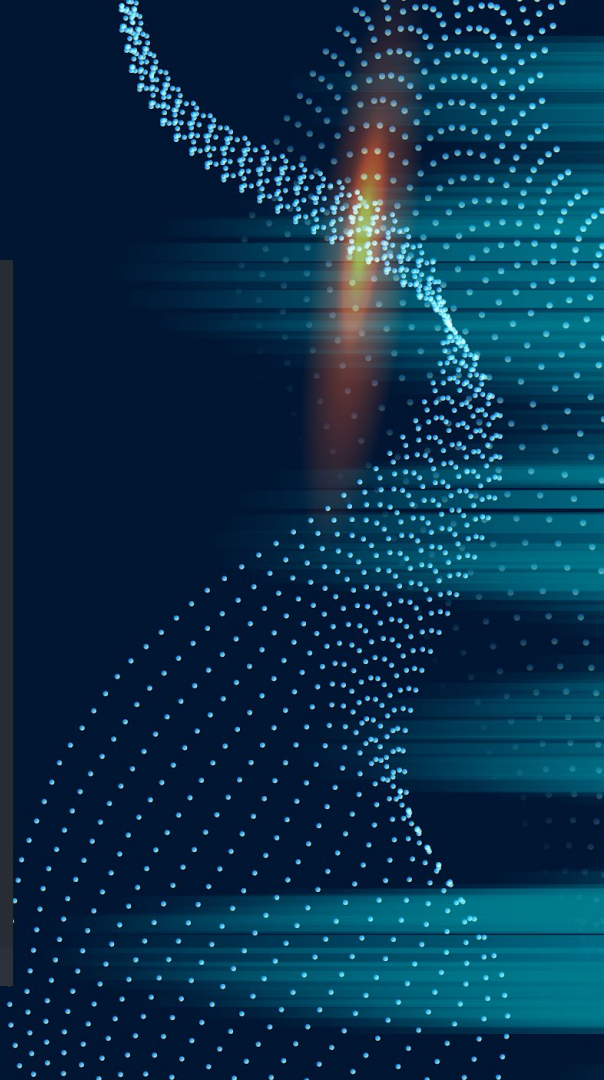
Les Tuples

- Un tuple est très similaire à une liste, avec la particularité qu'il est immuable, ce qui veut dire que l'objet ne peut pas changer une fois initialiser.

```
>>> my_tuple = (1,2,3)
>>> my_tuple[1] = 8
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Initialiser un tuple

```python
# initialiser un tuple vide
my_tuple = ()

# autre maniere d'initialiser un tuple vide
my_tuple = tuple()

# initialiser un tuple
my_tuple = (2, 5, "python")

#  initialiser un tuple a partire d'une liste
my_tuple = tuple([1, 2, 5])
```



Retour sur les Strings

- Nous avons déjà vu le String, ce qui faut savoir en plus c’est que le String est immuable, donc qu’on ne peut pas changer ces éléments une fois crée


SO how do I change The content of a string ?
That's the neat part, you don't

Par contre on peut contourner le Problème

- Toutes Les méthodes et manipulation de Strings retourne un nouvel objet (une nouvelle chaine de caractère)

```
# changer la valleur d'un caractere dans un String

my_string = "hello world"
# on remplace le ' ' par un '_'
my_string = my_string[:5] + "_" + my_string[6:]

my_string = "hello world"
# dans ce cas on peu aussi utiliser la methode replace
my_string = my_string.replace(" ", "_")
```

```python
my_list = [1, 2]
my_list.append(3) # cette methode modifie directement la liste

my_string = "hello world"
my_string.replace(" ", "_") # cette methode ne modifie
# pas la chaine de caractere par contre elle retourne une nouvelle

# on doit donc assigner la valeur a une autre variable
# si on veut l'utilier
my_string = my_string.replace(" ", "_")
```

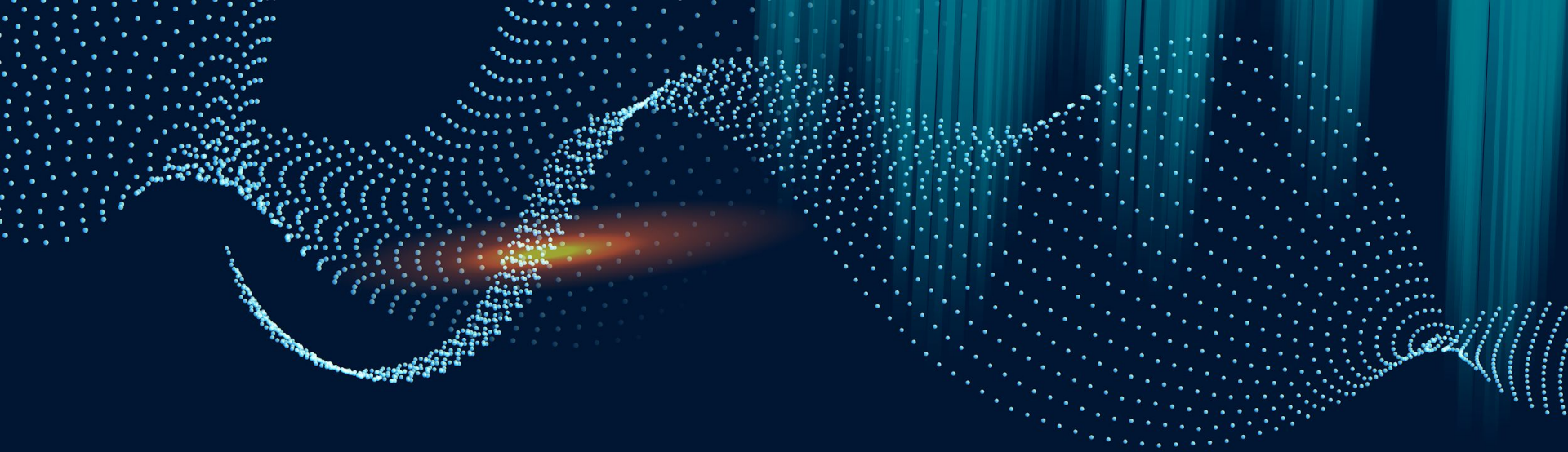
Découvrez les Dictionnaire

- Un dictionnaire fonctionne sur un logique de clé/valeur, chaque élément du dictionnaire à une clé qui est unique avec une valeur qui lui est associé.

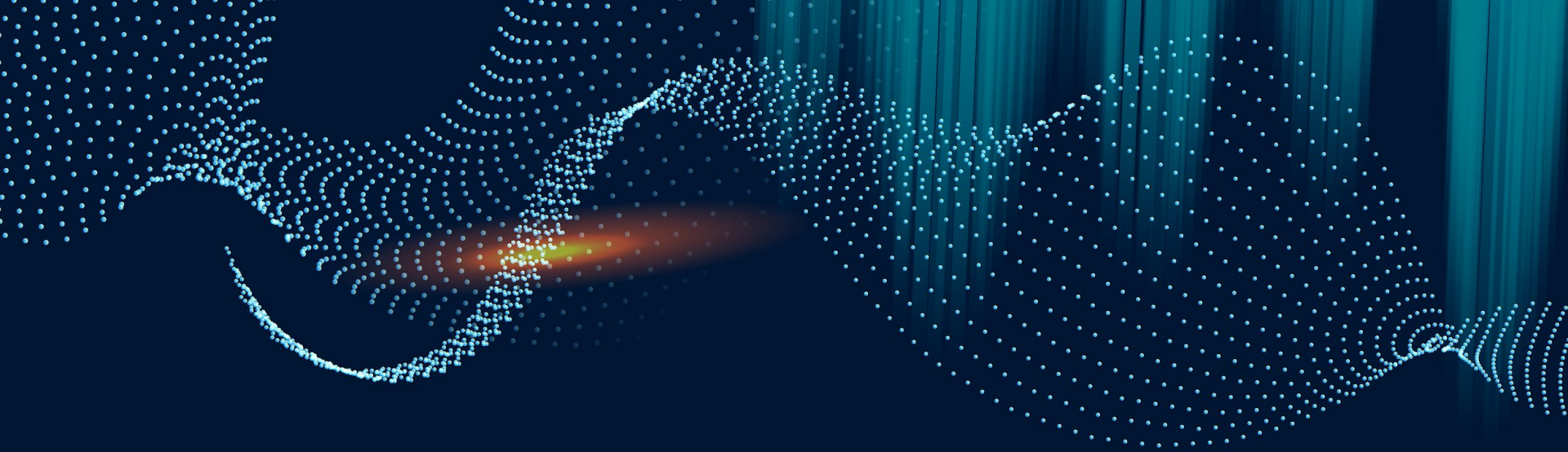
```
my_dictionary_user = {
    "pseudo" : "aymenkhs",
    "nom" : "khouas",
    "prenom" : "aymen",
    "age" : 23,
}
```

Initialiser un dictionnaire

```
my_dictionary = {}

my_dictionary = {"orange" : 2, "pomme": 3}

my_dictionary = dict()

my_dictionary = dict(orange=2, pomme=3)
```



Let’s talk about sets!

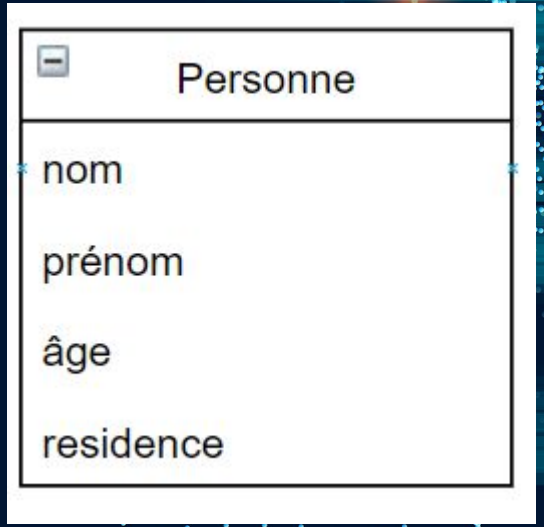


02 Oriented Object Programming

Une Classe ? Un Objet ? Ça se mange ?

Notre première class

```python
class Personne: # Définition de notre classe Personne
    """Classe définissant une personne caractérisée par :
    - son nom
    - son prénom
    - son âge
    - son lieu de résidence"""


    def __init__(self): # Notre méthode constructeur
        # (On definit les attributs de notre class
        # dans cette methode init)
        self.nom = "khouas"
        self.prenom = "aymen"
        self.age = 23
        self.residence = "bab ezzouar"
```



Instancier et accéder à l'objet

```
>>> user = Personne()
>>> user
<__main__.Personne object at 0x7f0f23e29040>
>>> print(user.nom)
khouas
>>> print(user.prenom)
aymen
```

Passer des attributs a nos class

```python
class Personne: # Définition de notre classe Personne
    """Classe définissant une personne caractérisée par :
    - son nom
    - son prénom
    - son âge
    - son lieu de résidence"""

    def __init__(self, nom, prenom, age, residence): # Notre méthode constructeur
        # (On definit les attributs de notre class
        # dans cette methode init)
        self.nom = nom
        self.prenom = prenom
        self.age = age
        self.residence = residence

# Instancier l'objet
user = Personne("khouas", "aymen", 23, "bab ezzouar")
```

Les méthodes

- Les méthodes sont des “fonctions” propre à l’objet.

```python
class Personne: # Définition de notre classe Personne
    def __init__(self, nom, prenom, age, residence): #
        self.nom = nom
        self.prenom = prenom
        self.age = age
        self.residence = residence

    def est_mineur(self):
        return self.age < 18

user = Personne("khouas", "aymen", 23, "bab ezzouar")
print(user.est_mineur())
```

Les attributs et méthodes de class

- Des attributs et méthode qui vont s'appliquer non pas à une instance de class (un objet) mais à toute la class

```python
class Personne: # Définition de notre classe Personne

    NB_USERS = 0
    def __init__(self, nom, prenom, age, residence): #
        self.nom = nom
        self.prenom = prenom
        self.age = age
        self.residence = residence
```

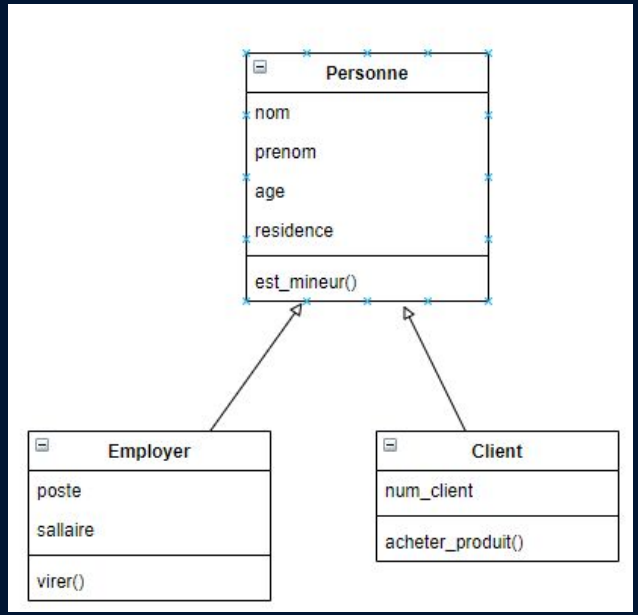
Les attributs et méthodes de class

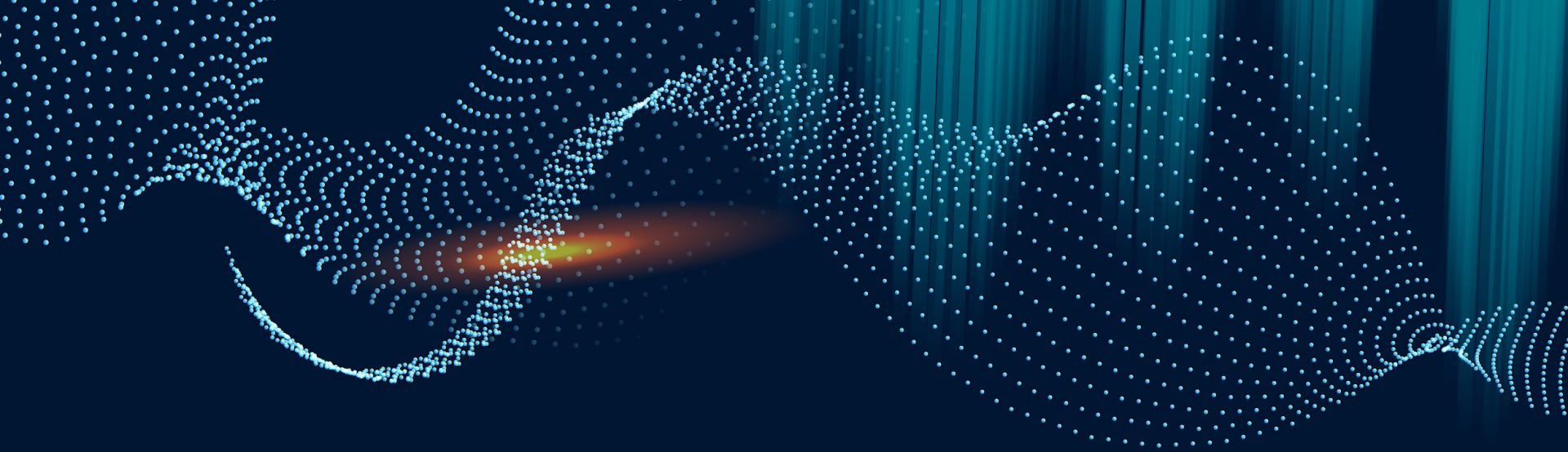
- Pour définir une méthode de class on ajoute @classmethod avant cette dernière

```python
class Personne:
    MAX_USERS = 2000
    NB_USERS = 0
    def __init__(self, nom, prenom, age, residence):
        self.nom = nom
        self.prenom = prenom
        self.age = age
        self.residence = residence
        Personne.NB_USERS += 1

    @classmethod
    def check_users(cls):
        if Personne.NB_USERS >= Personne.MAX_USERS:
            print("database full, we must free some space")
        else:
            print("there is ennough space")
```

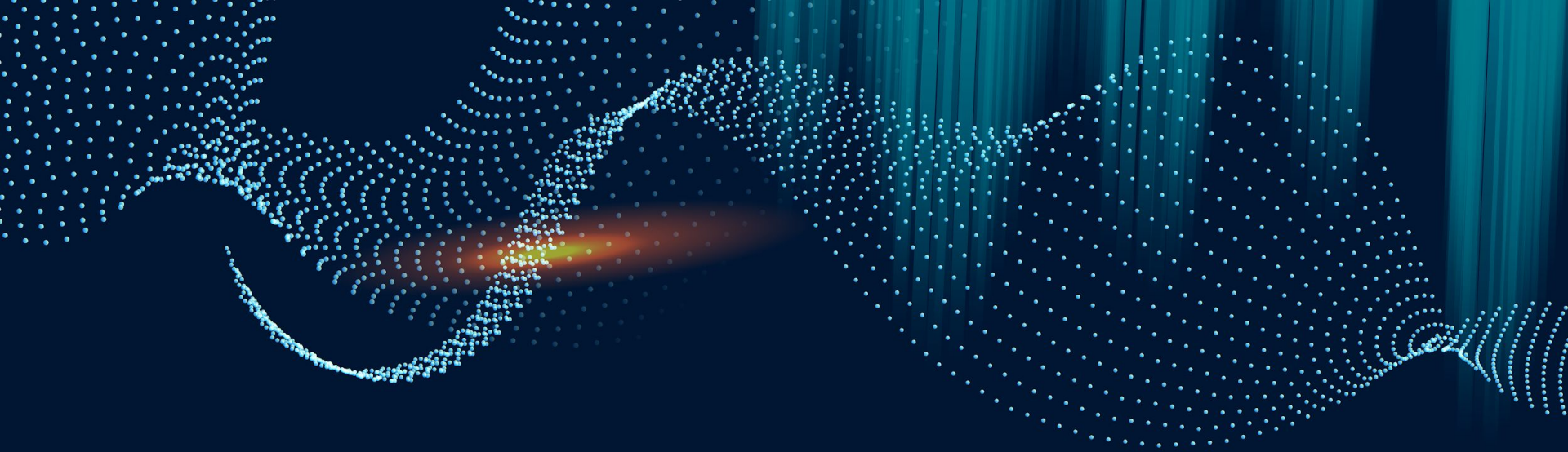
Héritage

- l'héritage est une fonctionnalité qui permet de déclarer qu'une classe fille sera modelée sur une classe mère

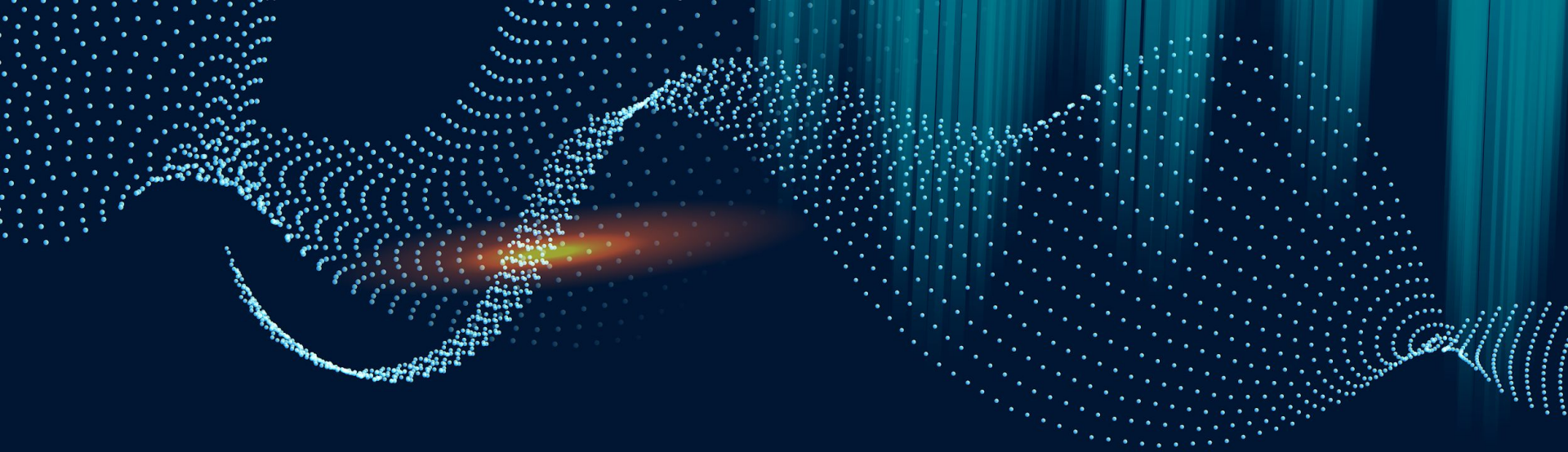




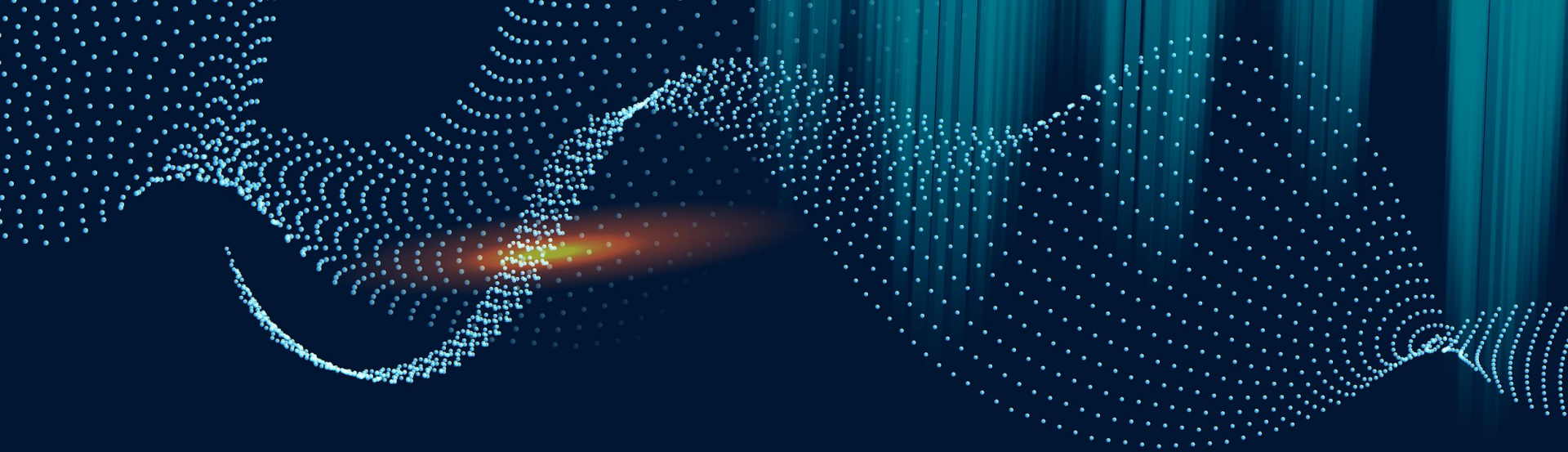
Les propriété



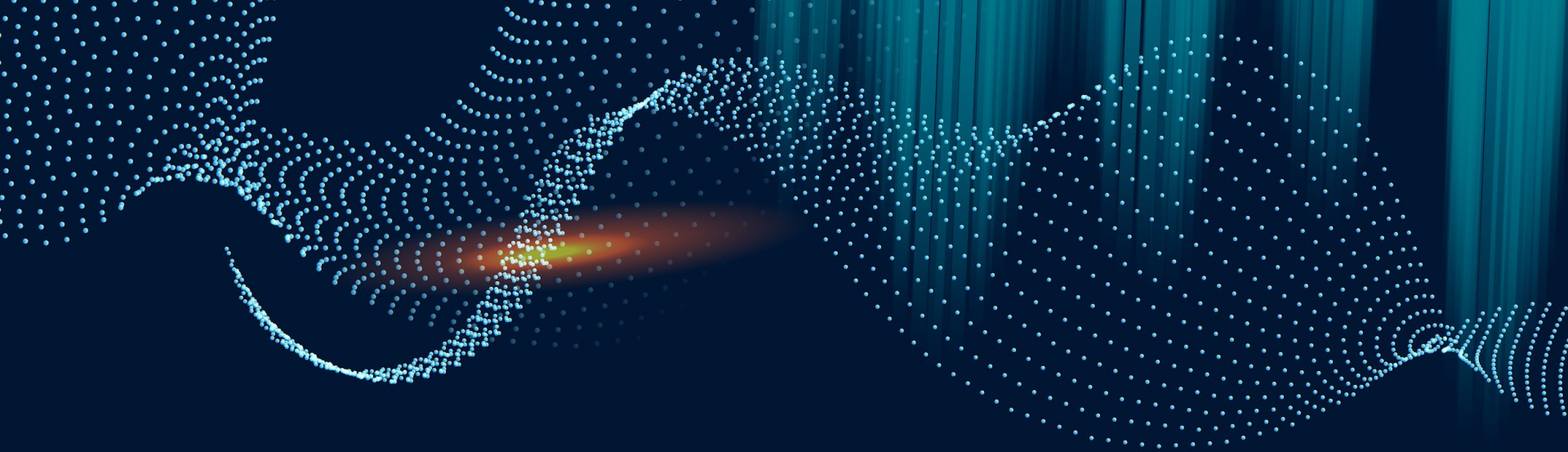
03 Exploring DATA Structures Methods



0X | Bonus



Les fichiers



Sets

THANKS!

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

By OpenMindsClub