Programmation orientée objet

Langages et programmation

Exercice 1 *QCM d'activation*

Pour chaque question, une seule réponse est correcte.

1. On considère le code suivant :

```
class Chien:
    def __init__(self, nom, age):
        self.nom = nom
        self.age = age

rex = Chien("Rex", 5)
```

Dans ce code, `rex` est :

A. une classe

B. un attribut

C. une instance (objet)

D. une méthode

2. Quel est le rôle du paramètre `self` dans une méthode ?

A. C'est un mot-clé obligatoire de Python

B. Il fait référence à la classe elle-même

C. Il fait référence à l'objet sur lequel la méthode est appelée

D. Il contient la valeur de retour de la méthode

3. On considère :

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

p = Point(3, 7)
```

Que vaut `p.x + p.y` ?

A. une erreur car on ne peut pas additionner des attributs

B. `(3, 7)`

C. `10`

D. `"37"`

4. Que se passe-t-il si on oublie `self` dans la définition d'une méthode ?

```python
class Voiture:
    def __init__(self, marque):
        self.marque = marque
    def klaxonner():      # self oublie
        return "Bip !"

v = Voiture("Renault")
v.klaxonner()
```

   A. La méthode affiche `"Bip !"`

   B. Python lève une erreur `TypeError`

   C. La méthode retourne `None`

   D. Python ignore le `self` manquant

5. Quelle méthode spéciale faut-il définir pour que `print(obj)` affiche une représentation lisible ?

   A. `__init__`

   B. `__add__`

   C. `__repr__`

   D. `__eq__`

Corrigé de l'exercice 1.

1. **Réponse C.** `rex` est une instance de la classe `Chien`, créé par l'appel `Chien("Rex", 5)`.
2. **Réponse C.** `self` fait référence à l'objet courant. Quand on écrit `rex.aboyer()`, Python passe automatiquement `rex` comme valeur de `self`.
3. **Réponse C.** `p.x` vaut 3 et `p.y` vaut 7, donc 3 + 7 = 10.
4. **Réponse B.** Python passe automatiquement l'objet `v` comme premier argument, mais la méthode n'attend aucun paramètre : `TypeError: klaxonner() takes 0 positional arguments but 1 was given`.
5. **Réponse C.** `__repr__` (ou `__str__`) est appelée par `print()` pour obtenir une représentation en chaîne.

Exercice 2 *Classe Rectangle (guidé)*

On souhaite créer une classe `Rectangle` pour représenter des rectangles géométriques.

Partie A. Compléter le code :

```python
class Rectangle:
    def __init__(self, longueur, largeur):
        self._________ = longueur
        self._________ = largeur

    def perimetre(self):
        return ________________________

    def surface(self):
```

```
        return ________________________

    def __repr__(self):
        return f"Rectangle({self.longueur}, {self.largeur})"
```

Partie B. Créer un rectangle de longueur 7 et de largeur 5, puis afficher son périmètre et sa surface.

Partie C (enrichissement). Créer une classe `Parallelepipede` qui hérite de `Rectangle` et possède un attribut `hauteur` et une méthode `volume()`.

Corrigé de l'exercice 2. Partie A.

```
class Rectangle:
    def __init__(self, longueur, largeur):
        self.longueur = longueur
        self.largeur = largeur

    def perimetre(self):
        return 2 * (self.longueur + self.largeur)

    def surface(self):
        return self.longueur * self.largeur

    def __repr__(self):
        return f"Rectangle({self.longueur}, {self.largeur})"
```

Partie B.

```
r = Rectangle(7, 5)
print(r.perimetre())  # 24
print(r.surface())    # 35
```

Partie C.

```
class Parallelepipede(Rectangle):
    def __init__(self, longueur, largeur, hauteur):
        Rectangle.__init__(self, longueur, largeur)
        self.hauteur = hauteur

    def volume(self):
        return self.longueur * self.largeur * self.hauteur
```

Remarque : l'héritage n'est pas exigible au baccalauréat.

Exercice 3 *Classe CompteBancaire (guidé)*

1. Écrire le constructeur avec les paramètres `numero_compte`, `nom` et `solde`.
2. Écrire `versement(montant)` qui ajoute `montant` au solde.
3. Écrire `retrait(montant)` qui retire `montant` du solde.
4. Écrire `agios()` qui applique des agios de 5 % (le solde est multiplié par 0.95).
5. Écrire `__repr__` qui retourne `"Compte n°1234 (Marcel) : 300 euros"`.

6. Tester : créer un compte n°1234 pour Marcel (solde 300), retrait de 500, versement de 2000, agios. Afficher après chaque opération.

Corrigé de l'exercice 3.

```python
class CompteBancaire:
    def __init__(self, numero_compte, nom, solde):
        self.numero = numero_compte
        self.nom = nom
        self.solde = solde

    def versement(self, montant):
        self.solde += montant

    def retrait(self, montant):
        self.solde -= montant

    def agios(self):
        self.solde = self.solde * 0.95

    def __repr__(self):
        return f"Compte n°{self.numero} ({self.nom}) : {self.solde} euros"
```

Test :

```python
c = CompteBancaire(1234, "Marcel", 300)
print(c)          # Compte n°1234 (Marcel) : 300 euros
c.retrait(500)
print(c)          # Compte n°1234 (Marcel) : -200 euros
c.versement(2000)
print(c)          # Compte n°1234 (Marcel) : 1800 euros
c.agios()
print(c)          # Compte n°1234 (Marcel) : 1710.0 euros
```

Exercice 4 *Classe Cercle*

On modélise un cercle C(O, r) de centre O(a, b) et de rayon r.

1. Définir la classe `Cercle` avec un constructeur prenant `a`, `b` et `r`.
2. Écrire `surface()` qui retourne πr^2.
3. Écrire `perimetre()` qui retourne $2\pi r$.
4. Écrire `contient(x, y)` qui retourne `True` si $\sqrt{(x-a)^2+(y-b)^2} \leqslant r$.
5. Tester avec C de centre $(1, 2)$ et de rayon 3. Le point $(2, 3)$ est-il à l'intérieur ? Et $(5, 5)$?

Corrigé de l'exercice 4.

```python
from math import pi, sqrt

class Cercle:
    def __init__(self, a, b, r):
        self.a = a
        self.b = b
```

```
        self.r = r

    def surface(self):
        return pi * self.r ** 2

    def perimetre(self):
        return 2 * pi * self.r

    def contient(self, x, y):
        distance = sqrt((x - self.a) ** 2 + (y - self.b) ** 2)
        return distance <= self.r
```

Pour $(2,3)$: $\sqrt{(2-1)^2+(3-2)^2} = \sqrt{2} \approx 1.41 \leqslant 3 \longrightarrow$ `True`.

Pour $(5,5)$: $\sqrt{(5-1)^2+(5-2)^2} = \sqrt{25} = 5 > 3 \longrightarrow$ `False`.

Exercice 5 *Classe Chronomètre*

1. Écrire le constructeur avec `h`, `m`, `s` (valeurs par défaut à 0).
2. Écrire `__repr__` au format `"02h15m30s"`.
3. Écrire `en_secondes()` qui retourne le temps total en secondes.
4. Écrire `avancer(s)` qui ajoute `s` secondes en gérant les dépassements.
5. Écrire `__add__` pour additionner deux chronomètres avec `+`.
6. Tester : créer `1h45m50s`, avancer de 75 secondes, additionner avec `0h30m20s`.

Corrigé de l'exercice 5.

```
class Chronometre:
    def __init__(self, h=0, m=0, s=0):
        self.h = h
        self.m = m
        self.s = s

    def __repr__(self):
        return f"{self.h:02d}h{self.m:02d}m{self.s:02d}s"

    def en_secondes(self):
        return self.h * 3600 + self.m * 60 + self.s

    def avancer(self, s):
        total = self.en_secondes() + s
        self.h = total // 3600
        self.m = (total % 3600) // 60
        self.s = total % 60

    def __add__(self, other):
        total = self.en_secondes() + other.en_secondes()
        h = total // 3600
        m = (total % 3600) // 60
        s = total % 60
        return Chronometre(h, m, s)
```

Test :

```python
c1 = Chronometre(1, 45, 50)
c1.avancer(75)
print(c1)            # 01h47m05s
c2 = Chronometre(0, 30, 20)
c3 = c1 + c2
print(c3)            # 02h17m25s
```

Vérification : $6350 + 75 = 6425$ s $\rightarrow$ 1 h 47 m 5 s. Puis $6425 + 1820 = 8245$ s $\rightarrow$ 2 h 17 m 25 s.

Exercice 6 *Classe myString (synthèse)*

Les chaînes Python (`str`) sont **immuables**. On crée une classe `myString` qui encapsule une chaîne et ajoute des méthodes de modification.

1. Constructeur stockant une chaîne dans un attribut `s`.
2. `__repr__` retournant la chaîne entourée de guillemets simples.
3. `append(texte)` qui concatène `texte` à la fin.
4. `pop(i)` qui supprime le caractère à l'indice `i` et le renvoie (comme `list.pop`).
5. `__len__` pour que `len(obj)` retourne la longueur.
6. `__eq__` pour comparer deux `myString` avec `==`.
7. Tester :

```python
s1 = myString("hello")
s1.append(" world")
print(s1)             # 'hello world'
print(len(s1))        # 11
s1.pop(5)
print(s1)             # 'helloworld'
s2 = myString("helloworld")
print(s1 == s2)       # True
```

Corrigé de l'exercice 6.

```python
class myString:
    def __init__(self, s):
        self.s = s

    def __repr__(self):
        return f"'{self.s}'"

    def append(self, texte):
        self.s = self.s + texte

    def pop(self, i):
        caractere = self.s[i]
        self.s = self.s[:i] + self.s[i + 1:]
        return caractere

    def __len__(self):
        return len(self.s)

    def __eq__(self, other):
```

```
        if isinstance(other, myString):
            return self.s == other.s
        return False
```

`append` modifie l'attribut `self.s` par concaténation. `pop` reconstruit la chaîne sans le caractère à l'indice donné. `__eq__` vérifie d'abord que `other` est un `myString` avec `isinstance`. Cet exercice illustre le principe d'**encapsulation**.