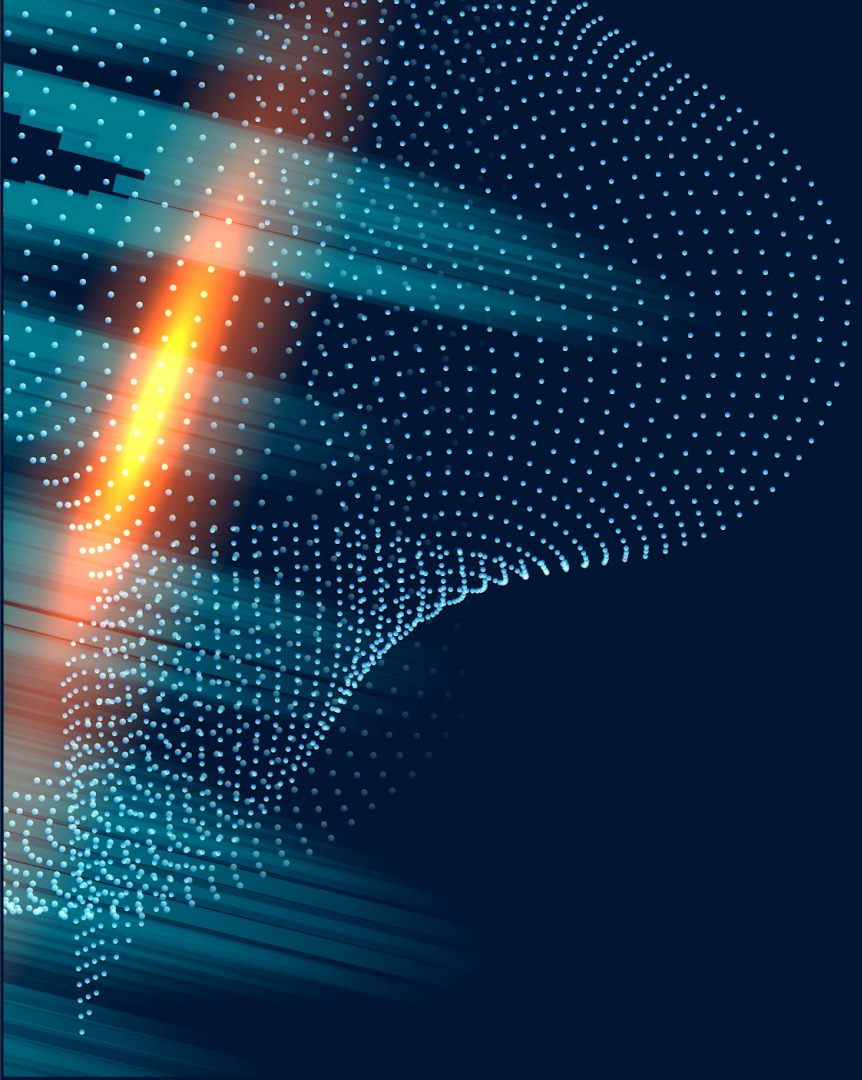
PYTHON Workshop

Advanced Python Concepts

“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

More OOP concepts

02

Special methods

03

Exceptions

04

Return to Lists

05

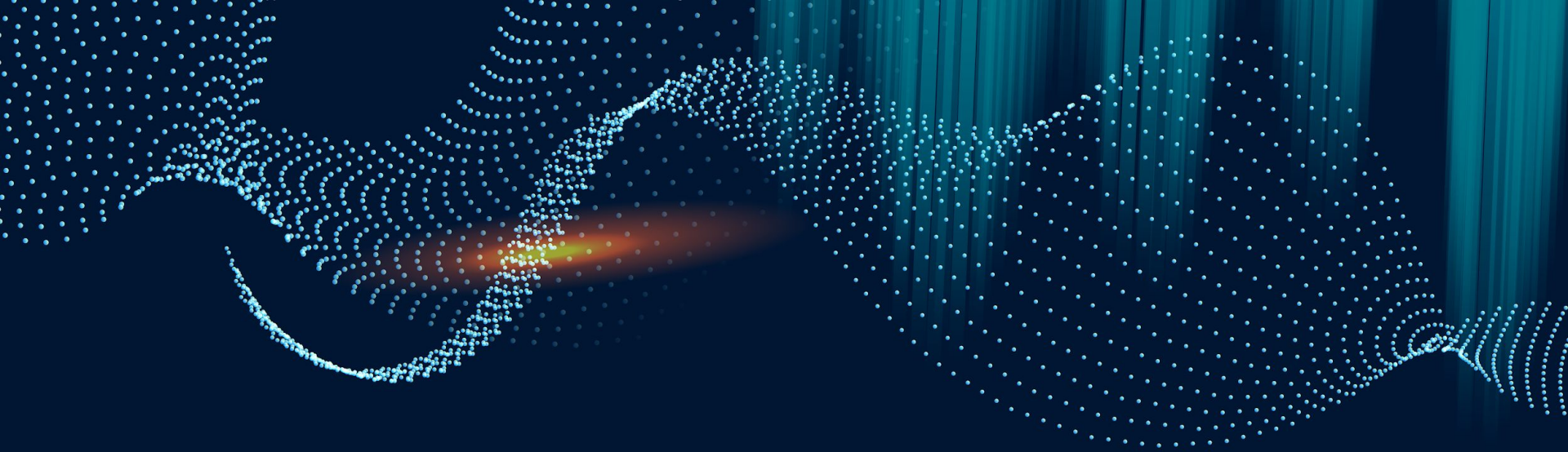
Iterators

(haven't been done in this session)

06

Decorators

(haven't been done in this session)



01 More OOP concepts

Class method vs static method

Class method

- Prend ‘cls’ comme premier paramètre
- Peut modifier les attributs et l’état de la classe
- On utilise le décorateur @classmethod pour la créer.

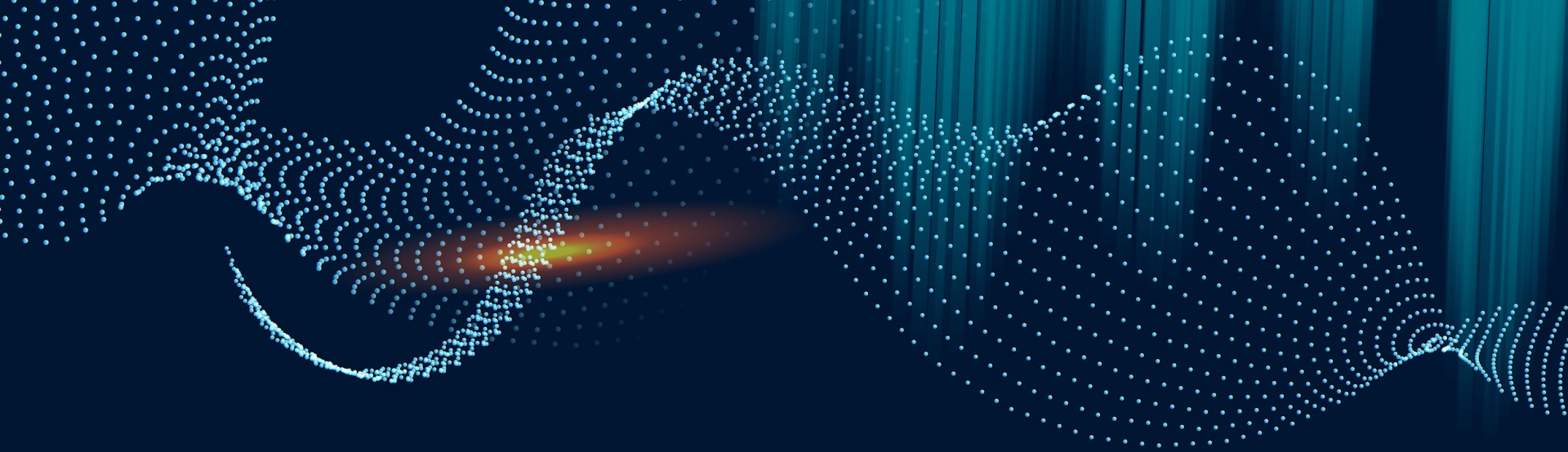
static method

- Ne prend pas ‘cls’ comme premier paramètre.
- Prend des paramètres et fonctionne comme une simple fonction à l’intérieur de la classe sans y avoir vraiment accès
- On utilise le décorateur @staticmethod pour la créer.

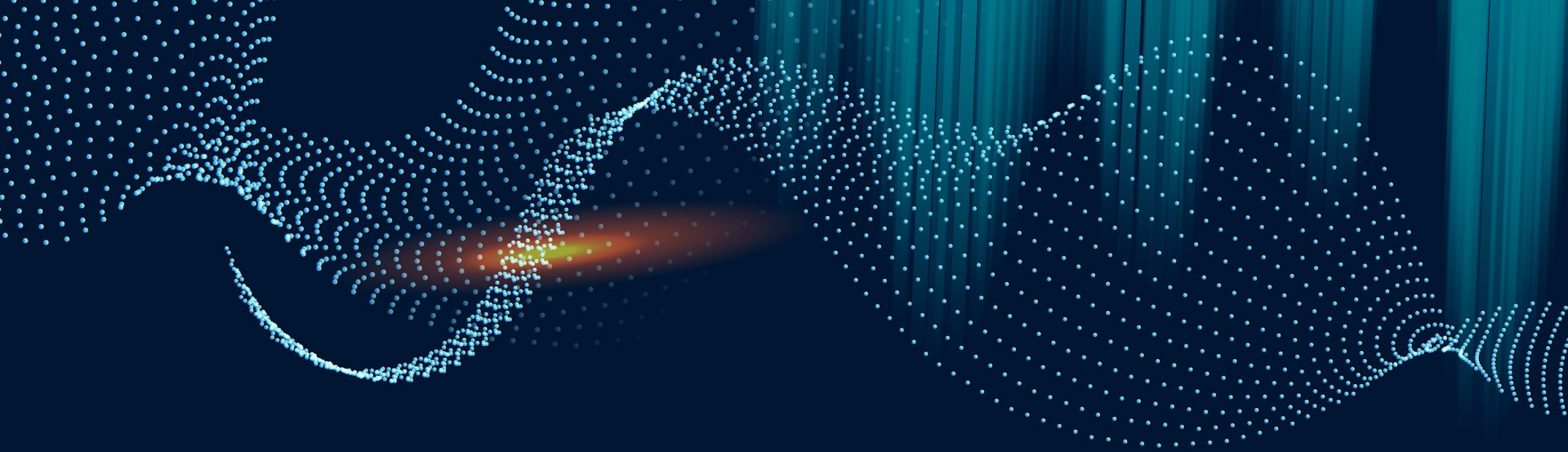
L’encapsulation en python

- En python on utilise le ‘_’ avant un attribut comme convention pour déterminer qu’une variable est censé être privé.
- On peut aussi utiliser ‘__’ pour forcer la variable à être privé.

```
class Personne:
    def __init__(self, nom, prenom):
        self._nom = nom
        self.__prenom = prenom
```



Les propriété



02 Special methods

Someone said special?

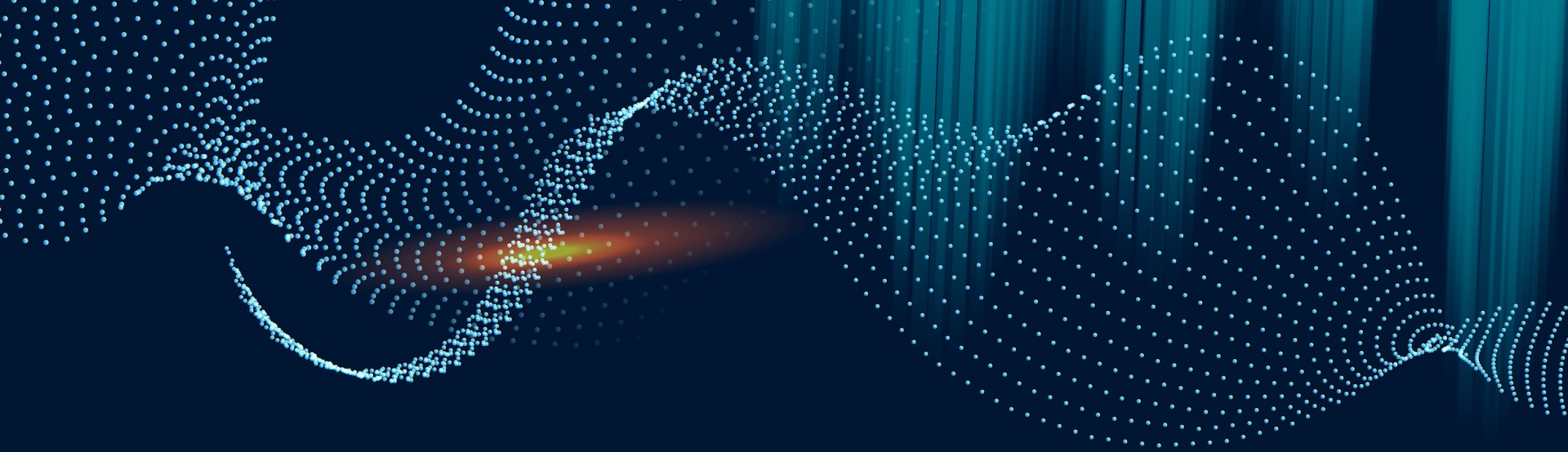
Les méthodes speciales

- Les méthodes spéciales sont des méthodes particulières que Python reconnaît et sait utiliser, dans certains contextes.
- Elles peuvent servir à indiquer quoi faire quand l'interpréteur se retrouve devant une expression comme objet1 + objet2 ou objet[index].
- Elles commencent et finissent par '__' (oui comme la méthode __init__)

Méthodes de Comparaisons et Opérations

object.**__add__**(*self*, *other*)
object.**__sub__**(*self*, *other*)
object.**__mul__**(*self*, *other*)
object.**__matmul__**(*self*, *other*)
object.**__truediv__**(*self*, *other*)
object.**__floordiv__**(*self*, *other*)
object.**__mod__**(*self*, *other*)
object.**__divmod__**(*self*, *other*)
object.**__pow__**(*self*, *other*[, *modulo*])
object.**__lshift__**(*self*, *other*)
object.**__rshift__**(*self*, *other*)
object.**__and__**(*self*, *other*)
object.**__xor__**(*self*, *other*)
object.**__or__**(*self*, *other*)

object.**__lt__**(*self*, *other*)
object.**__le__**(*self*, *other*)
object.**__eq__**(*self*, *other*)
object.**__ne__**(*self*, *other*)
object.**__gt__**(*self*, *other*)
object.**__ge__**(*self*, *other*)

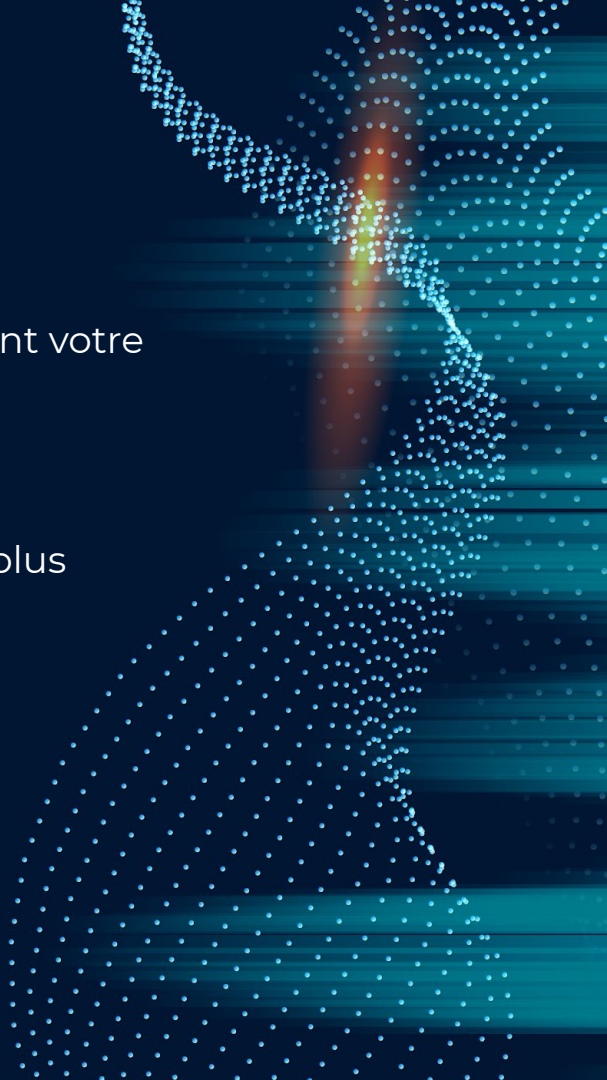


03 Exceptions

If you expect it, catch it!

Les Exceptions

- Il s’agit d’erreurs que peut rencontrer Python en exécutant votre programme.
- On peut intercepter ces dernières et les traiter.
- Il est très utile de les intercepter, mais il ne faut pas non plus aller vers l’extrême et intercepter toutes les exceptions.



Structure d’une Exception

```
try:
    nominateur = int(input())
    denominateur = int(input())
    result = nominateur / denominateur
except ZeroDivisionError as e:
    print("une erreur est survenue \n:{}".format(e))
    return None
except ValueError as e:
    print("une erreur est survenue \n:{}".format(e))
    return None
else:
    print("Aucune erreur, good job")
finally:
    print("Se code s'execute a la fin dans tous les cas")
```

Le mot clé “raise”

- Utiliser pour lever une exception à l’intérieur du code

```
def test_age(age):
    if age < 18:
        raise ValueError("vous ne pouvez pas continuer sur se site")
```

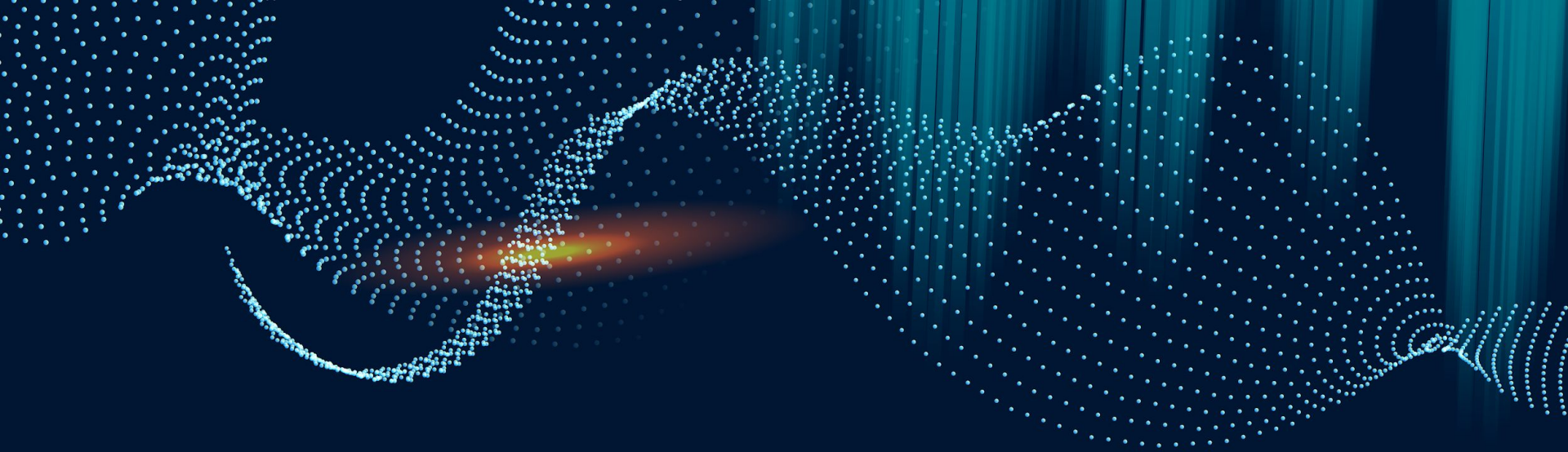
Créé vos propres Exceptions

```
class MilkBeforeCerealError(Exeptions):

    def __init__(self, message):
        self.message = message

    def __str__(self):
        return self.message
```

```
raise MilkBeforeCerealError("WHAAATT!! NO YOU CAN'T JUST PUT THE MILK BEFORE THE CEREALS")
```



04 Return to Lists

There is more to the lists than it seems

List Compréhensions

```python
# list comprehension

my_list = [i for i in range(20)]

number_list = [5, 10, 4, 52, 7, 81, 23]

even_list = [i for i in number_list if i % 2 == 0]

even_list_strings = [str(i) for i in number_list if i % 2 == 0]
```

Function with an infinity of parameters

```
def function(*args):
    for value in args:
        print(value)

function(5, 1, 't')
```

Function with an infinity of named parameters

```
def function(**named_args):
    for key in named_args:
        print("arg name : {}, value : {}".format(key , named_args[key]))

function(orange = 2, pomme=3, bannanes=10)
```

Petit Instant Promo :
TGM demain 20h

Prisons, Puisements or
réhabilitation
+
Movie night after that

THANKS!

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

By OpenMindsClub