Cours en distanciel G2 - Terminale NSI :Structures de données
semaine du 9 novembre

partie I

I) Vocabulaire : Commençons par un peu de vocabulaire .. en gras celui à connaître

Une **structure de données** est un ensemble d'informations et de relations .

L'interface est la spécification d'une structure de données, elle décrit les actions , les relations , les préconditions ...

Le TAD est un Type Abstrait de Données : on l'appelle ainsi car il correspond à un descriptif , un cahier des charges, mais pas à des données concrètes, réelles .. pour l'instant.
Abstrait <-> Concret.

Construction , Création et initialisation: le moment ou on passe du type abstrait au type concret.
Consultation : Permet de fournir un attribut du type concret.
Modification / Évaluation : Modification ou création d'un type concret

Type immuable : ne peut changer d'état , par exemple un entier n'est pas un flottant .
Type mutable: on peut changer l'état après la création , par exemple une liste

Implémenter c'est mettre en œuvre

La règle d'OR :
Ne pas s'intéresser à ce que fait le système (aspect fonctionnel) mais à ce qui constitue le système (aspect structurel).

Un exemple : on désire créer le TAD Couleur sur le modèle RVB/ 3 octets

interface :

Le type du TAD : Couleur

Que faut -il pour la création - De quelles informations à besoin le constructeur ?

Pour créer une couleur, il faut passer trois valeurs entières (on n'utilise pas de tuple, car une fois définié, on ne peut pas changer les valeurs d'un tuple

Compléter en indiquant les types de paramètres **les opérations** suivantes pour TAD Couleur :
Entrée -> Sortie

getRouge : Couleur ->Entier

Commentaires : cette fonction retourne la composante rouge de la couleur, en entrée on passe la couleur, en sortie la composante rouge

setRouge : Couleur x Entier -> Couleur

cette fonction change la composante rouge de l'objet couleur

mélanger :Couleur x Couleur x Réel (taux) -> Couleur

Un exemple **d'implémentation** (d'écriture) du code de la classe illustrant l'objet TAD Couleur:

```
2  class Couleur:
3      R: int
4      V :int
5      B :int
6
7      def __init__(self,rouge:int,vert:int,bleu:int):
8          self.R,self.V,self.B = rouge,vert,bleu
```

__init__ est le constructeur

La classe est définie lignes 2 à 8.

Ligne 11, on crée une **instance** de la classe Couleur nommée UneCouleur.

```
11  UneCouleur = Couleur(255,0,0)
```

La classe Couleur est le TAD, (**abstrait**, c'est un modèle vide) et l'objet UneCouleur est **concret**.

Surcharge d'un opérateur :

Heureusement que l'on ne doit pas tout définir dans un langage de programmation !

Imprimons notre Couleur UneCouleur :

ligne 12, on ajoute:

```
11  UneCouleur = Couleur(255,0,0)
12  print(UneCouleur)
```

On constate que l'affichage n'est pas très lisible **(déjà vu**).

```
<__main__.Couleur object at 0x7f4a65a0fba8>
```

Quand on utilise print pour l'objet UneCouleur, on appelle en réalité un fonction prédéfinie nommée str. Nous allons la réécrire (cela s'appelle **surcharger** une fonction)

```
2  class Couleur:
3      R: int
4      V :int
5      B :int
6
7      def __init__(self,rouge:int,vert:int,bleu:int):
8          self.R,self.V,self.B = rouge,vert,bleu
9
10     def __str__(self):
11         return "Rouge:" + str(self.R) + " Vert: " + str(self.V) + " Bleu:" + str(self.B)
```

La sortie est bien plus lisible ! A noter que la fonction __str__ , n'affiche rien, elle retourne une chaîne de caractère qui sera affichée.

On peut instancier plusieurs objets :

```
UneCouleur = Couleur(255,0,0)
UneDeuxiemeCouleur = Couleur(0,255,0)
print(UneCouleur)
print(UneDeuxiemeCouleur)
```

```
Rouge:255 Vert: 0 Bleu:0
Rouge:0 Vert: 255 Bleu:0
```

En réalité les lignes 2, 3 et 4 " ne servent à rien ":

```
class Couleur:

    def __init__(self,rouge:int,vert:int,bleu:int):
        self.R,self.V,self.B = rouge,vert,bleu

    def __str__(self):
        return "Rouge:" + str(self.R) + " Vert: " + str(self.V) + " Bleu:" + str(self.B)
```

C'est juste plus "élégant" et cela apporte un peu plus de lisibilité au code, surtout avec un petit commentaire

```
2  class Couleur:
3
4      #Pour la composante rouge
5      R :int
6      #Pour la composante verte
7      V :int
8      #Pour la coposante Bleue
9      B :int
10
11
12     def __init__(self,rouge:int,vert:int,bleu:int):
13         self.R,self.V,self.B = rouge,vert,bleu
14
15     def __str__(self):
16         return "Rouge:" + str(self.R) + " Vert: " + str(self.V) + " Bleu:" + str(self.B)
17
18 UneCouleur = Couleur(255,0,0)
19 UneDeuxiemeCouleur = Couleur(0,255,0)
20 print(UneCouleur)
21 print(UneDeuxiemeCouleur)
```

La portée des variables :

Dans le code précédent, les variables R,V et B sont appelées variables d'instance, leur protée est limitée à l'instance crée (l'objet):

```
23  print(UneCouleur.R)
24  print(UneDeuxiemeCouleur.R)
```

```
255
0
```

A chacun son rouge quoi !

Maintenant, modifions le code ainsi :

```
1
2   class Couleur:
3
4       #Pour la composante rouge
5       R :int
6       #Pour la composante verte
7       V :int
8       #Pour la coposante Bleue
9       B :int
10      
11      image = "RVB"
12      
13      def __init__(self,rouge:int,vert:int,bleu:int):
14          self.R,self.V,self.B = rouge,vert,bleu
15          
16      def __str__(self):
17          return "Rouge:" + str(self.R) + " Vert: " + str(self.V) + " Bleu:" + str(self.B)
18
19  UneCouleur = Couleur(255,0,0)
20  UneDeuxiemeCouleur = Couleur(0,255,0)
21  #print(UneCouleur)
22  #print(UneDeuxiemeCouleur)
23
24  print(UneCouleur.image)
25  print(UneDeuxiemeCouleur.image)
26
27  UneCouleur.image ='EchelleDeGris'
28  print(UneDeuxiemeCouleur.image)
29
30  UneCouleur.V = 200
31  print(UneCouleur)
```

Voici la sortie :

```
RVB
RVB
RVB
Rouge:255 Vert: 200 Bleu:0
```

Voici ce que l'on peut lire dans la documentation python :
En général, les variables d'instance stockent des informations relatives à chaque instance alors que les variables de classe servent à stocker les attributs et méthodes communes à toutes les instances de la classe.

Q1) Quelles sont sur notre exemple les variables de classe et les variables d'instance ?

Effets indésirables ?

ligne 30, on affecte directement la couleur V (vert) à partir du code . C'est simple, c'est clair . Maintenant, imaginez qu'on passiez votre classe Couleur à un camarade sur un travail en projet.

Vous connaissez bien les couleurs, mais pas lui . Et hop il utilise la commande :

UneCouleur.V = 400 , cela risque de poser un problème lors de l'exécution !

Voici la solution proposée par python :

```
1
2  class Couleur:
3
4      #Pour la composante rouge
5      R :int
6      #Pour la composante verte
7      V :int
8      #Pour la coposante Bleue
9      B :int
10
11     image = "RVB"
12
13     def __init__(self,rouge:int,vert:int,bleu:int):
14         self.R,self.V,self.B = rouge,vert,bleu
15
16     def __str__(self):
17         return "Rouge:" + str(self.R) + " Vert: " + str(self.V) + " Bleu:" + str(self.B)
18
19     @property
20     def V(self):
21         return self.__V
22     @V.setter
23     def V(self,composanteverte):
24         assert composanteverte >=0 and composanteverte <= 255," la valeur proposée pour le vert n'est pas correcte "
25         self.__V = composanteverte
26
27 UneCouleur = Couleur(255,0,0)
28 #UneDeuxiemeCouleur = Couleur(0,255,0)
29 #print(UneCouleur)
30 #print(UneDeuxiemeCouleur)
31
32 print(UneCouleur.image)
33 #print(UneDeuxiemeCouleur.image)
34
35 #UneCouleur.image ='EchelleDeGris'
36 #print(UneDeuxiemeCouleur.image)
37
38 UneCouleur.V = 500
39 print(UneCouleur)
```

```
RVB
Traceback (most recent call last):
  File "TADCouleur1.py", line 38, in <module>
    UneCouleur.V = 500
  File "TADCouleur1.py", line 24, in V
    assert composanteverte >=0 and composanteverte <= 255," la valeur proposée p
our le vert n'est pas correcte "
AssertionError:  la valeur proposée pour le vert n'est pas correcte
```

C'est tipabo ?