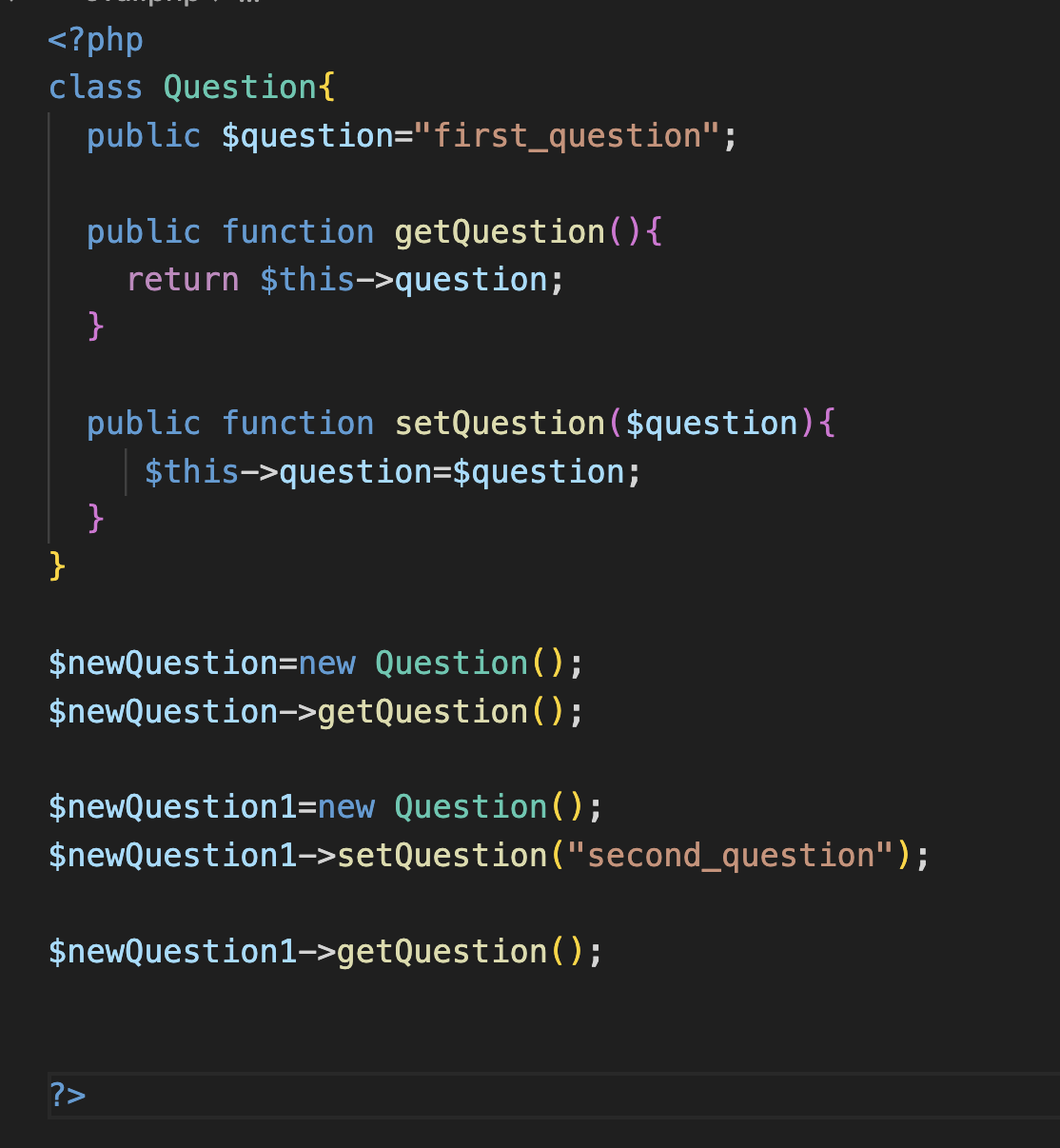
Évaluation Poo/MVC Alt-Rh



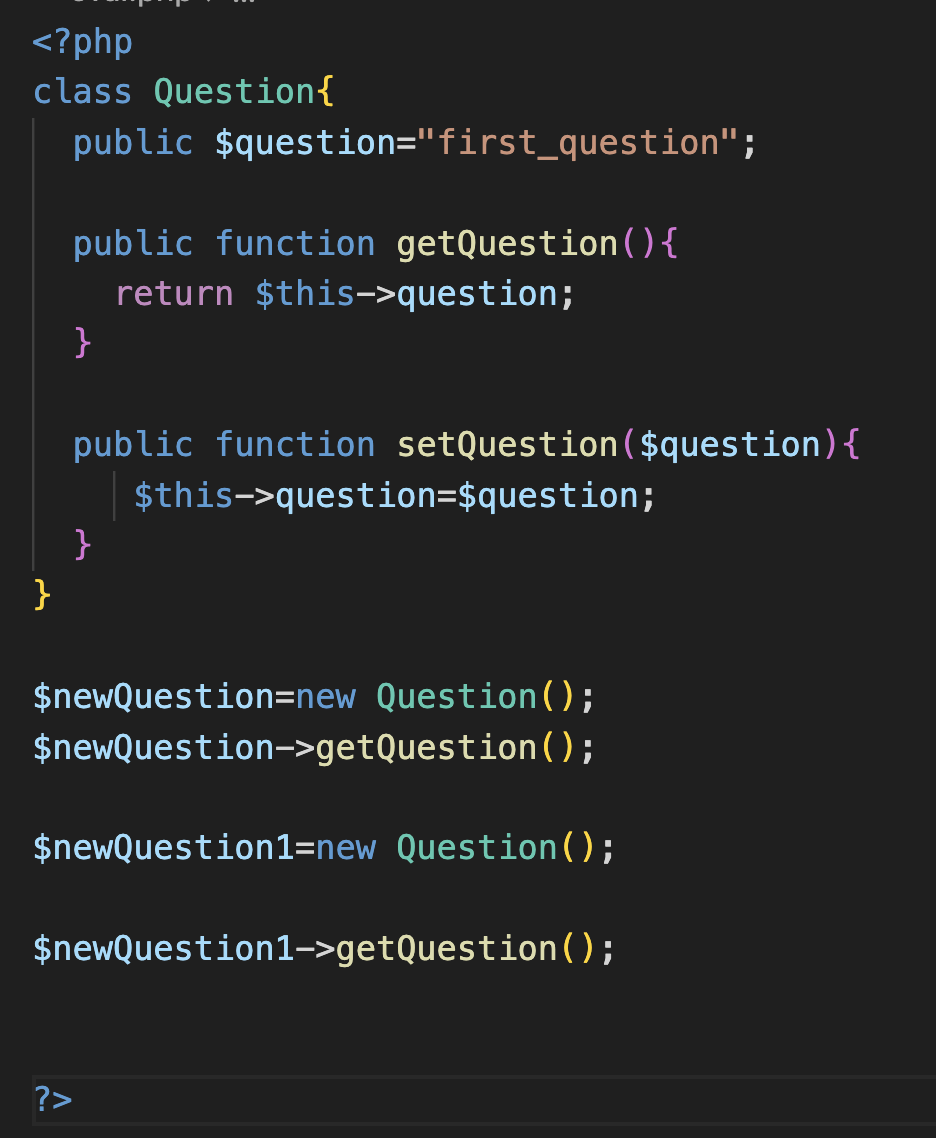
En vous inspirant de la capture d’écran au-dessus, répondez au questions suivants

1. Lors de la première appel a ma méthode getQuestion, $this va correspondre :
   1. A null
   2. A mon instance $newQuestion
   3. A mon instance $newQuestion1
   4. A la classe elle même

1. Lors de ma premiere appel a ma methode getQuestion, si j’en fait un echo, je vais voir s’afficher
   1. Une chaine de caractère « first\_question »
   2. Une chaine de caractère « second\_question »
   3. Null

1. Lors de ma 2eme appel a la methode getQuestion, si j’en fait un echo, je vais voir s’afficher
   1. Une chaine de caractère « first\_question »
   2. Une chaine de caractère « second\_question »
   3. Null

J’ai maintenant retiré mon appel à la méthode setQuesstion, dans ce cas



1. La valeur de retour de le premier appel a getQuestion sera identique au deuxième appel
   1. Oui
   2. Non
2. Lors de la première appel a la méthode getQuestion mon $this va correspondre a la même valeur que lors de ma 2eme appel (dit autrement, est ce que la valeur de $this sera identique dans les deux appels ?)
   1. Oui
   2. Non

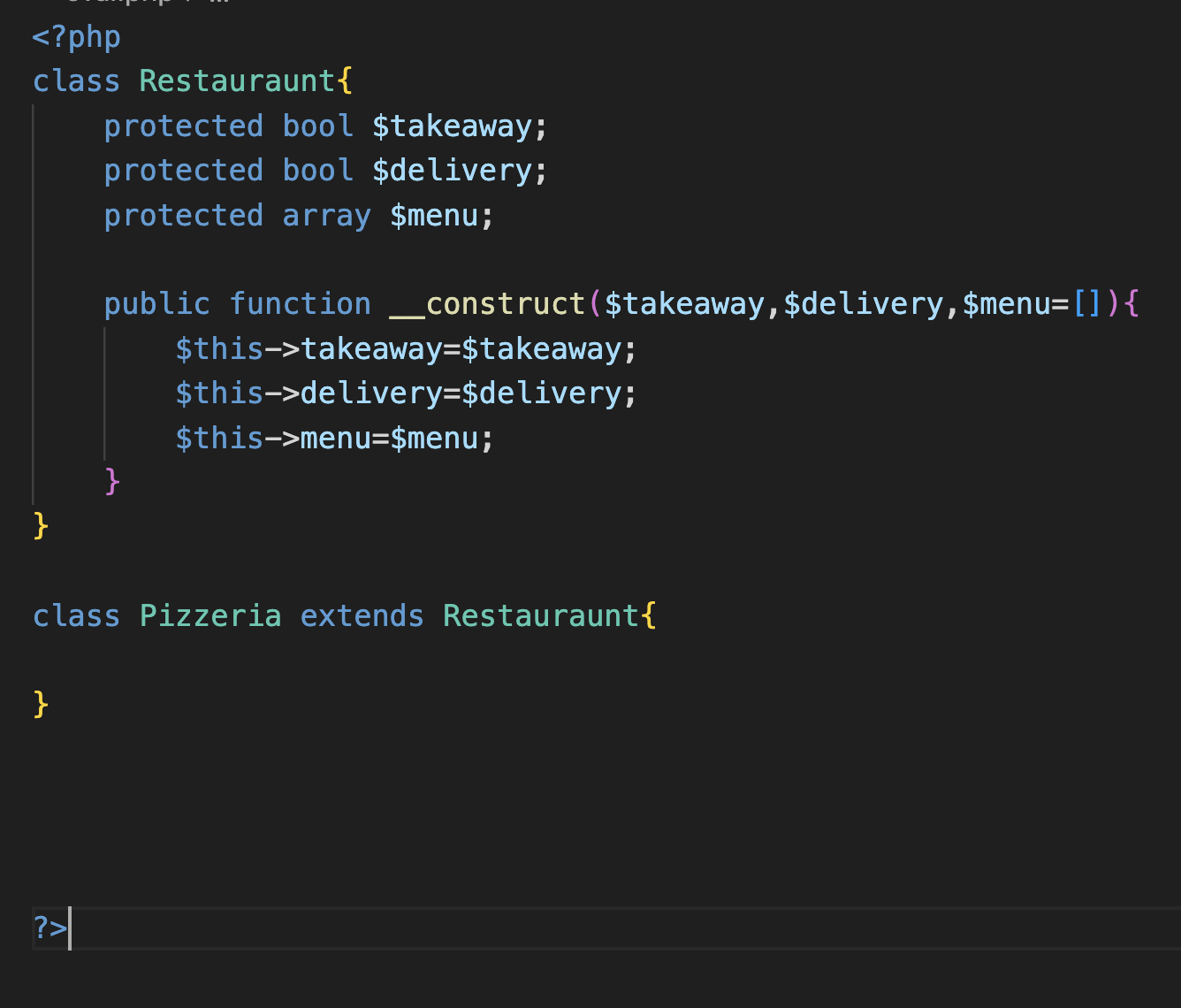
En vous inspirant du capture d’écran (scroll un peu), répondez aux questions suivantes



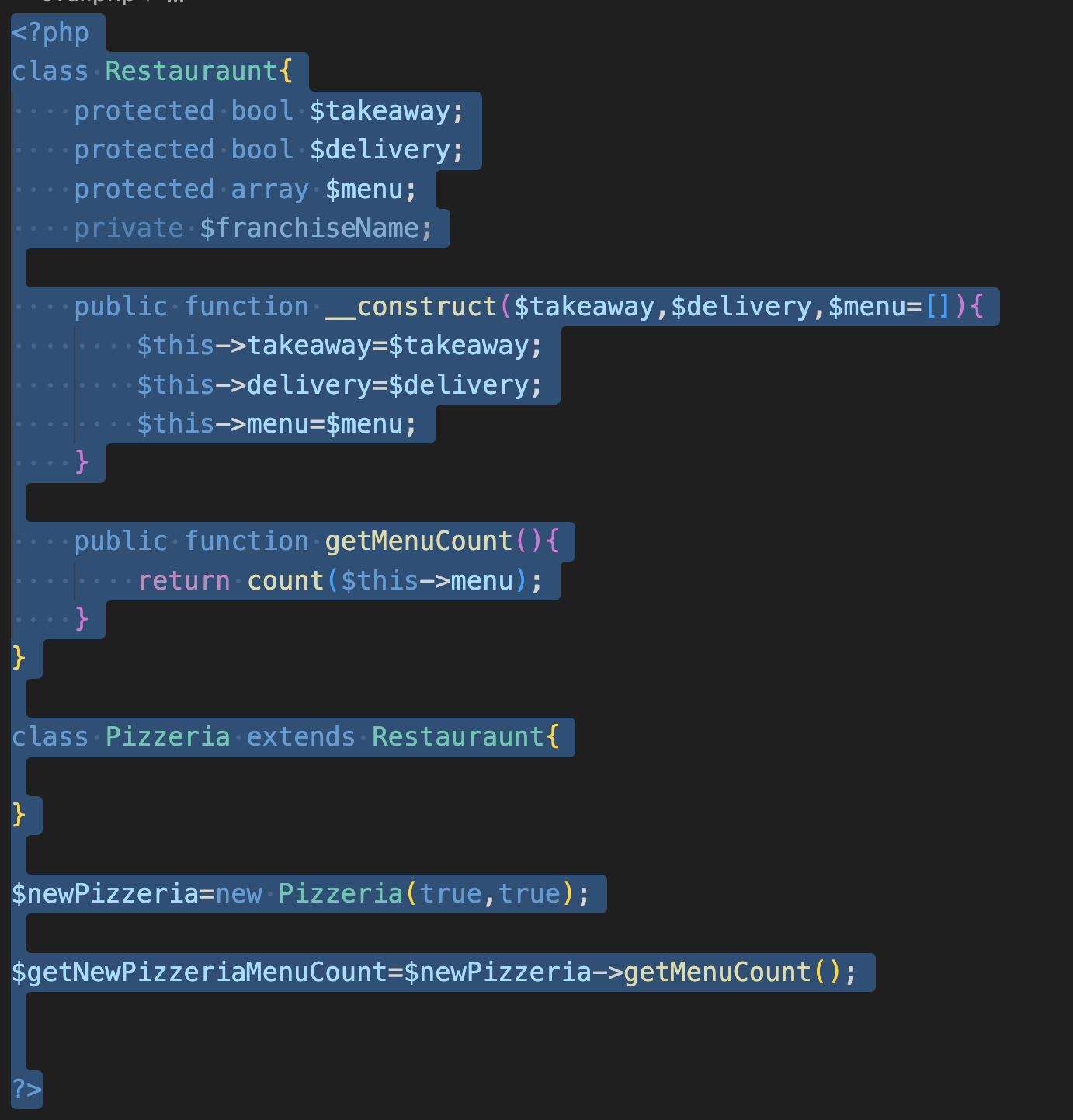
:

1. Lors que j’instancie Football sans mettre d’argument, une erreur va se produire
   1. Oui
   2. Non

1. Si je créé une méthode dans la classe Football, par défaut, elle sera accessible dans la classe Sport.
   1. Oui
   2. Non
2. Pour pouvoir faire appel au constructeur de Sport dans le constructeur que j’ai defini dans Football, il suffit de faire appel au constructeur de mon parent sans rien faire d'autre
   1. Oui
   2. Non
3. Si je fais un echo de $football->getName() je vais voir s’afficher « My\_Sport » a l’ecran.
4. Oui
5. Non



1. Ici si j’instancie ma classe Restaurant sans préciser d’arguments, je vais avoir une erreur
   1. Oui
   2. Non

Maintenant je rajoute qq petits modifications

1. Ce code que je vient de mettre est susceptible de déclencher des erreurs
   1. Oui
   2. Non

1. La valeur $getNewPizzeraiMenuCount sera :
   1. Null
   2. Zero
   3. Une erreur car la methode getMenuCount n’appartient pas a la classe Pizzeria
2. Ici ma classe Pizzeria pourra utiliser **toutes** les proprietés de la classe Restaurant
   1. Oui
   2. Non
3. Pourquoi est ce que le fait d’écrire un setter est plus sécure que de modifier une propriété directement ?
   1. Le mot clé set dans un setter est magique
   2. Les méthodes sont par défaut plus sécure que les propriétés même si on n’y rajoute aucun contrôle
   3. Un setter est une méthode et dans une méthode nous pouvons rajouter des contrôles pour nous assurer que nous ne mettons pas n’importe quoi
   4. Aucune des réponses
4. J’ai une propriété que je veux initialiser lors de mon instanciation et ensuite je ne veux plus y toucher. J’ai néanmoins la liberté d’affecter la valeur que je souhaite a cette propriété mais je ne veux que le faire une seul fois (lors de l’instanciation) . Dans ce cas il sera recommandé de créer un setter pour cette propriété ?
   1. Oui
   2. Non
5. J’ai une classe Tree qui représente une table trees dans ma base de données. J’ai une méthode pour récupérer tous les « trees » dans ma base de données. Répondez aux deux affirmations suivantes.

1)Cette méthode me retournera une valeur différente selon l’instance eventuelle qui y fait appel.

2)Il serai une bonne idée de passer cette méthode en statique

* + 1. Non et oui
    2. Oui et non
    3. Oui et oui
    4. Non et non

1. Dans nos minis projets que nous avons vu avec Altoouter, toutes nos lignes de code vont réellement s’exécuter dans combien de fichiers ?
   1. 1
   2. 2
   3. 3
   4. 4



1. Comment s’appelle réellement ma classe ici ?
   1. World
   2. hello\World
   3. hello
2. Nous avons maintenant la possibilité d’instancier une classe sans avoir a faire un require\_once auparavant du fichier et nous n’allons pas directement tomber dans une erreur, ce mécanisme est possible grâce a une fonction qui s’appelle :
   1. spl\_autoload\_register qui prend en parametre un callback et c’est dans ce callback que nous allons réellement devoir écrire un require\_once en nous inspirant du nom complet de notre classe qui est fournie en tant que parametre de notre callback
   2. spl\_autoload\_register qui ne prends aucun paramètre mais par magie, il sait ou aller !
   3. Mon constructeur va automatiquement faire le require
3. Si je mets ma classe que j’ai appelé World et qui se trouve dans le namespace hello dans un dossier qui s’apelle goodbye et dans un fichier qui s’apelle Mars.php ca marchera quand même
   1. Oui car spl\_autoload\_register est très flexible
   2. Non car notre callback passé en spl\_autoload\_register est censé construire le chemin correspondant en fonction du nom complet de notre classe
   3. Ca dépend, si on écrit la classe dans un fichier qui s’appelle Jupiter.php, l’ordinateur sera content car il n’a jamais visité la planète Jupiter. Il sera donc de bonne humeur