Numérique et sciences informatiques
Classe de terminale

I Programmation et algorithmes
C Notions fondamentales en programmation
1. La programmation orientée objet

I Première approche d'une classe d'objets : instances et attributs

I.1 Premières notions d'une classe : attributs et instances

• Une **classe** est une structure de données définie par un ensemble de propriétés appelées **attributs**.

• Un attribut est défini par :

* son nom

* le type de la valeur associée à cet attribut.

• Un objet Python possédant cette structure est appelé un **objet** ou une **instance** de la classe.

Représentation d'une classe

I.2 Création d'une classe en Python

a) La syntaxe en Python

• La création d'une classe est introduite par une première ligne commençant par le mot-clé `class`. L'ensemble du code définissant les éléments de cette classe est indenté.

• Les premières lignes de cette définition doivent avoir la forme suivante :

```
class Nom_de_classe :

    def __init__(self, arg1, arg2 ...) :
        self.at1 = ...
        self.at2 = ...
        ...
```

Remarques sur la syntaxe

• la méthode `__init__` est **le constructeur de la classe**. Il permet de créer et d'initialiser les attributs qui seront donnés à toute instance (ou objet) de la classe. Elle est appelée donc à chaque nouvelle création d'une instance de la classe.

• Il n'est pas nécessaire que la méthode `__init__` comporte autant d'argument que la classe possède d'attributs car certains attributs pourront être initialisés avec des valeurs par défaut.

• L'entête de la méthode comporte toujours le mot-clé `self` comme premier argument. Ce mot-clé `self` désigne l'instance que l'on vient de créer et pour laquelle on crée et initialise des attributs.

• Une règle de bonne pratique en Python demande de toujours commencer un nom de classe par une lettre majuscule.

b) Un exemple de création de classe

• On veut créer une structure de données `Chrono` qui exprime une durée en heures, minutes et secondes. Ainsi,

* le nom de la classe est `Chrono` ;

* les objets (ou instances) de cette classe posséderont tous trois attributs `heures`, `minutes` et `secondes` dont les valeurs seront de type `int`.

• On peut écrire le code suivant :

```
class Chrono :

    def __init__(self, h, m, s) :
        self.heures = h
        self.minutes = m
        self.secondes = s
```

I.3 Création d'instances et manipulations des attributs d'instance

a) La création d'une instance

• Une fois que la classe a été créée, on peut créer des instances de la classe dans le programme principal en suivant la syntaxe suivante :

```
obj = Nom_de_classe(v1, v2 , ...)
```

• Dans cette instruction de création d'un objet de la classe, on doit donner le même nombre d'arguments que ceux mis dans l'en-tête de la méthode `__init__` de la classe sans compter l'argument `self`. Lors de l'exécution de la méthode `__init__`, la valeur `v1` sera affectée à l'argument `arg1`, la valeur `v2` à l'argument `arg2`, etc.

• Cette instruction déclenche deux actions :

* La création d'un objet de la classe `Nom_de_classe` qui sera affecté à la variable `var`.

* L'appel de la méthode `__init__` qui crée et initialise les attributs de cet objet à l'aide des arguments `v1, v2 ,` etc. donnés dans l'instruction.

• *Exemples de création d'instances de la classe `Chrono` :*

```
T1 = Chrono(3, 51, 24)
T2 = Chrono(4, 48, 14)
T3 = Chrono(2, 38, 24)
```

b) Lecture de la valeur d'un attribut

• Pour lire la valeur d'un attribut `att` d'une variable `obj` qui a pour valeur un objet de la `classe Nom_de_classe`, il suffit d'écrire l'expression suivante :

```
obj.att
```

• Exemples :

```
>>> T1.heures
3
>>> T2.minutes
48
```

```
>>> T3.secondes
24
```

c) Modification de la valeur d'un attribut

• Pour modifier la valeur d'un attribut `attrib` d'un objet de la classe il suffit de lui affecter une nouvelle valeur de la façon suivante :

```
obj.attrib = nouv_val
```

• Par exemple, on peut augmenter la valeur de `T2` de 5 minutes de la façon suivante :

```
>>> T2.minutes += 5
>>> T2.minutes
53
```

• Représentation de l'état d'un objet à un instant donné :

Remarque importante

Comme les tableaux et les dictionnaires, les objets d'une classe sont des objets mutables. Ils sont donc soumis aux effets de bord.

Exemple :

```
>>> T4 = T1
>>> T4.heures
3
>>> T4.heures = 7
>>> T1.heures
7
```

II Prolongements sur la notion de classe : création de méthodes

II.1 La notion d'encapsulation

• Un principe général : on veut éviter que l'utilisateur ait accès directement à la totalité ou à une partie des attributs d'une classe. Ceci permet notamment de garantir à tout instant l'intégrité des données (leur cohérence) car des modifications non contrôlées de ces données peut compromettre leur cohérence générale et donc produire des erreurs dans la manipulation des données.

• On associe donc à une classe des **méthodes**, c'est-à-dire des fonctions dédiées à la manipulation des objets de la classe.

• On, peut distinguer différents types de méthodes :

* les **accesseurs** : pour lire les informations contenues dans l'objet ;

* les **mutateurs** : pour modifier ou mettre à jour les informations contenues dans l'objet ;

* d'autres méthodes qui remplissent des fonctions d'affichage, de comparaison, etc.

II.2 Définition de méthodes dans la classe et appel de la méthode

a) Définition de la méthode dans la classe

• Pour définir une méthode on utilise la syntaxe suivante,

```
    def methode(self [, arg1, arg2, ...]) :
        ...
```

Erreurs à éviter

• Ne pas oublier l'indentation qui permet d'indiquer que la méthode fait partie de la définition de la classe.

• Ne pas oublier le premier argument `self` qui est obligatoire.

• La méthode peut ne pas comporter d'autre argument que `self`.

b) Appel de la méthode dans un programme principal

• Le premier argument écrit dans l'en-tête de la méthode est toujours le mot-clé `self` qui désigne l'objet auquel on applique la méthode.

• Si la méthode renvoie un résultat (donc comporte au moins un `return`) :

```
res = obj.methode(v1, v2, ...)
```

• Si la méthode ne renvoie pas de résultat, par exemple si elle effectue une modification d'un attribut ou bien un affichage :

```
obj.methode(v1, v2, ...)
```

Attention : comme pour un appel de fonction, il ne faut pas oublier d'écrire les parenthèses dans un appel de méthode. Ceci est aussi vrai, même si cette méthode ne comporte pas d'autre argument que `self`.

Par exemple, on peut définir et utiliser une méthode `reset` dans la classe `Chrono` de la façon suivante :

```
Class Chrono :

    def __init__(self, s, m, h) :
        ...

    def reset(self) :
        self.secondes = 0
        self.minutes = 0
        self.heures = 0

T = Chrono(2, 34, 28)
T.reset()
```

• Le constructeur `__init__` est aussi une méthode de la classe. En revanche cette méthode est appelée lors de la création d'une instance de la classe par une instruction de la forme

```
obj = Nom_de_classe(...)
```

et non pas par une instruction de la forme :

```
obj.__init__(...)
```

Les caractères '__' qui encadrent 'init' servent à indiquer qu'il s'agit là d'une méthode cachée qui ne doit pas être appelée de cette façon (même si une instruction de cette forme ne déclenche aucun message d'erreur).

II.3 Exemples de création de méthodes pour la classe `Chrono` :

a) Affichage d'un chronomètre

• On peut créer une méthode d'affichage dans la console de la valeur d'un chronomètre :

```
    def afficher(self) :
        print(str(self.heures) + "h " + str(self.minutes) \
              + "m " + str(self.secondes) + "s.")
```

• L'exécution du programme principal suivant :

```
T = Chrono(2, 7, 43)
T.afficher()
```

générera l'affichage suivant dans la console :

```
2h 7m 43s.
```

• Mais on peut aussi utiliser la méthode prédéfinie `__str__` pour renvoyer une chaîne de caractères :

```
    def __str__(self) :
        return str(self.heures) + "h " + str(self.minutes)\
         + "m " + str(self.secondes) + "s."
```

Cette méthode `__str__` sera appelée à chaque appel `print(T)` si `T` est un objet de la classe `Chrono`. Cet appel affichera donc le contenu de l'objet `T` sous le format défini par cette méthode.

• Ainsi l'exécution du programme suivant :

```
T = Chrono(5, 12, 29)
print(T)
```

générera l'affichage suivant dans la console :

```
5h 12m 29s.
```

b) Ajout de temps à un chronomètre

```
    def ajout_temps(self, h, m, s) :
        nb_sec = self.secondes + s
        self.secondes = nb_sec%60
        nb_min = self.minutes + m + nb_sec//60
        self.minutes = nb_min%60
        nb_hr = self.heures + h + nb_min//60
        self.heures = nb_hr
```

• Le programme principal suivant

```
T = Chrono(6, 51, 49)
print("1re valeur de T :", T)
T.ajout_temps(2, 24, 54)
print("2me valeur de T :", T)b
```

génèrera l'affichage dans la console :

```
1re valeur de T : 6h 51m 49s.
2me valeur de T : 9h 16m 43s.
```