

Notions théoriques de classe

Table des matières

I. Notions de classes et d'instance	3
II. Exercice : Quiz	6
III. Apprendre à utiliser les attributs et les méthodes	7
IV. Exercice : Quiz	13
V. Essentiel	14
VI. Auto-évaluation	14
A. Exercice	14
B. Test	14
Solutions des exercices	16

I. Notions de classes et d'instance

Durée : 1 h

Environnement de travail : PC connecté à internet

Contexte

Une classe est la définition d'un concept métier, elle contient les attributs comme des valeurs et des méthodes avec les fonctions. Les classes permettent de réunir des données et des fonctionnalités. En créant une nouvelle classe, vous pouvez créer un nouveau type d'objet, et de nouvelles instances de ce type peuvent être construites. Chaque instance peut avoir ses propres attributs, ce qui permet de définir son état. Une instance peut aussi avoir des méthodes (définies par la classe de l'instance) pour modifier son état.

Prenons l'exemple d'une banque qui possède un fichier contenant des informations sur ses clients. Ces derniers sont dans l'incapacité d'avoir accès directement à ces informations. Cependant, ils peuvent demander à leur banquier les données les concernant et qui sont détenus par la banque. Si ces données se révèlent inexactes, le client peut aussi demander sa modification.

Cette banque est comparable à un « *objet* » qui détient des informations et des moyens qui peuvent permettre de lire et de modifier ces informations. En réalité, cette banque cache son fonctionnement interne ainsi que les données de ses clients, vue de l'extérieur. Au sens informatique, l'objet désigne une entité possédant des informations et des méthodes qui permettent de les manipuler. On peut donc définir une classe comme un assemblage de variables appelées « *attributs* » et de fonctions nommées « *méthodes* ». La programmation objet regroupe l'ensemble des propriétés associées aux classes.

Remarque

Objectifs

- Comprendre la notion de classe
- Comprendre la différence avec la notion d'instance

Définition

Qu'est-ce qu'une classe ?

On appelle « *classe* » l'ensemble formé par les variables ou attributs ou les fonctions (méthodes). Notez que les attributs sont des variables accessibles depuis toutes les méthodes de la classe où elles sont définies. Par ailleurs, les classes sont modifiables en Python.

Le nom des classes

En Python, le nom d'une classe ne peut commencer par un chiffre ou un symbole de ponctuation. Vous ne pouvez pas non plus utiliser un mot-clé du langage comme `while` ou `if`. À part ces contraintes, Python est permissif sur le nom des classes.

Voici l'implémentation d'une classe en Python ne possédant pas de membre, attribut ou méthode :

(1)

```
1 class MaClass :
2     # Pour l'instant, elle est vide
3     pass
4     # C'est pourquoi on utilise le mot-clé pass
```

Le mot-clé `class` est précédé du nom de la classe. Le corps de la classe est en retrait comme dans le corps d'une fonction. Dans un corps de classe, il est possible de définir :

- Des fonctions (qui deviendront des méthodes de la classe).
- Des variables (qui deviendront des attributs de la classe).
- Des classes imbriquées, internes à la classe principale.

Il est possible d'organiser le code de façon plus précise grâce à l'imbrication de classes. Il est possible et même conseillé de répartir les classes d'une application dans plusieurs fichiers (modules). Il est plus lisible et logique de déclarer certaines classes dans une classe « hôte ». La déclaration d'une classe imbriquée dans une autre ressemble à ceci :

(2)

```
1 <python>
2 Classe contenante, déclarée normalement
3 class Fruit :
4     #Classes contenues, déclarées normalement
5     class Pomme :
6         pass
7     class Orange :
8         pass
```

La seule implication de l'imbrication est qu'il faut désormais passer par la classe `Fruit` pour utiliser les classes `Pomme` et `Orange` en utilisant l'opérateur d'accès « *point* » : `Fruit.Pomme` et `Fruit.Orange`. Exactement comme on le ferait pour un membre classique.

L'imbrication `nested class` n'impacte en rien le comportement du contenant ni du contenu. Gardez en tête qu'il s'agit d'une pratique anecdotique.

Pour information, `.Fruit Pomme` et `Orange` ne se coderait pas avec des classes imbriquées. On aurait une classe abstraite dont hérite une classe `Pomme` et une classe `Orange`, ou alors juste un attribut `Genre` sur la classe `Fruit`.

Qu'est-ce qu'une instance ?

Si vous étiez un artisan qui fabrique des vases, vous utiliseriez des moules pour appliquer des caractéristiques variables comme la forme ou la couleur à vos produits. Il faut voir le moule comme une classe et les vases comme des instances.

Si on exécute le code précédent de déclaration de classe vide, rien ne s'affiche. C'est normal, puisqu'on n'a fait que déclarer une classe. Après cette exécution, l'environnement Python sait qu'il existe désormais une classe nommée « *MaClasse* ». Maintenant, il va falloir l'utiliser.

Une classe permet de présenter, d'exposer les données du concept qu'elle représente. Afin de manipuler réellement ces données, il va falloir une représentation concrète. C'est l'instance ou l'objet de la classe.

Une instance (ou objet) constitue un exemple d'une classe. L'instanciation est le mécanisme qui permet de créer un objet à partir d'une classe. Si l'on compare une instance à un vêtement, alors la classe est le patron ayant permis de le découper et la découpe en elle-même est l'instanciation. Par conséquent, une classe peut générer de multiples instances, mais une instance ne peut avoir comme origine une seule classe.

Le rôle des parenthèses

Si l'on veut créer une instance de `MaClasse` :

(3)

Les parenthèses sont importantes. Elles indiquent un appel de méthode. Dans le cas précis d'une instanciation de classe, la méthode s'appelle `__init__` : c'est le constructeur de la classe. Si cette méthode n'est pas implémentée dans la classe, alors un constructeur par défaut est automatiquement appelé, comme c'est le cas dans cet exemple. Il est désormais possible d'afficher quelques informations sur la console :

```
1 instance = MaClasse()
```

(4)

Lorsqu'on affiche sur la sortie standard la variable `instance`, l'interpréteur informe qu'il s'agit d'un objet de type `__main__.MaClasse` avec l'adresse de mémoire.

D'où vient ce `__main__` ? Il s'agit du module dans lequel `MaClasse` a été déclarée. En Python, un fichier source correspond à un module, et le fichier qui sert de point d'entrée à l'interpréteur est appelé `__main__`. Le nom du module est accessible *via* la variable spéciale `__name__`.

Exemple

```
1 print(instance)
2 > faire
```

(5)

Le fichier `a.py` sert d'entrée à l'interpréteur Python : ce module se voit donc attribuer `__main__` comme nom. Lors de l'import du module `b.py`, le fichier est entièrement lu et affiche le nom du module qui correspond, lui, au nom du fichier.

Une classe faisant toujours partie d'un module, la classe `MaClasse` a donc été assignée au module `__main__`.

L'adresse hexadécimale de la variable `instance` correspond à l'emplacement mémoire réservé pour stocker cette variable. Cette adresse permet, entre autres, de différencier deux variables qui pourraient avoir la même valeur.

Exemple

```
1 # Ceci est le module b.
2 print(« Nom du module du fichier b.py : » + __name__)
3 chiffre = 42
4 b.py

1 # Ceci est le module a.
2 print(« Nom du module du fichier a.py : » + __name__)
3 import b
4 print(b.chiffres)
5 a.py

1 $ python a.py
2 Nom du module du fichier a.py : __main__
3 Nom du module du fichier b.py : b
4 (5-bis)
```

Lorsqu'on assigne `a` et `b` et qu'on affiche `b`, on constate que les variables pointent vers la même zone mémoire. Par contre, si l'on assigne à `b` une nouvelle instance de `MaClasse`, alors son adresse n'est plus la même : il s'agit d'une nouvelle zone mémoire allouée pour stocker un nouvel exemplaire de `MaClasse`.

Exemple

```
1 a = MaClasse()
2 print('Info sur 'a' : {}'.format(a))
3 b =a
4 print('Info sur 'b' : {}'.format(b))
5 b = MaClasse()
6 print('Info sur 'b' : {}'.format(b))
```

(6)

En Python, tout est objet, y compris les classes. N'importe quel objet peut être affecté à une variable, et les classes ne font pas exception. Il est donc tout à fait valide d'assigner MaClasse à une variable classe, et son affichage sur la sortie standard confirme bien que classe est une classe et pas une instance. D'où l'importance des parenthèses lorsque l'on désire effectuer une instanciation de classe. En effet, les parenthèses précisent bien que l'on appelle le constructeur de la classe, et on obtient par conséquent une instance. L'omission des parenthèses signifie que l'on désigne la classe elle-même.

Exemple

```
print(MaClasse)

x

classe = MaClasse
print(classe)
```

Exemple

« Appliquer la notion de classe »

Intitulé :

Créer une classe Palindrome contenant une méthode de classe `EstPalindrome()` qui renvoie une valeur booléenne indiquant si une chaîne de caractères passée en argument est un palindrome. Un palindrome est une chaîne qui peut se lire indifféremment de gauche à droite ou de droite à gauche. Les caractères non alphanumériques sont pris en compte.

Une chaîne d'un seul caractère ou une chaîne vide sont des palindromes.

Indice (optionnel) :

Si les premiers et derniers caractères sont identiques et si la sous-chaîne restante est elle-même un palindrome, alors la chaîne entière est un palindrome.

L'utilisation de l'indice - 1 permet de récupérer le dernier caractère d'une chaîne.

L'utilisation du caractère ':' permet d'extraire une sous-chaîne en précisant les indices de début et de fin.

Solution :

```
1 Class Palindrome :
2     def estPalindrome(s) :
3         if len(s) <=1 :
4             return True
5         return s[0] == s[-1] and Palindrome.estPalindrome(s[1 :-1])
6 print(Palindrome.estPalindrome('Sonar'))
```

Exercice : Quiz

[solution n°1 p.17]

Question 1

Une classe est composée uniquement d'attributs.

- ☐ Vrai
- ☐ Faux

Question 2

En Python, il n'est pas possible de modifier les classes.

- ☐ Vrai
- ☐ Faux

Question 3

En Python, le nom d'une classe commence par un chiffre ou un symbole de ponctuation.

- ☐ Vrai
- ☐ Faux

Question 4

Une instance est un objet créé à partir d'une classe.

- ☐ Vrai
- ☐ Faux

Question 5

Les parenthèses sont négligeables en Python.

- ☐ Vrai
- ☐ Faux

III. Apprendre à utiliser les attributs et les méthodes

Objectifs

- Apprendre à utiliser les membres d'une classe
- Apprendre à utiliser les méthodes

Définition Qu'est-ce qu'un attribut ?

Un attribut est une variable associée à une classe. Pour définir un attribut au sein d'une classe, il suffit d'assigner une valeur à cet attribut dans le corps de la classe.

(1)

```
1 Class Cercle :  
2     # Déclaration d'un attribut de classe 'rayon'  
3     # auquel on assigne la valeur 2  
4     rayon = 2  
5 print(Cercle.rayon)
```

Les attributs définis ainsi sont appelés « *attributs de classe* », car ils sont liés à la classe, par opposition aux « *attributs d'instance* » dont la vie est liée à l'instance à laquelle ils sont rattachés. Les attributs de classe sont automatiquement reportés dans les instances de cette classe et deviennent des attributs d'instance.

Puisqu'un attribut de classe est lié à la classe et non pas à l'instance, il existe durant toute la durée d'exécution du programme. Plus précisément, il existe tant que la classe à laquelle il est lié est définie. Un attribut d'instance, quant à lui, n'existe qu'à travers l'instance qui lui est liée. Ainsi, si l'instance est détruite, l'attribut d'instance l'est également.

(2)

```
1 c = Cercle()
2 print(c.rayon)
```

(3)

Tandis qu'un attribut de classe survit à toutes les instances de cette classe.

Exemple

```
1 #Déclaration d'une instance de Cercle
2 c = Cercle()
3 # Déclaration d'un attribut d'instance 'rayon'
4 c.rayon = 5
5 # Affichage de cet attribut d'instance
6 print(c.rayon)
7 #Destruction de l'instance de Cercle
8 del(c)
9 #Affichage de l'attribut d'instance
10 print(c.rayon)
11 #c n'est plus défini dans l'environnement d'exécution de Python
12 #Par conséquent, les attributs liés à cette instance non plusieurs
```

(4)

Si l'attribut de classe est copié dans chaque objet instancié en tant qu'attribut d'instance, il n'en demeure pas moins que ces attributs demeurent indépendants l'un de l'autre. Toute modification sur l'attribut d'instance n'a aucun impact sur l'attribut de classe. De façon similaire, si la valeur de l'attribut de classe est changée, cela n'a aucun impact sur les objets déjà instanciés (mais cela en aura sur les futures instances).

Exemple

```
1 #Déclaration d'une instance de Cercle.r
2 c = Cercle()
3 #Destruction de l'instance de Cercle
4 del(c)
5 #Affichage de l'attribut de classe
6 print(Cercle.rayon)
7 #L'attribut de classe existe toujours
```

(5)

Faire le choix entre l'attribut de classe et l'attribut d'instance

Le choix entre attribut de classe et attribut d'instance dépend de la portée que l'on veut accorder à cet attribut. Si l'on désire que la valeur soit globale à tout le programme, alors l'attribut de classe est la solution, car il ne dépend d'aucune instance. Cependant, toute modification de sa valeur a des répercussions sur l'ensemble de l'application, ce qui peut entraîner des comportements inattendus si ce changement n'est pas totalement maîtrisé.

Ces bogues comportent d'autant plus de risques que le code est volumineux. L'attribut de classe est donc à utiliser avec précaution. Si la valeur de l'attribut est dépendante de chaque instance de classe, alors l'attribut d'instance est la bonne solution. Cette valeur a une durée de vie égale à l'objet qui la contient, et tout changement n'a que des implications locales à l'objet.

Qu'est-ce qu'une méthode ?

Une méthode est une fonction définie dans une classe ayant comme premier argument une instance de cette classe.

Exemple

```
1 c.rayon = 4
2 print(c.rayon)
3 #Attribut de l'instance c
4 print(Cercle.rayon)
5 #Attribut de la classe Cercle
6 Cercle.rayon = 6
7 print(Cercle.rayon)
8 #Attribut de la classe dont la valeur vient d'être modifiée
9 print(c.rayon)
10 # Attribut de l'instance c, qui demeure inchangé.
```

(6)

L'utilisation du mot-clé `def` se fait comme lorsqu'on définit une fonction dans l'espace de noms global. Le faire dans le corps de la classe, à l'instar des attributs, lie la fonction à la classe. Cependant, pour être appelée, une méthode doit obligatoirement prendre en premier argument une instance de la classe à laquelle elle est liée. La convention est de nommer cet argument « `self` ».

`Self` n'est pas un mot-clé du langage comme `class` ou `while`. Vous pouvez très bien donner un autre nom à ce premier paramètre. C'est toutefois déconseillé. La plupart des développeurs Python utilisent cette convention.

Oublier l'argument `self` provoque une erreur lors de l'appel de la méthode.

Exemple

```
1 Class Cercle :
2     #Déclaration d'une méthode nommée périmètre
3     Def périmètre(self) :
4         # Définition du corps de la méthode avec une valeur de retour
5         #return 2 * 3.14 * self.rayon
6 c = Cercle()
7 c.rayon = 2
8 #Appel de la méthode périmètre() de l'instance c.
9 print(c.périmètre())
```

(7)

Le message d'erreur peut prêter à confusion : aucun paramètre n'est donné à la méthode `périmètre()`, alors pourquoi l'interpréteur Python se plaint-il d'en recevoir un pendant, alors qu'il n'en attend aucun ?

Contrairement à d'autres langages orientés objet où l'instance est automatiquement accessible dans la méthode, en Python l'instance est un paramètre de la méthode qui, lui, est automatiquement passé en argument à l'appel. Lorsqu'on appelle une méthode *via* une instance, c'est en fait la méthode de « classe » qui est appelée avec l'instance en premier paramètre `self`. Par conséquent, ces deux lignes sont équivalentes.

Exemple

```
1 Class Cercle 2 :
2     def périmètre() :
3         return 2 * 3.14 * rayon
4 c2 = Cercle 2()
5 c2.rayon = 2
6 print(c2.périmètre())
```

(8)

Si l'on appelle une méthode n'ayant pas `self` comme premier argument, Python devient plus explicite. Si on omet l'argument de `périmètre()` on obtient :

```
1 print(c.périmètre())
2
3 print(Cercle.périmètre(c))
```

(9)

L'erreur est alors plus claire : la méthode doit être appelée avec une instance de `Cercle` 2 comme premier argument. Ce n'est effectivement pas le cas puisque l'argument `self` n'a pas été déclaré dans la liste des paramètres. Ce message d'erreur apporte une autre lumière sur les mécanismes en œuvre. Il s'agit du terme *unbound method* (« méthode non liée »). En Python, tout est objet, y compris les méthodes.

Exemple

```
1 print(Cercle 2.périmètre())
2 TypeError
```

(10)

« *unbound method* » signifie que la méthode n'est liée à aucune instance et, par conséquent, elle ne peut être appelée sans argument. Afin de l'utiliser correctement, il faut donc lui fournir une instance de `Cercle` comme premier argument : c'est le fameux `self`. Par contre, une « *bound method* » est bien liée à une instance de `Cercle`.

Puisqu'elle est déjà liée, plus besoin de lui fournir l'instance, l'argument `self` est déjà établi. En tant qu'objet, une méthode peut également être assignée à une variable.

Exemple

```
1 #Affichage de la méthode de « classe »
2 print(Cercle.périmètre)
3
4 #Affichage de la méthode « d'instance »
5 print(c.périmètre)
```

(11)

Il est important d'observer l'utilisation des parenthèses pour bien comprendre ces exemples. Les parenthèses sont utilisées afin de provoquer l'appel d'une méthode. Sans parenthèses, on n'appelle pas, on accède à la méthode comme s'il s'agissait d'un attribut « *classique* », par exemple un entier ou une chaîne de caractères. Une fois la variable `p` assignée, comme il s'agit désormais d'une méthode, l'utilisation des parenthèses est de mise pour appeler cette méthode. On dit que `p` est « *appelable* ».

Python propose la fonction native `callable()` qui vérifie si un objet peut être utilisé comme une fonction, s'il est callable.

Exemple

```
1 p= Cercle.perimetre
2 c = Cercle()
3 c.rayon = 2
4 print(p)
5
6 # p est la méthode perimetre de la classe Cercle.
7 # Cette méthode est stockée à l'adresse mémoire identique.
8
9 print(p(c))
10 # L'appel de la méthode de classe avec une instance en paramètre produit le résultat attendu
11 p = c.perimetre
```

```

12 print(p)
13 # p est la méthode perimetre de la classe Cercle liée à l'instance de Cercle qui est stockée à
    l'adresse indiquée
14 print(p())
15 #L'appel sans argument de la méthode liée à l'instance produit le résultat attendu

```

(12)

L'utilisation de méthodes en tant qu'objets permet de puissantes combinaisons, stockage en tant qu'attribut, liste de méthodes ou encore des méthodes qui prennent d'autres en argument.

Exemple

```
1 print(callable(p))
```

(13)

Exemple Quelques exemples de code sur la notion de méthode

```

1 c = Cercle()
2 c.rayon = 3
3 #Déclaration d'une liste de méthodes.
4 méthode cercle = [Cercle.diamètre, Cercle.perimetre, Cercle.aire]
5 #Boucle parcourant cette liste de méthodes.
6 For m in méthodes cercle :
7     #Affichage de l'appel de la méthode courante m avec l'instance de Cercle c comme premier
    argument (self)
8     print(m(c))
9 >>> 6 #Résultat de c.diamètre()
10 18.84 #Résultat de c.périmètre()
11 28.26 #Résultat de c.aire()

```

Intitulé :

Définir une propriété qui doit avoir la même valeur pour chaque instance de classe. Définissez un attribut de classe « couleur » avec une valeur, la valeur blanc par défaut. Utilisez le code suivant pour cet exercice.

```

1 class Vehicle:
2     def __init__(self, name, max_speed, mileage):
3         self.name = name
4         self.max_speed = max_speed
5         self.mileage = mileage
6 class Bus(Vehicle):
7     pass
8 class Car(Vehicle):
9     pass

```

Indice :

Le résultat attendu :

Color: White, Vehicle name: School Volvo, Speed: 180, Mileage: 12

Color: White, Vehicle name: Audi Q5, Speed: 240, Mileage: 18

Solution :

```

1 class Vehicle:
2     # Class attribute
3     color = "White"
4     def __init__(self, name, max_speed, mileage):
5         self.name = name
6         self.max_speed = max_speed
7         self.mileage = mileage

```

```

8 class Bus(Vehicle):
9     pass
10 class Car(Vehicle):
11     pass
12 School_bus = Bus("School Volvo", 180, 12)
13 print(School_bus.color, School_bus.name, "Speed:", School_bus.max_speed, "Mileage:",
      School_bus.mileage)
14 car = Car("Audi Q5", 240, 18)
15 print(car.color, car.name, "Speed:", car.max_speed, "Mileage:", car.mileage)

```

L'intérêt de l'encapsulation

Objectifs :

- Comprendre l'intérêt de l'encapsulation
- Distinguer les différentes visibilité

Contexte

Un des paradigmes de la POO est que chaque classe doit avoir accès au strict minimum d'informations nécessaires pour accomplir son rôle. Une gestion trop laxiste de l'accès à l'information peut provoquer des bogues, des incompréhensions dans le code des dépendances inutiles, voire néfastes au projet. D'où l'intérêt pour les classes de maîtriser l'exposition de leurs membres.

Il existe trois visibilité, représentées par différents symboles en UML (*Unified Modeling Language*) :

- Publique (symbole +) : le membre est accessible à toutes les autres classes du programme.
- Protégée (symbole #) : le membre est accessible uniquement aux classes dérivées de la classe en question.
- Privée (symbole -) : le membre est accessible uniquement à la classe en question.

L'intérêt de la visibilité est de contrôler l'accès du reste du programme aux membres d'une classe. Il y a des données qui doivent être accessibles en lecture, mais pas en écriture par exemple. Dans ce cas, il est nécessaire de ne pas offrir une interface pour modifier ces données. Il existe des attributs utiles uniquement à la classe qui les définit afin d'effectuer des calculs compliqués. La modification de ces attributs pourrait provoquer des erreurs.

Dans le but de contrôler ces attributs, il faut faire appel à de l'encapsulation, c'est-à-dire les protéger en définissant leur visibilité comme privée, et utiliser des méthodes pour y accéder : ce sont les accesseurs.

Généralement, les accesseurs sont :

- Un getter pour récupérer la valeur de l'attribut
- Un setter pour lui assigner une valeur

Les accesseurs n'ont rien d'obligatoire.

En Python, tout est public. Il n'existe pas de mécanisme prévu pour empêcher l'appel d'une méthode ou d'un attribut privé. Cependant, il existe une convention qui simule ce comportement attendu :

- Si un membre de classe ne fait pas partie de l'interface publique, mais est uniquement présent pour une question d'implémentation, alors son nom est préfixé par un underscore (tiret du 8). Cela ne l'empêche pas d'être accessible de l'extérieur de la classe. Cependant, à condition de rédiger une documentation claire et d'informer les participants au projet, l'effet escompté devrait être obtenu.
- Si un membre est littéralement privé, donc ni public ni protégé, alors il est possible de préfixer son nom avec deux underscores. Lorsque l'interpréteur Python rencontre un nom de membre avec deux underscores, il préfixe automatiquement ce nom par un underscore suivi du nom de la classe. Cela permet d'éviter des conflits de nom entre classes mères et classes filles.

Exemple **Code**

```

1 Class ClasseMere :
2     _ membre = 'Je suis ta mère'
3 class Classe Fille :
4     __membre = 'Nooon !'
5 print(ClasseMere._ClasseMere__membre)
6 >>> Je suis ta mère
7 print(ClasseFille._ClasseFille__membre)
8 >>> 'Nooon !'
9 print(ClasseFille.__membre)
10 error

```

Exemple « Appliquer la notion de membre protégé »

Intitulé :

Vous travaillez avec des données confidentielles.

Une personne X essaye d'échapper à la justice. Créez une première classe `Datas` contenant l'objet `Confidentiel`, qui sera protégée.

Créez une seconde classe `Non Confidentiel` qui prend comme argument votre première classe.

Essayez d'imprimer l'objet `Confidentiel` à partir des deux classes.

Indice (optionnel) :

Vous devez obtenir une erreur d'attribut avec la classe `Datas`, puisque l'objet `Confidentiel` est protégé.

Solution :

```

1 class Datas:
2     def __init__(self):
3         # Membres protégés
4         self._Confidentiel = 'Données confidentielles'
5 # Classe dérivée
6 class NonConfidentiel(Datas):
7     def __init__(self):
8         # Constructeur de la classe Base
9         Datas.__init__(self)
10        print("Affichage des membres de la classe Justice: ")
11        print(self._NonConfidentiel)
12 obj1 = NonConfidentiel()
13 obj2 = Datas()
14 # L'appel des membres protégés en dehors de la classe doit générer une erreur d'attribut
15 print(obj2.a)

```

Exercice : Quiz

[solution n°2 p.17]

Question 1

Les attributs d'instance sont liés à la classe.

- ☐ Vrai
- ☐ Faux

Question 2

Un attribut d'instance peut exister sans l'instance lui-même.

- ☐ Vrai
- ☐ Faux

Question 3

Toute modification de l'instance de classe peut avoir des répercussions sur l'ensemble d'une application.

- ☐ Vrai
- ☐ Faux

Question 4

Les méthodes constituent des « *objets* » en Python.

- ☐ Vrai
- ☐ Faux

Question 5

Assigner une valeur à un attribut dans le corps de la classe permet de le définir.

- ☐ Vrai
- ☐ Faux

V. Essentiel

En résumé, une classe est un ensemble d'attributs et de méthodes. On entend par « *attributs* » des variables associées à une classe et qui se comportent comme des variables globales pour toutes les méthodes de la classe. La méthode, quant à elle, est un ensemble de fonctions associées à une classe. La particularité des méthodes est qu'elles ont directement accès aux données de la classe. On ne peut parler de « *notions de classe* » sans parler d'« *instances* ». En effet, une instance (ou objet) représente un exemple d'une classe. Ainsi, une instance d'une classe `c` définit une variable de type `c`. Notez que le terme « *instance* » ne peut être appliqué qu'aux variables ayant pour type une classe.

VI. Auto-évaluation

A. Exercice

Vous venez de créer votre entreprise et de recruter vos premiers employés. Pour bien gérer votre organisation, vous souhaitez créer une base de données pour tous vos employés.

Question 1

[solution n°3 p.18]

Écrivez un programme qui stockera à l'aide de la classe `Person()` le prénom et retournera un message d'accueil avec une fonction `greet()`. Finalement, ajoutez un numéro d'identifiant à l'aide de la classe `Employee()` qui prend comme argument votre première classe.

Question 2

[solution n°4 p.19]

Quelle est la différence entre une classe et une instance ?

B. Test

Exercice 1 : Quiz

[solution n°5 p.19]

Question 1

Lequel des mots-clés suivants marque le début de la définition de classe ?

- ☐ def
- ☐ return
- ☐ class
- ☐ Tous

Question 2

Lesquels des énoncés suivants sont corrects ?

- ☐ Une variable de référence est un objet
- ☐ Une variable de référence fait référence à un objet
- ☐ Un objet peut contenir d'autres objets
- ☐ Un objet peut contenir les références à d'autres objets

Question 3

En Python, une classe est pour un objet concret :

- ☐ Une nuisance
- ☐ Une instance
- ☐ Une distraction
- ☐ Un plan

Question 4

En Python, une fonction dans une définition de classe est appelée :

- ☐ Une fonction de classe
- ☐ Une opération
- ☐ Une méthode
- ☐ Aucune de ces propositions

Question 5

Qu'est-ce que l'instanciation en termes de terminologie POO ?

- ☐ La suppression d'une instance de classe
- ☐ La modification d'une instance de classe
- ☐ La copie d'une instance de classe
- ☐ La création d'une instance de classe

Question 6

Lors de l'héritage d'une autre classe, à quelles méthodes de classes pouvez-vous accéder ?

- ☐ Aucune
- ☐ Toutes
- ☐ Cela dépend de la classe

Question 7

Comment hériter d'une autre classe ?

- ☐ En la nommant
- ☐ En appelant un module de la nouvelle classe
- ☐ En passant le nom de l'autre classe comme paramètres
- ☐ Ce n'est pas possible

Question 8

Lors de l'instanciation de votre nouvelle classe, quels paramètres de l'ancienne classe devez-vous passer ?

- ☐ Le premier
- ☐ Cela dépend de la classe
- ☐ Tous


Solutions des exercices

Exercice p. 6 Solution n°1**Question 1**

Une classe est composée uniquement d'attributs.

☐ Vrai

☒ Faux


 Une classe est un ensemble de variables (attributs) et de fonctions (méthodes).

Question 2

En Python, il n'est pas possible de modifier les classes.

☐ Vrai

☒ Faux


 Les classes sont modifiables en Python.

Question 3

En Python, le nom d'une classe commence par un chiffre ou un symbole de ponctuation.

☐ Vrai

☒ Faux


 En Python, le nom d'une classe ne commence jamais par un chiffre ou un symbole de ponctuation.

Question 4

Une instance est un objet créé à partir d'une classe.

☒ Vrai

☐ Faux


 L'instanciation est le procédé par lequel on crée un objet à partir d'une classe. Une instance constitue donc un exemple d'une classe.

Question 5

Les parenthèses sont négligeables en Python.

☐ Vrai

☒ Faux

 Les parenthèses sont indispensables, car elles indiquent un appel de méthode.


Exercice p. 13 Solution n°2

Question 1

Les attributs d'instance sont liés à la classe.

☐ Vrai

☒ Faux


 Les attributs de classe sont liés à la classe, alors que les attributs d'instance sont liés à l'instance à laquelle ils sont rattachés.

Question 2

Un attribut d'instance peut exister sans l'instance lui-même.

☐ Vrai

☒ Faux


 Un attribut d'instance n'existe qu'à travers l'instance qui lui est liée. De ce fait, si l'instance est détruite, l'attribut d'instance l'est également.

Question 3

Toute modification de l'instance de classe peut avoir des répercussions sur l'ensemble d'une application.

☒ Vrai

☐ Faux

 Toute modification de la valeur de l'instance de classe peut entraîner des comportements inattendus, si ce changement n'est pas totalement maîtrisé.

Question 4

Les méthodes constituent des « objets » en Python.

☒ Vrai

☐ Faux


 En python, tout est objet, y compris les méthodes.

Question 5

Assigner une valeur à un attribut dans le corps de la classe permet de le définir.

☒ Vrai

☐ Faux

 Pour définir un attribut au sein d'une classe, il suffit d'assigner une valeur à cet attribut dans le corps de la classe.

p. 14 Solution n°3

```
1 class Person():
2     def __init__(self, name):
3         self.name = name
4     def greet(self):
5         return f' Bienvenue, {self.name}'
```

```
6 class Employee(Person)
7     def __init__(self, name, id_number):
8         self.name = name
9         self.id_number = id_number
10 e = Employee('empname', 1)
```

p. 14 Solution n°4


Une classe est composée d'attributs et de fonctions, alors qu'une instance est un exemple de classe.

Exercice p. 14 Solution n°5

Question 1

Lequel des mots-clés suivants marque le début de la définition de classe ?


- ☐ def
- ☐ return
- ☒ class
- ☐ Tous

 En python, `class` marque le début de la définition de classe. Notez que le nom d'une classe ne peut pas commencer par un chiffre ou un signe de ponctuation.

Question 2

Lesquels des énoncés suivants sont corrects ?


- ☐ Une variable de référence est un objet
- ☒ Une variable de référence fait référence à un objet
- ☐ Un objet peut contenir d'autres objets
- ☒ Un objet peut contenir les références à d'autres objets

 En Python, une variable de référence fait référence à un objet, et n'importe quel objet peut être affecté à une variable.

Question 3


En Python, une classe est pour un objet concret :

- ☐ Une nuisance
- ☐ Une instance
- ☐ Une distraction
- ☒ Un plan

 Une classe est un moyen de réunir des données et des fonctionnalités. Il s'agit donc d'un plan pour un objet concret.


Question 4

En Python, une fonction dans une définition de classe est appelée :

- ☐ Une fonction de classe
- ☐ Une opération
- ☒ Une méthode
- ☐ Aucune de ces propositions
-  En python, une méthode représente une fonction appartenant à un objet.


Question 5

Qu'est-ce que l'instanciation en termes de terminologie POO ?

- ☐ La suppression d'une instance de classe
- ☐ La modification d'une instance de classe
- ☐ La copie d'une instance de classe
- ☒ La création d'une instance de classe
-  L'instanciation est la création d'une instance de classe, une instance étant une variable donnée de la classe.


Question 6

Lors de l'héritage d'une autre classe, à quelles méthodes de classes pouvez-vous accéder ?

- ☐ Aucune
- ☒ Toutes
- ☐ Cela dépend de la classe
-  Il est possible d'accéder à toutes les méthodes au cours de l'héritage d'une autre classe.


Question 7

Comment hériter d'une autre classe ?

- ☐ En la nommant
- ☐ En appelant un module de la nouvelle classe
- ☒ En passant le nom de l'autre classe comme paramètres
- ☐ Ce n'est pas possible
-  Pour hériter d'une nouvelle classe en Python, vous pouvez passer le nom de l'autre classe comme paramètre. De plus, la sous-classe hérite des attributs et des méthodes de la classe mère.

Question 8

Lors de l'instanciation de votre nouvelle classe, quels paramètres de l'ancienne classe devez-vous passer ?

- ☐ Le premier
- ☐ Cela dépend de la classe
- ☒ Tous
-  Tous les paramètres de l'ancienne classe doivent être passés lorsqu'il s'agit de l'instanciation de votre nouvelle classe.