Bing

Bonjour, je suis ravi de vous aider à évaluer votre niveau de maîtrise de la POO en Python. Voici un QCM de 5 questions de type output code python, de niveau difficile. Pour chaque question, je vous donne la réponse ainsi qu'un feedback sur les pièges à éviter.

Q1. Quel est le résultat de l'exécution du code suivant ?

[python]

class Animal:

def \_\_init\_\_(self, nom):

self.nom = nom

def parler(self):

print("Je suis un animal")

class Chien(Animal):

def \_\_init\_\_(self, nom, race):

super().\_\_init\_\_(nom)

self.race = race

def parler(self):

print("Je suis un chien")

rex = Chien("Rex", "Berger allemand")

rex.parler()

A) Je suis un animal

B) Je suis un chien

C) Rex

D) Berger allemand

Réponse: B) Je suis un chien

Feedback: Le code utilise le concept d'héritage, qui permet à une classe fille (Chien) d'hériter des attributs et des méthodes d'une classe mère (Animal). La classe Chien redéfinit la méthode parler() de la classe Animal, ce qui signifie qu'elle remplace le comportement hérité par un comportement spécifique à la classe Chien. Ainsi, lorsque l'on appelle la méthode parler() sur une instance de la classe Chien, c'est la méthode redéfinie qui est exécutée, et non la méthode héritée. Il faut donc faire attention à ne pas confondre les méthodes héritées et les méthodes redéfinies.

Q2. Quel est le résultat de l'exécution du code suivant ?

[python]

class A:

def \_\_init\_\_(self):

self.x = 1

def f(self):

self.x += 1

return self.x

class B(A):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.y = 2

def f(self):

self.y += 1

return super().f() + self.y

b = B()

print(b.f())

A) 3

B) 4

C) 5

D) 6

Réponse: D) 6

Feedback: Le code utilise également le concept d'héritage, mais cette fois-ci, la classe fille (B) appelle la méthode f() de la classe mère (A) à l'aide de la fonction super(). Cela permet d'exécuter le comportement hérité en plus du comportement spécifique à la classe B. Ainsi, lorsque l'on appelle la méthode f() sur une instance de la classe B, on incrémente d'abord l'attribut y de 1, puis on appelle la méthode f() de la classe A, qui incrémente l'attribut x de 1 et renvoie sa valeur. On ajoute ensuite la valeur de y à la valeur renvoyée par super().f(), ce qui donne 6. Il faut donc faire attention à bien comprendre le fonctionnement de la fonction super(), qui permet d'accéder aux méthodes héritées.

Q3. Quel est le résultat de l'exécution du code suivant ?

[python]

class Point:

def \_\_init\_\_(self, x, y):

self.x = x

self.y = y

def \_\_add\_\_(self, other):

return Point(self.x + other.x, self.y + other.y)

def \_\_str\_\_(self):

return f"({self.x}, {self.y})"

p1 = Point(1, 2)

p2 = Point(3, 4)

p3 = p1 + p2

print(p3)

A) (4, 6)

B) (1, 2) + (3, 4)

C) TypeError: unsupported operand type(s) for +: 'Point' and 'Point'

D) None

Réponse: A) (4, 6)

Feedback: Le code utilise le concept de surcharge d'opérateurs, qui permet de définir le comportement des opérateurs (+, -, \*, /, etc.) pour des objets personnalisés. La classe Point définit la méthode spéciale add() qui prend en paramètre un autre objet Point et renvoie un nouvel objet Point dont les coordonnées sont la somme des coordonnées des deux objets. La classe Point définit également la méthode spéciale str() qui renvoie une représentation sous forme de chaîne de caractères d'un objet Point. Ainsi, lorsque l'on utilise l'opérateur + sur deux objets Point, on appelle la méthode add() qui renvoie un nouvel objet Point, et lorsque l'on utilise la fonction print() sur un objet Point, on appelle la méthode str() qui renvoie une chaîne de caractères. Il faut donc faire attention à bien définir les méthodes spéciales qui permettent de surcharger les opérateurs pour des objets personnalisés.

Q4. Quel est le résultat de l'exécution du code suivant ?

[python]

class Singleton:

\_instance = None

def \_\_new\_\_(cls, \*args, \*\*kwargs):

if cls.\_instance is None:

cls.\_instance = super().\_\_new\_\_(cls, \*args, \*\*kwargs)

return cls.\_instance

s1 = Singleton()

s2 = Singleton()

print(s1 is s2)

A) True

B) False

C) None

D) AttributeError: 'Singleton' object has no attribute '\_instance'

Réponse: A) True

Feedback: Le code utilise le concept de singleton, qui est un patron de conception qui garantit qu'une classe n'a qu'une seule instance et qu'elle est accessible globalement. La classe Singleton définit la méthode spéciale new() qui est appelée lors de la création d'un nouvel objet. Cette méthode vérifie si l'attribut de classe \_instance est None, ce qui signifie qu'aucune instance n'a été créée auparavant. Si c'est le cas, elle appelle la méthode new() de la classe mère (object) qui crée une nouvelle instance et l'affecte à l'attribut \_instance. Sinon, elle renvoie simplement l'attribut \_instance qui contient l'instance unique. Ainsi, lorsque l'on crée deux objets Singleton, on obtient en fait la même instance, ce qui fait que l'opérateur is renvoie True. Il faut donc faire attention à bien comprendre le fonctionnement de la méthode new(), qui permet de contrôler la création des objets.

Q5. Quel est le résultat de l'exécution du code suivant ?

[python]

class Personne:

def \_\_init\_\_(self, nom):

self.nom = nom

def \_\_repr\_\_(self):

return f"Personne({self.nom})"

def \_\_eq\_\_(self, other):

return self.nom == other.nom

p1 = Personne("Alice")

p2 = Personne("Bob")

p3 = Personne("Alice")

liste = [p1, p2]

print(p3 in liste)

A) True

B) False

C) None

D) AttributeError: 'Personne' object has no attribute 'nom'

Réponse: A) True

Feedback: Le code utilise également le concept de surcharge d'opérateurs, mais cette fois-ci pour l'opérateur in, qui vérifie si un élément appartient à une séquence (liste, tuple, chaîne, etc.). La classe Personne définit la méthode spéciale repr() qui renvoie une représentation sous forme de chaîne de caractères d'un objet Personne. La classe Personne définit également la méthode spéciale eq() qui prend en paramètre un autre objet Personne et renvoie True si leurs attributs nom sont égaux, et False sinon. Ainsi, lorsque l'on utilise l'opérateur in sur un objet Personne et une liste d'objets Personne, on compare les objets selon leur attribut nom et non selon leur identité. C'est pourquoi p3 in liste renvoie True, car il existe un objet dans la liste qui a le même nom que p3. Il faut donc faire attention à bien définir les méthodes spéciales qui permettent de comparer les objets selon des critères personnalisés.

J'espère que ce QCM vous a été utile pour évaluer votre niveau de maîtrise de la POO en Python. N'hésitez pas à me solliciter si vous avez besoin d'aide pour améliorer votre code ou pour apprendre d'autres concepts. Bon courage ! 😊