

# Classes

UE12 P24 - Python

# Classes

Vous pouvez définir vos propres types d'objets :

```
1 class User:
2     def __init__(self, name: str, age: int):
3         self.name = name
4         self.age = age
5
6 def greet_user(user: User) -> None:
7     print(f'Salut {user.name} !')
8
9 alice = User('Alice', 43)
10 greet_user(alice)
```

Salut Alice !

# Classes, attributs, méthodes

Une classe a des attributs et des méthodes :

```
1 class User:
2     def __init__(self, name: str, age: int):
3         self.name = name # Un attribut
4         self.age = age # Un 2e attribut
5
6     def say_hi(self) -> None: # Une méthode
7         print(f'Hello, my name is {self.name}!')
8
9 bob = User('Bob', 22)
10 print('attributes:', bob.name, bob.age)
11 bob.say_hi()
```

attributes: Bob 22

Hello, my name is Bob!

# Attributs d'objet et de classe

Un attribut peut être lié au concept (la **classe**), ou à un exemple de ce concept (un **objet**, une **instance**).

```
1 class Message:
2     MAX_LENGTH = 140
3
4     def __init__(self, content: str):
5         self.content = content
6
7 print('A class attribute:', Message.MAX_LENGTH)
8 print('An instance attribute:', Message('Hi!').content)
```

```
A class attribute: 140
```

```
An instance attribute: Hi!
```

# Méthodes et fonctions de classe

Il en va de même pour les fonctions. Elles peuvent être liées à l'**instance** (on parle de **méthode**) ou à la **classe**.

```
1  class Message:
2      def get_possible_types():
3          return ['private', 'public']
4
5      def __init__(self, content: str, is_public: bool):
6          self.content = content
7          self.is_public = is_public
8
9      def get_type(self):
10         return 'public' if self.is_public else 'private'
11
12 print('A class function:', Message.get_possible_types())
13 print('A method:', Message('Hi!', False).get_type())
```

A class function: ['private', 'public']

A method: private

La différence est leur 1er paramètre : **self** pour une méthode.

# Classes et conventions

Par convention, un attribut ou une méthode dont le nom commence par un `_` ne doit pas être utilisé hors de la classe.

```
1 class Server:
2     def __init__(self, users: 'list[dict]', channels: 'list[dict]'):
3         self._users = users
4         self.channels = channels
5
6     def get_users(self):
7         return self._users
```

# Méthodes magiques - 1

Les méthodes qui commencent et terminent par `__` sont automatiquement appelées par Python dans certains cas.

```
1 class Channel:
2     def __init__(self, id: int, name: str):
3         self.id = id
4         self.name = name
```

# Méthodes magiques - 2

Une classe nouvellement créée n'est pas très pratique :

```
1 class Channel:
2     def __init__(self, id: int, name: str):
3         self.id = id
4         self.name = name
5
6 print(Channel(1, 'First channel'))
```

<\_\_main\_\_.Channel object at 0x0000022D25CECC20>

Heureusement, une méthode magique permet d'améliorer ça :

```
1 class Channel:
2     def __init__(self, id: int, name: str):
3         self.id = id
4         self.name = name
5
6     def __repr__(self) -> str:
7         return f'Channel(id={self.id}, name={self.name})'
8
9 print(Channel(1, 'First channel'))
```

Channel(id=1, name=First channel)



# Méthodes magiques - 3

Les méthodes magiques permettent d'implémenter toutes sortes de comportements :

```
1 class Channel:
2     def __init__(self, id: int, name: str):
3         self.id = id
4         self.name = name
5
6 print(Channel(1, 'First channel') == Channel(1, 'Changed name'))
```

False

```
1 class Channel:
2     def __init__(self, id: int, name: str):
3         self.id = id
4         self.name = name
5
6     def __eq__(self, other) -> bool:
7         return self.id == other.id
8
9 print(Channel(1, 'First channel') == Channel(1, 'Changed name'))
```

True

# Classes - Le minimum



Tip

Lorsque vous développez une nouvelle classe, implémentez toujours au moins `__init__` et `__repr__`.

```
1 class Channel:
2     def __init__(self, id: int, name: str):
3         self.id = id
4         self.name = name
5
6     def __repr__(self) -> str:
7         return f'Channel(id={self.id}, name={self.name})'
```

# Dataclasses

Le code précédent est un peu répétitif : le nom de chaque attribut apparaît 4 fois !

Lorsqu'une classe est essentiellement un conteneur d'attributs, Python nous permet d'aller plus vite avec les **dataclasses**.

```
1 from dataclasses import dataclass
2
3 @dataclass
4 class Channel:
5     id: int
6     name: str
7
8 print(Channel(1, 'Town square'))
```

```
Channel(id=1, name='Town square')
```

# Héritage

Des propriétés partagées entre 2 classes peuvent être mutualisées en une **classe mère** commune dont les **classes filles** héritent.

# Héritage

```
1 class Animal:
2     NUMBER_OF_LEGS: int
3     def move(self):
4         print(f"I'm moving thanks to my {self.NUMBER_OF_LEGS} legs!")
5
6 class Cat(Animal):
7     NUMBER_OF_LEGS = 4
8     def meow(self):
9         print('Meow')
10
11 class Snake(Animal):
12     NUMBER_OF_LEGS = 0
13
14 cat = Cat()
15 cat.move()
```

I'm moving thanks to my 4 legs!

```
1 cat.meow()
```

Meow

```
1 Snake().move()
```

I'm moving thanks to my 0 legs!

# Héritage et override

Si la fonction de la **classe mère** ne nous plaît pas, on peut la redéfinir dans la **classe fille** :

```
1 from typing import override
2
3 class Animal:
4     NUMBER_OF_LEGS: int
5     def move(self):
6         print(f"I'm moving thanks to my {self.NUMBER_OF_LEGS} legs!")
7 class Bird(Animal):
8     @override
9     def move(self):
10         print("I'm flying!")
11
12 Bird().move()
```

I'm flying!



Note

- `override` n'est disponible que dans les versions récentes de Python.
- son usage n'est pas obligatoire

# Héritage et **super**

Lorsqu'on redéfinit une méthode dans une classe fille, on peut utiliser la méthode de la classe mère grâce à **super** :

```
1  from typing import override
2
3  class Animal:
4      NUMBER_OF_LEGS: int
5      def move(self):
6          print(f"I'm moving thanks to my {self.NUMBER_OF_LEGS} legs!")
7  class Bird(Animal):
8      NUMBER_OF_LEGS = 2
9
10     @override
11     def move(self):
12         super().move()
13         print("I'm flying!")
14
15  Bird().move()
```

I'm moving thanks to my 2 legs!

I'm flying!