NSI Donnée Structurée : les classes

J. Courtois

7 septembre 2023

Motivation

- Nous avons vu jusque là des données de type simple (int, float,boolean, string, . . .) et quelques types plus structurés (n-upplets et list)
- On aimerait être capable de créer ses propres structures de données en fonction d'un problème donné.
- Par exemple, un type Personne qui organise les informations relatives à une personne, ie. un objet qui contient 3 informations : le nom, la date de naissance, la ville d'habitation.
- Nous allons utiliser pour cela la notion de classe

Déclaration

```
class NomClasse:
        Délaration1
        Délaration2
        .
        .
        .
        DélarationN
```

Le mot-clef class
permet de délarer l'existence d'un nouveau type d'objet nommé NomClasse

le constructeur : __init__

```
def __init__(self,var1,var2,. . . ,varN ):
    self.champs1 = val1
    self.champs2 = val2
    ....
    self.champsN = valN
    SequeneInstrution
```

C'est cette fonction qui précise les informations (attributs ou champs) contenues dans les objets

Le mot-clef self désigne une instance de ce type d'objet et permet de définir des valeurs propres pour cet objet :

Exemple

```
class Personne:
    def __init__(self, nom, dateNaiss, villeNaiss):
        self.nom = nom
        self.date = dateNaiss
        self.ville = villeNaiss
```

Créer un objet

```
X = NomClasse(var1,var2,...,varN)
```

```
class Personne:
    def __init__(self, nom, dateNaiss, villeNaiss):
        self.nom = nom
        self.date = dateNaiss
        self.ville = villeNaiss
p=Personne("Fabien", 1978, "Paris")
```

Récupérer les valeurs contenues dans les champs des objets

```
X = NomClasse(var1,var2,...,varN)
X.champs1
X.champs2
...
X.champsN
```

On récupère les champs en passant par l'instance

```
class Personne:
    def __init__(self, nom, dateNaiss, villeNaiss):
        self.nom = nom
        self.date = dateNaiss
        self.ville = villeNaiss
p=Personne("Fabien", 1978, "Paris")
print(p.nom)
```

Les champs de la classe

```
class NomClasse:
    varClass1=val1
    ...
    varClassN=valN
```

On peut définir des champs de classe, ie. partagés par tous les objets.

```
class Personne:
    population=0
    def __init__(self, nom, dateNaiss, villeNaiss):
        self.nom = nom
        self.date = dateNaiss
        self.ville = villeNaiss
        Personne.population=Personne.population+1
p=Personne("Fabien", 1978, "Paris")
print(Personne.population,p.population)
```

Valeurs par défaut

On peut, lors de la définition du constructeur, préciser des valeurs par défaut. Exemple :

```
class Personne:
    population=0
    def __init__(self, nom, dateNaiss, villeNaiss="Paris"):
        self.nom = nom
        self.date = dateNaiss
        self.ville = villeNaiss
        Personne.population=Personne.population+1
p=Personne("Fabien", 1978)
```

Valeurs par défaut

Attention il ne faut pas faire cela dans le cas d'une liste. Exemple :

```
class Personne:
    population=0
    def __init__(self, nom, dateNaiss, taillePoids=[]):
        self.nom = nom
        self.date = dateNaiss
        self.ville = villeNaiss
        self.taillePoids=taillePoids
        Personne.population=Personne.population+1
p=Personne("Fabien", 1978)
```

Les fonctions : première variante

On peut maintenant définir des fonctions qui prennent en argument des variables d'un nouveau type.

Même syntaxe qu'avant. Par exemple :

```
def habiteParis(p):
    return (p.ville=="Paris")
```

L'appel à cette nouvelle fonction se fait comme auparavant :

```
p=Personne("Fabien", 1978, "Paris")
print(habiteParis(p))
```

Les fonctions : deuxième variante

On peut aussi définir une fonction (on parle alors de méthode) comme étant propre aux objets.
La définition se fait alors dans la classe et utilise le mot-clef self pour désigner l'instance :

```
class Personne:
    ...
    def habiteParis(self):
        return (self.ville=="Paris")
```

L'appel à cette nouvelle méthode se fait en passant par l'objet lui-même :

```
p=Personne("Fabien", 1978, "Paris")
print(p.habiteParis())
```

Pour faciliter l'intégration à d'autres codes

Une série de fonctions (méthodes) ont un sens précis pour les classes, ce qui facilitent l'intégration du nouvel objet avec d'autre programme.

On a déjà vu le cas de la fonction __init__ mais il y en a d'autres :

- __str__ : pour permettre la conversion en chaîne de caratère de l'objet (appelée par print).
- __del__ : pour détruire l'objet.
- __eq__ : pour tester l'égalité structurelle entre deux objets de la classe.
- __lt__, __le__, __ge__ , . . . : pour ordonner les objets entre eux ;

Un exemple complet

```
class Personne:
    population=0
    def __init__(self, nom, dateNaiss, villeNaiss="Paris"):
        self.nom = nom
        self.date = dateNaiss
        self.ville = villeNaiss
        Personne.population=Personne.population+1
    def __str__(self):
        return "Nom : " + self.nom +" Date de naissance : "\
        + str(self.date) + "Ville : " + self.ville
    def __eq__(self,other):
        return (self.nom==other.nom and self.date==other.date \
        and self.ville==other.ville)
    def Affiche_nom(self):
        return self.nom
.....
```

Un exemple complet

```
class Personne:
.....
    def date(self):
        return self.date
    def ville(self):
        return self.ville
    def habiteVille(self, ville):
        return self.ville==ville

p1=Personne("Fabien",1978,"Paris")
p2=Personne("Fabien",1978)
print(p2)
print(p1==p2)
print(Personne.nom(p2),Personne.date(p2),Personne.ville(p2))
print(p2.habiteVille("Paris"))
```

Remarque : Valeurs par défaut

Attention il ne faut pas faire cela dans le cas d'une liste. Exemple :

```
class Personne:
    def __init__(self, nom, dateNaiss, taillePoids=[0,0]):
        self.nom = nom
        self.date = dateNaiss
        self.taillePoids=taillePoids
    def setTaillepoid(self,taille,poids):
        self.taillePoids[0]=taille
        self.taillePoids[1]=poids
    def __str__(self):
        a=self.nom+ " taille:"+str(self.taillePoids[0])+" poids: "+str(
        return a
p1=Personne("Fabien", 1978)
p1.setTaillepoid(1.83,65)
p2=Personne("Courtois", 1968)
print(p2)
```