TP : Classes et instances

Instruction générale : hormis pour les exercices corrigés collectivement, vous ferez valider votre travail par l'enseignant.

1 Classes

1.1 Déclaration d'une classe

La déclaration d'une classe en Python se fait à l'aide du mot clé `class` comme dans l'exemple suivant.

Exemple :

```
class Personnage :
    pass
```

`class` est un mot clé Python au mettre titre que `def` et crée automatiquement une indentation.

Remarque :

- Écrire le nom d'une classe avec une majuscule n'est pas obligatoire du point de vue programmatique mais est une convention forte que nous respecterons par la suite.
- `pass` est un mot clé permettant de créer une classe, une fonction, une boucle ou un test et de le laisser vide sans générer d'erreur.

Une fois la classe créée, nous pouvons déclarer ses attributs. Ils peuvent être de tous types, même des instances de classes.

Exemple : ajoutons quelques attributs à notre classe. On remarquera que l'attribut `amis` est une liste qui se remplira d'autres personnages donc d'autres instances de classes.

```
class Personnage :

    nom = "Triss Merigold"
    sexe = "Femme"
    amis = []
```

1.2 Constructeur de classe `__init__()`

La syntaxe des exemples ci-dessus est valide mais très peu pratique. Devoir déclarer les attributs d'une instance à l'intérieur est très pénible et pose problème si l'on souhaite déclarer d'autres instances de la même classe. En effet, tel qu'écrit ci-dessus, nos attributs sont en fait des attributs de classe et non d'instance.

Les constructeurs permettent de contourner ces problèmes. Il s'agit de méthodes appelées lors de la création de de l'instance afin de passer les attributs souhaités en paramètres. En Python, les constructeurs sont définies à l'aide de la syntaxe `__init__(self, paramètres)`.

Les doubles underscores sont indispensables. La fonction `__init__()` est automatiquement appelée par Python lors de l'instanciation de la classe, il est donc impératif d'en respecter la syntaxe sous peine d'erreur.

1.3 Le paramètre `self`

Le paramètre `self` est un paramètre muet servant à faire référence à l'instance créée ou à construire. Il est indispensable et se trouve en argument de toutes les méthodes la classe ; il précède également tous ces attributs avec un point. Il pourrait être nommé autrement comme `instance`, `objet`, `truc` mais `self` est une convention forte qu'il convient de respecter. Pour rappel « self » signifie « le moi » ou « auto- » (utiliser en préfixe) ; il convient donc parfaitement pour désigner l'entité crée ou utilisée en elle-même.

1.4 Contructeur II, le retour

Donnons à présent un exemple d'utilisation de `__init__()`.

Exemple : reprenons notre classe `Personnage`.

```
class Personnage :

    def __init__(self, nom : str, sexe : str = None) :

        """ sexe est attendu sous la forme "f" ou "h"."""

        self.nom = nom
        self.sexe = sexe
        self.amis = []
```

Comme on peut le voir `self` est le premier argument de la méthode `__init__()` et permet de définir les attributs d'instance.

Il convient ici de distinguer `self.nom` et `nom`. `nom` est un argument passé en entrée de la fonction `__init__()` dont la valeur est affectée à l'attribut d'instance `self.nom`. Il est pratique de les nommer similairement du fort lien entre les deux, il ne faut cependant pas les confondre, il s'agit de deux choses différentes. On pourrait très bien choisir d'affecter `sexe` à `self.nom` et réciproquement avec le code ci-dessous.

```
self.nom = sexe
self.sexe = nom
```

Remarques :

- On peut observer que comme dans les fonctions usuelles, il est possible de donner une valeur par défaut à un attribut. C'est le cas ici avec `sexe = None` ; pour rappel, `None` permet d'affecter à une variable un contenu vide.
- Tous les attributs n'ont pas forcément leur avatar dans les paramètres de `__init__()`. C'est le cas ici de la liste `self.amis`

Exercice 1. Créer des classes pour chacun des objets suivants. On programmera des constructeurs initialisant leurs attributs (donnés en dessous). On pourra ajouter des attributs supplémentaires si on le souhaite.

1. Équipe de Quidditch.
   - Nom.
   - Joueurs.
   - Nombre de joueurs.
2. Joueur de Quidditch.
   - Nom.
   - Prénom.
   - Balai.
3. Balai volant.
   - Nom.
   - Couleur (marron par défaut).

2 Instanciation de classe

2.1 Création d'un objet

Pour créer un objet en Python, il suffit d'appeler le nom de la classe en donnant en arguments les paramètres attendus par le constructeur (sauf `self` qui fait référence à l'instance elle-même).

Exemple : instancions deux personnages à l'aide du code ci-dessous.

```
triss =  Personnage("Triss Merigold","f")
yennefer =  Personnage("Yennefer de Vengerberg")
```

Exercice 2. Créer un joueur et une équipe de Quidditch et un balais volant (on laissera la couleur par défaut).

2.2 Modifier un objet

Il est possible de modifier un objet déjà crée en l'appelant lui et ses attributs à l'aide de la syntaxe `nom_objet.attribut = valeur`.

Exemple : Lors de la création de Yennefer, nous n'avons pas préciser que celle-ci est une femme, elle a donc récupérer la valeur par défaut `None` pour l'attribut `sexe`. Corrigeons cela.

```
print(yennefer.sexe)
yennefer.sexe = "f"
print(yennefer.sexe)
```

On peut voir ici que le `self` de la classe a été remplacé par l'objet lui-même, ici le personnage de Yennefer. En effet, `self` ne sert à faire référence à une instance quelconque qu'au sein de la classe. Une fois l'instance créée et en dehors de celle-ci, on peut l'appeler directement par le nom de la variable la contenant.

Exercice 3. Modifier la couleur du balai crée dans l'exercice précédent.

3 Méthodes

3.1 Défintion et appel d'une méthode

La création d'une méthode dans une classe se fait de la même façon qu'une fonction : grâce au mot clé `def`.

Exemple : ajoutons des méthodes à notre classe `Personnage` afin que ceux crées puissent se lier d'amitié ou non.

```
class Personnage :

    def __init__(self, nom : str, sexe : str = None) :

        """ sexe est attendu sous la forme "f" ou "h"."""

        self.nom = nom
        self.sexe = sexe
        self.amis = []

    def etre_ami(self,personne) :

        # On ajoute la personne à la liste des amis.
        self.amis.append(personne)

        # On s'ajoute à la liste des amis de la personne.
        personne.amis.append(self)

        print(f"{self.nom} et {personne.nom} sont amis.")
```

```
    def ne_plus_etre_ami(self,personne) :

        # On retire la personne à la liste des amis.
        self.amis.remove(personne)

        # On se retire de la liste des amis de la personne.
        personne.amis.remove(self)

        print(f"{self.nom} et {personne.nom} ne sont plus amis.")

    def se_presenter(self) :

        print(f"Je suis {self.nom}.")

        if len(self.amis)>0 :
            print("Je suis ami avec :")
            for ami in self.amis :
                print(ami.nom)
```

On peut alors appeler ces méthodes avec la syntaxe `nom_objet.méthode(arguments)`.

Exemple : faisons en sorte que Triss et Yennefer soient amies puis se présentent.

```
triss.etre_ami(yennefer)
triss.se_presenter()
yennefer.se_presenter()
```

Exercice 4.

1. Créer des méthodes `ajouter_joueur`, `retirer_joueur` et `changer_joueur` pour la classe `Equipe` en tenant compte des maximums de joueurs à chaque poste.
2. Ajouter, retirer et changer des joueurs de l'équipe créée.

Exercice 5.

1. Créer des méthodes `acquerir_balai`, `perdre_balai` et `changer_balai` pour la classe `Joueur`. Il faudra s'assurer qu'un balai ne puisse appartenir qu'à une seule personne.
2. Ajouter, retirer et changer des balais d'un ou de plusieurs joueurs.

3.2 Méthodes spéciales

3.2.1 Le constructeur

Nous l'avons déjà vu, le constructeur est en réalité une méthode spéciale servant à l'instanciation de la classe et au fait de donner des valeurs aux attributs.

3.2.2 La méthode `__str__()`

La méthode `__str__()` permet « d'effectuer » un `print` » de l'instance de classe. Plus précisément, il s'agit d'une méthode renvoyant une chaîne de caractères qui sera appelée par défaut et dont le résultat sera renvoyé si l'on exécute la commande `print(instance)`. Si cette méthode n'est pas définie, cette commande renvoie une « adresse mémoire » où est stockée l'instance.

Exemple : la méthode `se_presenter()` de la classe `Personnage` pourrait être remplacée par la méthode `__str__()` ci-dessous.

```
def __str__(self) :

    presentation = f"Je suis {self.nom}."

    if len(self.amis) > 0 :
        presentation += " Je suis ami avec  :"
        for ami in self.amis :
            presentation += "\n" + ami.nom

    return presentation
```

Nous obtiendrons alors le même résultat que précédemment avec la commande `print(triss)` à la place de `triss.se_presenter()`.

Exercice 6.

1. Utiliser `print` sur une instance de `Joueur` créée précédemment afin que le joueur se présente son non, son prénom, son balai et son équipe.
2. Programmer une méthode `__str__()` pour les différentes classes déjà programmées.

3.2.3 Accesseurs et mutateurs

Les accesseurs et mutateurs ne seront pas plus développés dans ce TP ; on précisera toutefois qu'ils sont en général programmés sous les noms `get` et `set`.