Bing

D'accord, voici un autre QCM de 5 questions de type output code python, de niveau plus 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 A:

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

self.x = x

def \_\_call\_\_(self, y):

return self.x + y

a = A(2)

print(a(3))

A) 5

B) 6

C) TypeError: 'A' object is not callable

D) None

Réponse: A) 5

Feedback: Le code utilise le concept de classe callable, qui est une classe dont les instances peuvent être appelées comme des fonctions. La classe A définit la méthode spéciale call() qui prend en paramètre un argument y et renvoie la somme de l'attribut x de l'instance et de y. Ainsi, lorsque l'on crée une instance de la classe A avec x = 2, on obtient un objet callable qui peut être appelé avec un argument y = 3, ce qui renvoie 5. Il faut donc faire attention à bien définir la méthode spéciale call() qui permet de rendre une classe callable.

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

[python]

class B:

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

self.\_\_x = 1

def get\_x(self):

return self.\_\_x

def set\_x(self, value):

self.\_\_x = value

b = B()

b.set\_x(2)

print(b.get\_x())

print(b.\_\_x)

A) 2

2

B) 2

AttributeError: 'B' object has no attribute '\_\_x'

C) 1

1

D) 1

AttributeError: 'B' object has no attribute '\_\_x'

Réponse: B) 2

AttributeError: 'B' object has no attribute '\_\_x'

Feedback: Le code utilise le concept d'encapsulation, qui est un mécanisme qui permet de protéger les attributs et les méthodes d'une classe en les rendant inaccessibles depuis l'extérieur. La classe B définit un attribut privé \_\_x qui ne peut être modifié ou lu que par les méthodes de la classe B. La classe B définit également deux méthodes publiques get\_x() et set\_x() qui permettent respectivement de renvoyer et de modifier la valeur de l'attribut \_\_x. Ainsi, lorsque l'on crée une instance de la classe B, on peut utiliser les méthodes get\_x() et set\_x() pour manipuler l'attribut \_\_x, mais on ne peut pas accéder directement à l'attribut \_\_x depuis l'extérieur de la classe B. C'est pourquoi b.get\_x() renvoie 2, car on a modifié la valeur de l'attribut \_\_x avec b.set\_x(2), mais b.\_\_x provoque une erreur AttributeError, car on essaie d'accéder à un attribut privé. Il faut donc faire attention à bien respecter le niveau d'accès des attributs et des méthodes d'une classe.

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

[python]

class C:

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

self.x = [1, 2, 3]

def \_\_getitem\_\_(self, index):

return self.x[index]

def \_\_setitem\_\_(self, index, value):

self.x[index] = value

c = C()

c[0] = 4

print(c[1])

A) 4

B) 2

C) TypeError: 'C' object does not support item assignment

D) None

Réponse: B) 2

Feedback: Le code utilise le concept d'indexation, qui est un mécanisme qui permet d'accéder aux éléments d'un objet séquentiel (liste, tuple, chaîne, etc.) à l'aide d'un indice entre crochets. La classe C définit un attribut x qui est une liste contenant les éléments [1, 2, 3]. La classe C définit également les méthodes spéciales getitem() et setitem() qui prennent en paramètre un indice et renvoient ou modifient respectivement la valeur de l'élément correspondant dans la liste x. Ainsi, lorsque l'on crée une instance de la classe C, on peut utiliser l'indexation pour accéder ou modifier les éléments de la liste x, comme si c était une liste. C'est pourquoi c[0] = 4 modifie le premier élément de la liste x, et c[1] renvoie le deuxième élément de la liste x, qui vaut 2. Il faut donc faire attention à bien définir les méthodes spéciales qui permettent d'implémenter l'indexation pour des objets séquentiels.

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

[python]

class D:

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

self.x = x

def \_\_iter\_\_(self):

return self

def \_\_next\_\_(self):

if self.x > 0:

self.x -= 1

return self.x

else:

raise StopIteration

d = D(3)

for i in d:

print(i)

A) 3

2

1

B) 2

1

0

C) 3

2

1

0

D) StopIteration

Réponse: B) 2

1

0

Feedback: Le code utilise le concept d'itérateur, qui est un objet qui permet de parcourir les éléments d'un objet itérable (liste, tuple, chaîne, etc.) à l'aide d'une boucle for. La classe D définit un attribut x qui est un entier positif. La classe D définit également les méthodes spéciales iter() et next() qui permettent de rendre la classe D itérable. La méthode iter() renvoie l'objet lui-même, ce qui signifie qu'il est à la fois itérable et itérateur. La méthode next() renvoie le prochain élément de l'itération, qui est la valeur de l'attribut x décrémentée de 1, ou lève une exception StopIteration si l'attribut x est nul ou négatif, ce qui signifie qu'il n'y a plus d'éléments à parcourir. Ainsi, lorsque l'on crée une instance de la classe D avec x = 3, on obtient un objet itérable qui peut être parcouru avec une boucle for. C'est pourquoi for i in d affiche successivement les valeurs de l'attribut x jusqu'à ce qu'il atteigne 0, ce qui provoque l'arrêt de la boucle. Il faut donc faire attention à bien définir les méthodes spéciales qui permettent de rendre une classe itérable.

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

[python]

class E:

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

self.\_\_x = 1

@property

def x(self):

return self.\_\_x

@x.setter

def x(self, value):

if value > 0:

self.\_\_x = value

else:

raise ValueError("x must be positive")

e = E()

e.x = -1

print(e.x)

A) -1

B) 1

C) ValueError: x must be positive

D) AttributeError: can't set attribute

Réponse: C) ValueError: x must be positive

Feedback: Le code utilise le concept de propriété, qui est un décorateur qui permet de définir des méthodes qui se comportent comme des attributs. La classe E définit un attribut privé \_\_x qui vaut initialement 1. La classe E définit également une propriété x qui est décorée par le décorateur @property. Cette propriété renvoie la valeur de l'attribut \_\_x lorsqu'on y accède en lecture. La classe E définit également un setter x qui est décoré par le décorateur @x.setter. Ce setter modifie la valeur de l'attribut \_\_x lorsqu'on y affecte une valeur en écriture, mais seulement si cette valeur est positive, sinon il lève une exception ValueError. Ainsi, lorsque l'on crée une instance de la classe E, on peut utiliser la propriété x comme si c'était un attribut public, mais en réalité on appelle les méthodes définies par la propriété et le setter. C'est pourquoi e.x = -1 provoque une erreur ValueError, car on essaie d'affecter une valeur négative à la propriété x, ce qui n'est pas autorisé par le setter. Il faut donc faire attention à bien comprendre le fonctionnement des propriétés, qui permettent de contrôler l'accès aux attrib