Exemples : Héritage et Polymorphisme en Python

A. Djebabla (exemple pédagogique)

But du document

Ce document montre trois concepts-clés orientés objet en Python :
— **Héritage** (subclassing),
— **Polymorphisme** via redéfinition de méthodes,
— **Polymorphisme via interface** (utilisation d'une classe abstraite avec `abc`).
Les exemples sont prêts à être copiés dans Overleaf (ou un fichier `.py`).

1 Héritage simple

Explication : une classe `Animal` comportant une méthode `parler`, puis deux sous-classes `Chien` et `Chat` qui héritent des attributs et méthodes de `Animal`.

```
class Animal:
def **init**(self, nom):
self.nom = nom

‘‘‘
def parler(self):
    return "..."
‘‘‘

class Chien(Animal):
def **init**(self, nom, race):
super().**init**(nom)
self.race = race

‘‘‘
def parler(self):  # red finition
    return "Woof!"
‘‘‘

class Chat(Animal):
def parler(self):
return "Miaou!"

# Utilisation

rex = Chien('Rex', 'Labrador')
minou = Chat('Minou')
print(rex.nom, rex.race, rex.parler())  # Rex Labrador Woof!
```

```
print(minou.nom, minou.parler())       # Minou Miaou!
```

2 Polymorphisme par redéfinition (overriding)

Le polymorphisme permet d'appeler la même méthode sur des objets de classes différentes. Exemple : une fonction `faire_parler` qui accepte un `Animal`.

```
def faire_parler(animal):
print(animal.parler())

# M me code que ci-dessus (Chien, Chat)

faire_parler(rex)    # appelle Chien.parler -> "Woof!"
faire_parler(minou) # appelle Chat.parler  -> "Miaou!"
```

Ici, `faire_parler` ne connaît pas la classe concrète ; elle invoque la méthode `parler` de l'objet reçu. C'est le

3 Polymorphisme via une "interface" (classe abstraite)

Python n'a pas d'interface au sens Java, mais on peut utiliser le module abc pour définir une classe abstraite qui impose des méthodes.

```
from abc import ABC, abstractmethod

class Volant(ABC):
@abstractmethod
def voler(self):
pass

class Avion(Volant):
def voler(self):
return "L'avion d colle et vole."

class Canard(Volant):
def voler(self):
return "Le canard bat des ailes et vole."

# Tentative d'instanciation de Volant l vera une erreur

# v = Volant()  # TypeError: Can't instantiate abstract class
   Volant with abstract methods voler

objets_volants = [Avion(), Canard()]
for o in objets_volants:
print(o.voler())
```

Remarques finales

- Le polymorphisme rend le code plus flexible et facilite l'extension : on peut ajouter de nouvelles classes sans modifier les fonctions qui les utilisent.
- L'utilisation d'une classe abstraite permet d'imposer un contrat (comme une interface) entre classes.
- Pensez à documenter vos classes et à utiliser des tests unitaires pour vérifier les comportements polymorphes.

--

Fichier prêt pour Overleaf. Si vous voulez, je peux :
- ajouter des diagrammes UML simples,
- fournir des exercices pour les étudiants,
- convertir en présentation Beamer.