R308 : Consolidation de la programmation
Chapitre 1b : Membres statiques

Table des matières

1 Attributs statiques et méthodes statiques

Un attribut statique est une variable qui est partagée par toutes les instances d'une même classe.
Une méthode statique (ou méthode de classe) est liée à une classe et non pas d'un objet en particulier

2 Attribut statique ou variable de classe en python :

Il est déclaré en dehors des méthodes.
Il est initialisé lors de la déclaration.
Règle de nommage : nom de la variable exclusivement en majuscule.
Dans les méthodes, la variable statique est accessible par l'intermédiaire du nom de la classe elle même

```
MaClasse.MA_VARIABLE_STATIQUE = 1
```

Il faudra implémenter une méthode statique pour manipuler la variable statique depuis l'extérieur de l'objet.

Méthode statique ou méthode de classe :

Elle est précédée du décorateur **@staticmethod**
Elle n'a pas de paramètre self !!! puisqu'elle n'est pas liée à un objet
L'appel d'une méthode statique se fait par l'intermédiaire de la classe

```
MaClasse.ma_methode_statique(param1, parm2)
```

2.1 Exemple : complexe et manipulation de complexes :

Nous allons créer deux classes :

Une classe « **Complexe** » qui représentera un complexe
Un attribut statique permettra de définir le contexte de travail et de modifier le format d'affichage ("math" sous la forme a + i.b et "elec" pour la forme [ro ; téta]).

Une classe « **MaClasseMath** » qui permettra de faire des opérations avec des complexes.
Les méthodes de la classe « MaClasseMath » seront statiques

Représentation UML :

Remarque : UNIQUEMENT DEUX ATTRIBUTS

```
Classe : Complexe
```

```
Attributs :
__reel: float
__img: float
Attributs statiques:
MODE_REPRESENTATION: str = "math" # ou "telecom"
```

```
Méthodes :
__init__(self):
set_rectangulaire(self,reel: float, img: float)->None:
set_polaire(self,ro: float, teta: float)->None:
get_reel(self)-> float :
get_img(self)-> float :
get_ro(self)-> float :
get_teta(self)-> float :
get(self)-> str:
Méthode statique :
change_mode(nouveau_mode: str)-> None :
```

Classe : MaClasseMath
Attributs :
Méthodes statiques multiplier(c1: Complexe, c2: Complexe)-> Complexe diviser(c1: Complexe, c2: Complexe)-> Complexe

2.2 Définition partielle de la classe Complexe (à compléter en TP)

```
class Complexe:
    # declaration de la variable statique
    MODE_REPRESENTATION: str = "math" # ou "elec"
    def __init__(self):
        # declaration des attributs
        self.__reel: float = 0.0
        self.__img: float = 0.0
    set_rectangulaire(self,reel: float, img: float)->None:
        ...
    set_polaire(self,ro: float, teta: float)->None:
        ...
    get_reel(self)-> float :
        ...
    get_img(self)-> float :
        ...
    get(self)-> str:
        ...
    @staticmethod
    def set_mode_affichage(mode: str)->None:
        if mode in ("math", "elec"):
            Complexe.MODE_REPRESENTATION = mode # modification de la variable statique
```

2.3 Exemple de manipulation de complexes :

```
if __name__ == "__main__":
    # déclarez 3 variables de type Complexe nommés z1, z2, z3
    z1: Complexe = None
    z2: Complexe = None
    z3: Complexe = None
    # instanciez les z1 et z2
    z1 = Complexe()
    z2 = Complexe()
    # changez le mode d'affichage
    Complexe.set_mode_affichage("elec")
    # multipliez z1 et z2 puis mettez le résultat dans z3
    z3 = MaClasseMath.multiplier(z1, z2)
    ...
```

2.4 La méthode __str__()

La méthode __str__() permet de retourner une chaîne de caractère qui décrit l'objet.
La méthode __str__() remplace avantageusement la méthode get()
La méthode __str()__ est appelée automatiquement à partir de l'objet … si elle existe.

1. Exemple d'une classe Personne :

```
class Personne:
      def __init__(self, nom:str, age:int):
            self.__nom: str = nom
            self.__age: int = age
      def __str__(self) -> str: # surdefinition de la méthode __str__()
            return f"{self.__nom} age : {str(self.__age)} ans"

if __name__ == "__main__":
      # déclaration des objet
      p1: Personne
      p2: Personne
      # instanciation
      p1 = Personne("phi", 58)
      p2 = Personne(age=12, nom="toto") # (1)
      # affichage
      print(p1) # (2)
      print(p2) # (2)
```

(1) : en nommant les attributs, il est possible de les préciser dans un ordre quelconque.
Affichage :

```
phil age : 58
toto age : 12
```
(2) : appel à la méthode __str__() de chaque objet

2.5 Affichage du dictionnaire par défaut d'un objet :

L'attribut __dict__ d'un objet permet de récupérer le dictionnaire associé à cet objet.

```
if __name__ == "__main__":
      # déclaration des objets
      p1: Personne
      p2: Personne
      # instanciation
      p1 = Personne("phi", 58)
      p2 = Personne(age=12, nom="toto")
      # affichage
      print(p1.__dict__)
      print(p2.__dict__)
```

Affichage :

```
{'_Personne__nom': 'phil', '_Personne__age': 58} (1)
{'_Personne__nom': 'toto', '_Personne__age': 12}
```
(1) le double underscore __ entraîne la réécrire du nom de l'attribut afin d'éviter les conflits de noms dans les sous-classes.

2.6 La fonction setattr() :

La méthode setattr() définit la valeur de l'attribut spécifié de l'objet spécifié.
La syntaxe est la suivante :

```
      setattr(object, attribute, value)
```

1. Exemple (a partir de la classe Personne précédente) :

```
setattr(p1, "_Personne__nom", "bob") # (1)
setattr(p2, "_Personne__age", 78)
```
(1) : ATTENTION : le nom de l'attribut doit correspondre à celui du dictionnaire de l'objet.

2.7 Instanciation d'un objet à partir d'un dictionnaire : première solution

Il est possible d'instancier un objet à partir d'un dictionnaire.
L'opérateur ** permet de « déballer » un dictionnaire.

```
# définition du dictionnaire
dict_p3 = { "nom" : "bob", "age" : 63} # (1)
# instanciation d'un personne à partir d'un dictionnaire
p3 = Personne(**dict_p3)
```
(1) : ATTENTION : dans ce cas, les clés correspondent aux paramètres du constructeur

2.8 Instanciation à partir d'un dictionnaire : seconde solution

Il peut être intéressant d'affecter les attributs directement à partir d'un dictionnaire.
- le dictionnaire peut être lu dans un fichier.
- le nombre d'attributs peut être important

1. Exemple :

```
class Employe: # création des attributs a partir d'un dictionnaire
      def __init__(self, dictionnaire: dict):
            # definition des attributs
            self.__nom: str
            self.__age : int
            self.__ pays : str
            # affectation des attributs
            for cle, valeur in dictionnaire.items():
                  setattr(self, cle, valeur)
```

```
p4: Employe

dict_p4 = {
    "_Employe__nom": "etienne",
    "_Employe__age": 34,
    "_Employe__pays": "france",
    "_Employe__anniversaire": "1988-12-27",
    "_Employe__id": 227417393,
    "_Employe__internet": False,
    "_Employe__langue": "francais",
    "_Employe__genre": "homme"
}
p4 = Employe(dict_p4)
```