2N6 Programmation 2

avec python™

retour

Retour sur les notions de programmation objet

Le mot-clef **class** signifie que nous allons faire une classe (nécessaire)

Les noms des classes commencent toutes par une majuscule par convention

retour

La méthode "magic" ou dunder **__init__** est toujours appelée lorsqu'on crée un nouvel objet. Il s'agit du constructeur de la **classe**

```
class Chaise:
    def __init__(self, largeur, profondeur, hauteur):
        self.largeur = largeur
        self.profondeur = profondeur
        self.hauteur = hauteur

    def ajuster_hauteur(self, nouvelle_hauteur):
        self.hauteur = nouvelle_hauteur
```

Ici, toutes nos méthodes commencent par **self,** qui référencie l'objet qui est créé.

Valeurs par défauts et instanciations

retour

```python
class Chaise:
    def __init__(self, largeur = 40, profondeur = 40, hauteur = 110):
        self.largeur = largeur
        self.profondeur = profondeur
        self.hauteur = hauteur
```

```python
chaise_3 = Chaise()
chaise_4 = Chaise(35, 45, 95)

print("chaise 3 :")
print(chaise_3.largeur, chaise_3.profondeur, chaise_3.hauteur)

print("chaise 4 :")
print(chaise_4.largeur, chaise_4.profondeur, chaise_4.hauteur)
```

PROBLÈMES SORTIE TERMINAL

```
chaise 3 :
40 40 110
chaise 4 :
35 45 95
```

Variables de classes

retour

```
class Employe:
    liste_employe = []
    next_ID = 1000

    def __init__(self,nom,prenom) -> None:
        self.nom = nom
        self.prenom = prenom
        self.ID = Employe.next_ID

        Employe.next_ID += 1
        Employe.liste_employe.append(self)
```

Variables appartenant à la classe Employe

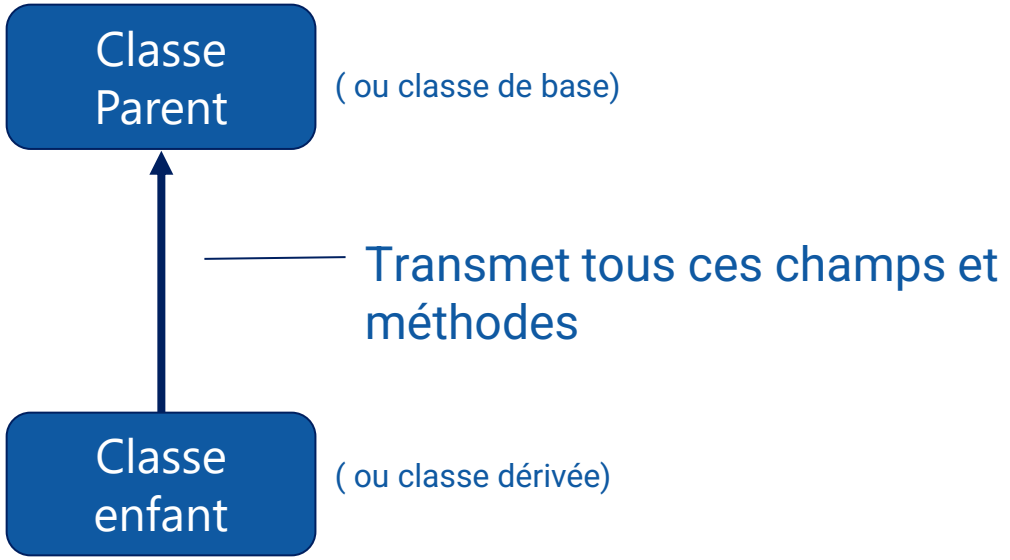
Variables appartenant à l'objet créé à partir de la classe Employe

Ici on modifie les variables de classes. Toutes les instanciations accèderont aux même variables de classe

L'héritage des classes

retour

- Permet de définir une classe à partir d'une classe déjà existante

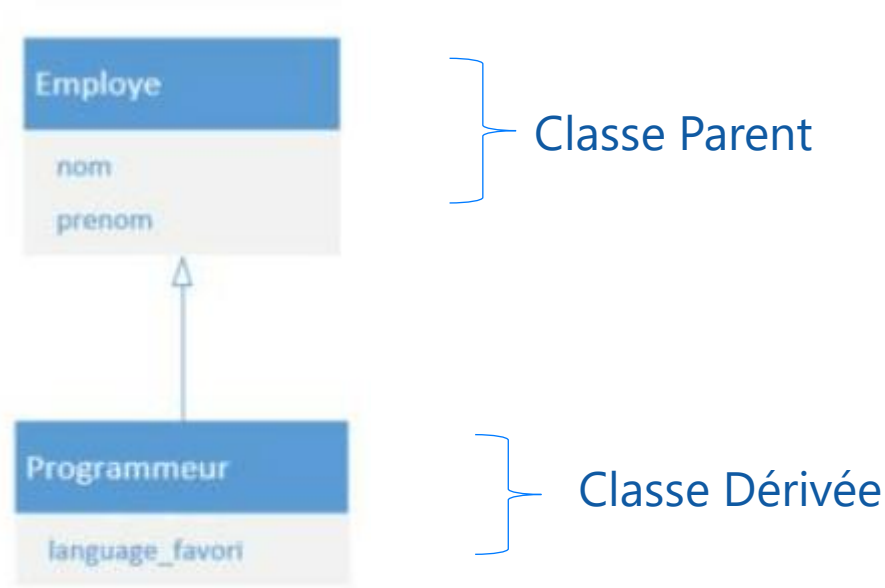


- Permet d'hériter des champs et des méthodes de la classe parent.

Modélisation UML

retour

- L'héritage est illustré ainsi dans UML



Un Programmeur EST un Employe

Héritage

retour

```
class Programmeur(Employe):
    def __init__(self, nom, prenom,language_favori) -> None:
        super().__init__(nom, prenom)
        self.language_favori = language_favori
```

super() fait référence au parent
Employe. On utilise sa méthode __init__()
pour lui passer les valeurs de ses propriétés.

Méthodes de classes

retour

```python
class Employe:
    nb_employes = 0
    base_augmentation = 1.04

    def __init__(self, prenom, nom, salaire):
        self.prenom = prenom
        self.nom = nom
        self.salaire = salaire
        self.courriel = prenom + '.' + nom +'@gmail.com'
        Employe.nb_employes +=1

    def nom_complet(self):
        return '{} {}'.format(self.prenom, self.nom)

    def donner_augmentation(self):
        self.salaire = int(self.salaire * self.base_augmentation)
        # nous utilisons self car l'augmentation de base pourrait varier selon l'employé instancié

    @classmethod
    def from_string(cls, emp_str):
        "Constructeur pour créer un employé à partir d'une chaine séparée avec un '-'"
        prenom, nom, salaire = emp_str.split('-')
        return cls(prenom, nom, salaire)
```

Decorator

Méthode de classe

Fait référence à la classe

Méthodes de classes

retour

- On fait des méthodes de classe quand la méthode ne fait pas référence aux objets instanciés
- Cette méthode sera la même pour tous les objets instanciés.
- On doit utiliser le decorator **@classmethod** pour identifier que c'est une méthode de classe et pour pouvoir appeler la classe en utilisant **cls**

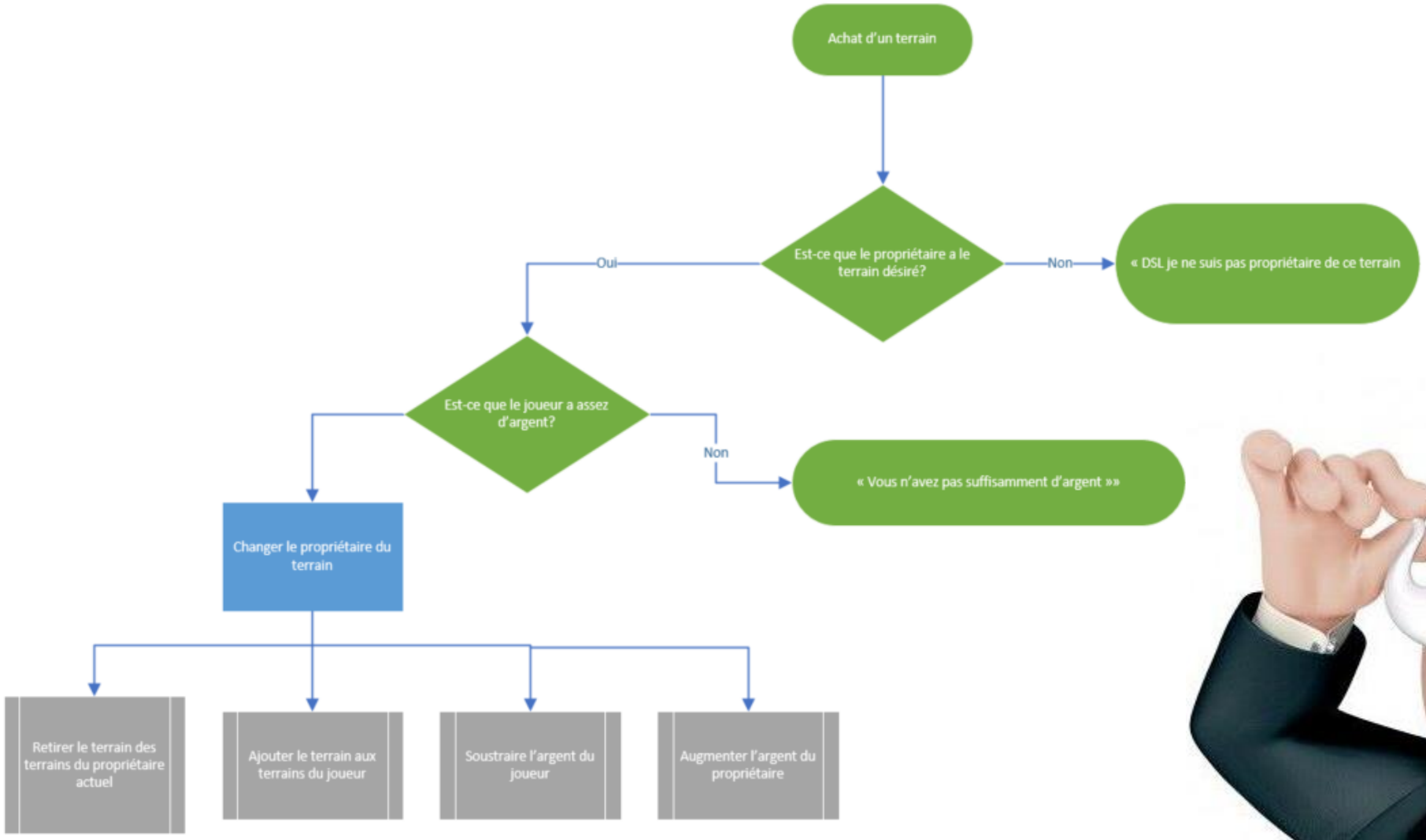
Ex : Monopoly - Pseudocode

retour

- Vérifier si le « propriétaire » a bel et bien le terrain qu'on veut acheter
  - Si oui
    - Vérifier si on a assez d'argent
      - Si oui
        - Retirer le terrain de la liste des terrains du propriétaire
        - Retirer le cash du joueur
        - Ajouter le cash dans le montant_cash du propriétaire
        - Ajouter le terrain de la liste des terrains du joueur
      - Si non
        - Écrire un message « Vous n'avez pas assez d'argent pour acheter le terrain X »
  - Si non
    - Écrire un message « Désolé, je ne suis pas propriétaire de ce terrain »

Ex : Monopoly – Diagramme de flux

retour



Aujourd'hui

- Deux exercices de consolidation :
  1. Ajouter des logiciels à des postes de travail
     a. Retour sur pseudocode
     b. Diagrammes de flux
     c. Héritage
     d. Variables de classes
     e. Méthodes de classes
     f. Instanciation d'objets

  2. Monopoly
     a. Création des classes
     b. Instanciation d'objets
     c. Relations d'un objet à N objets (1 joueur a une liste de terrains)