Cégep Limoilou

Formatif 3

420-SF3-RE

Objectifs : Réaliser des modèles objets propres en python

Exercice 1 : Simulation d'un Zoo

Objectif : Réalisation d'un modèle objet propre en python

Modélisez un système permettant de gérer différents types d'animaux dans un zoo. Vous devez utiliser :

- une **classe abstraite** Animal (avec abc.ABC)
- des **sous-classes concrètes** : Lion, Pingouin, Perroquet
- des méthodes polymorphes (parler, se_deplacer)
- des **attributs privés/protégés**, gérés avec **propriétés**
- des **constructeurs avec paramètres optionnels**

Contraintes techniques

1. Animal doit être une classe abstraite avec au moins deux méthodes abstraites :
    - `parler()`
    - `se_deplacer()`
2. Chaque animal aura des **attributs encapsulés** :
    - nom
    - âge
    - poids
3. Implémentez la méthode `__str__()` dans chaque sous-classe qui retourne une description de l'animal, ainsi que les méthodes abstraites.
4. Dans certaines sous-classes, créez un attribut optionnel dangereux (booléen, avec valeur par défaut dépendant du type d'animal) pour illustrer un **paramètre par défaut**. Pour fixer les idées le lion et le perroquet ont l'attribut mais pas le pingouin. Le lion est toujours dangereux , alors que le perroquet n'est pas dangereux par défaut mais peut l'être.
5. Profitez autant que possible des possibilités de l'attribut **@dataclass**. Essayez de faire un bilan de ce qu'il apporte

Exemple de programme principal

```
# Programme principal
if __name__ == "__main__":

    #Le lion est toujours dangereux pas d'attribut dans le constructeur
    lion = Lion("Simba", 5, 180.5)

    #Le pingouin n'a pas d'attrinut dangereux
    pingouin = Pingouin("Pingu", 3, 12.3)

    #Ce perroquet particulier est dangereux
    perroquet = Perroquet("Coco", 17, 3.3,dangereux=True)

    zoo = [lion, pingouin,perroquet]

    for animal in zoo:
        print(animal)
        animal.parler()
        animal.se_deplacer()
        print("----------------------")
```

Exemple de résultat d'exécution

```
Lion(_Animal__nom='Simba', _Animal__age=5, _Animal__poids=180.5,
dangereux=True)

Simba est muet

Simba court

----------------------

Pingouin(_Animal__nom='Pingu', _Animal__age=3,
_Animal__poids=12.3)

Pingu est muet

Pingu titube

----------------------

Perroquet(_Animal__nom='Coco', _Animal__age=17,
_Animal__poids=3.3, dangereux=True)

Coco repete

Coco vole

----------------------------------------
```

Exercice 2 : Modélisation d'une station de mesure

On souhaite modéliser un système qui gère différents **capteurs de mesure** dans un laboratoire.

Objectifs :

- Mettre en place une **classe abstraite**.
- Implémenter plusieurs **sous-classes spécialisées.**
- Appliquer le **polymorphisme** dans une simulation
- Utiliser **des liens** entre les objets

Classe abstraite Capteur (héritage + abstraction)

- Attributs communs : `id_capteur (str), unite (str).`
- L'Id du capteur est généré automatiquement lors de la construction, il est incrémenté à chaque nouveau capteur
- Méthodes abstraites :
    - `mesurer()` → retourne une valeur simulée (float).
- Méthode concrète :
    - `__str__ ()` → retourne une chaîne : "Capteur ID001 qui mesure en °C"

Sous-classes concrètes de Capteur :

- Thermometre (température en °C, valeur simulée entre -20 et 50).
- Barometre (pression en hPa, valeur simulée entre 950 et 1050).
- Luxmetre (lumière en lux, valeur simulée entre 0 et 100000).

Chaque sous-classe implémente `mesurer()` différemment (par exemple en utilisant random….(…)).

Classe StationMesure (lien entre objets)

- Attributs : `nom_station, capteurs.`
- Méthodes :
    - `ajouter_capteur(capteur)` → ajoute un capteur à la station.

- `effectuer_mesures()` → retourne un dictionnaire `{id_capteur: valeur}` en appelant `mesurer()` sur chaque capteur.

Programme principal

- Crée une station appelée "Station Limoilou".
- Ajoute un thermomètre, un baromètre et deux luxmètres.
- Affiche la description de tous les capteurs.
- Lance plusieurs séries de mesures et affiche les résultats.

Exemple d'exécution :

```
Station: Limoilou
      Capteur ID1 qui mesure en C
      Capteur ID2 qui mesure en hPa
      Capteur ID3 qui mesure en lux
      Capteur ID4 qui mesure en lux

----Mesure  0-----
      Mesure de Thermometre (ID1) : 21.66 C .
      Mesure de Barometre (ID2) : 234.55 hPa .
      Mesure de Luxmetre (ID3) : 133242.81 lux .
      Mesure de Luxmetre (ID4) : 7697.58 lux .
----Mesure  1-----
      Mesure de Thermometre (ID1) : -2.1 C .
      Mesure de Barometre (ID2) : 2286.29 hPa .
      Mesure de Luxmetre (ID3) : 166302.19 lux .
      Mesure de Luxmetre (ID4) : 59813.8 lux .
----Mesure  2-----
      Mesure de Thermometre (ID1) : 41.61 C .
      Mesure de Barometre (ID2) : 673.17 hPa .
      Mesure de Luxmetre (ID3) : 75881.25 lux .
      Mesure de Luxmetre (ID4) : 78122.54 lux .
----Mesure  3-----
      Mesure de Thermometre (ID1) : 29.06 C .
      Mesure de Barometre (ID2) : 399.93 hPa .
      Mesure de Luxmetre (ID3) : 47685.18 lux .
      Mesure de Luxmetre (ID4) : 1812.04 lux .
----Mesure  4-----
      Mesure de Thermometre (ID1) : -18.71 C .
      Mesure de Barometre (ID2) : 137.7 hPa .
      Mesure de Luxmetre (ID3) : 114748.26 lux .
      Mesure de Luxmetre (ID4) : 33483.57 lux .
```

Exercice 3: Sérialiser et désérialiser notre station de mesure

On souhaite stocker par sérialisation (pickle et json) notre station de mesure

Objectifs :

- Mettre en place la sérialisation/désérialisation pickle
- Mettre en place la sérialisation/désérialisation json

Pickle :

A partir du code de l'exercice 2 (ou du corrigé), implémenter la sérialisation de notre station dans un fichier pkl, puis lire ce fichier pkl pour charger la station avant l'exécution du programme principal.

Json :

- Implémenter les méthodes `to_dict()` dans les classes `StationMesure` et `Capteur` pour mettre en place la sérialisation json
- Implémenter la sérialisation dans un fichier à proprement parler.
- Implémenter une méthode de classe `from_dict(data)` dans les classes `StationMesure` et `Capteur` qui construit la hiérarchie d'objet complète d'une station.
  Indication, pour construire un objet à partir de l'objet cls:

```
def from_dict(cls, data):
    station = cls()#Appel du constructeur de l'objet basé sur la classe
```

- Implémenter la désérialisation depuis un fichier à proprement parler.