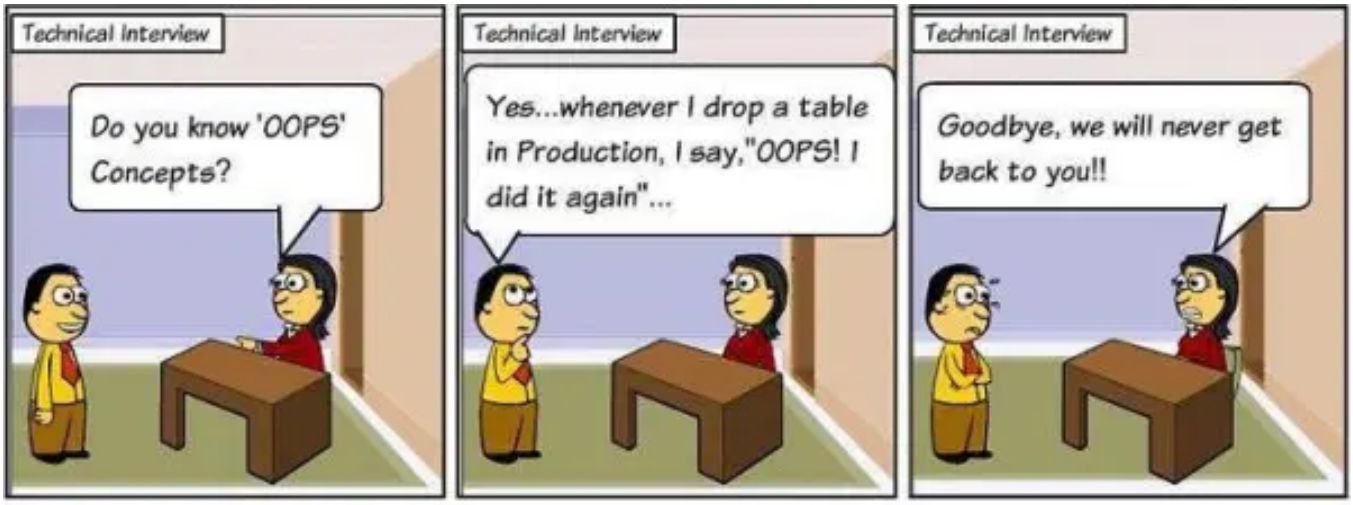
La Plateforme

Jour 1 : Les Classes

La programmation c’est class

Simplifiez votre code, organisez vos idées, structurez vos programmes

Jusqu'à aujourd'hui, vous avez développé vos programmes au moyen de la programmation procédurale. La résolution de vos problèmes se faisait par un ensemble d’instructions qui se découpait en plusieurs actions afin de résoudre votre problème. Cette méthode est intuitive lorsque l’on apprend à développer, mais vous avez dû vous rendre compte que cette méthode pose un certain nombre d'inconvénients, votre code est long, fastidieux et difficile à maintenir.
La POO (programmation orienté objet) ne permet pas de faire plus de choses, mais elle permet toutefois d'organiser son code de façon claire et structurée , elle facilite le travail coopératif et rend son code maintenable et réutilisable.



Job 1

Créer une classe “**Operation**” avec un constructeur (méthode qui sera appelée lors de l’instance de la classe). Ajouter les attributs “nombre1” et “nombre2” initialisés avec des valeurs par défaut. Instancier votre première classe et imprimer l’objet en console.

Résultat attendu :

```
<__main__.Operation object at 0x000002541F869CF0>
```

Job 2

À l’aide de la classe “**Operation**” crée au-dessus, imprimer en console la valeur des attributs “nombre1” et “nombre2”.

Résultat attendu :

```
Le nombre1 est 12
Le nombre2 est 3
```

Job 3

Modifier votre classe “**Operation**” afin d’y ajouter la méthode **addition**. Cette méthode additionne “nombre1” et “nombre2” et affiche en console le résultat.

Job 4

Créez une classe “Personne” avec les attributs “nom” et “prenom”. Ajoutez une méthode “**SePresenter()**” qui retourne le nom et le prénom de la personne. Ajoutez aussi un constructeur prenant en paramètres de quoi donner des valeurs initiales aux attributs “nom” et “prenom”. Instanciez plusieurs personnes avec les valeurs de construction de votre choix et faites appel à la méthode “**SePresenter()**” afin de vérifier que tout fonctionne correctement.

Résultat attendu :

```
Je suis John Doe
Je suis Jean Dupond
```

Job 5

Créez une classe **Animal** avec un attribut age initialisé à zéro et prénom initialisé vide dans son constructeur.

Instancier un objet Animal et écrire en console l’attribut âge. Créer une méthode “**vieillir**” qui ajoute 1 à l'âge de l’animal. Faire vieillir votre animal et afficher son âge en console.

Résultat attendu :

```
L'age de l'animal 0 ans
# Age de l'animal apres appel de la methode viellir
L'age de l'animal 1 ans
```

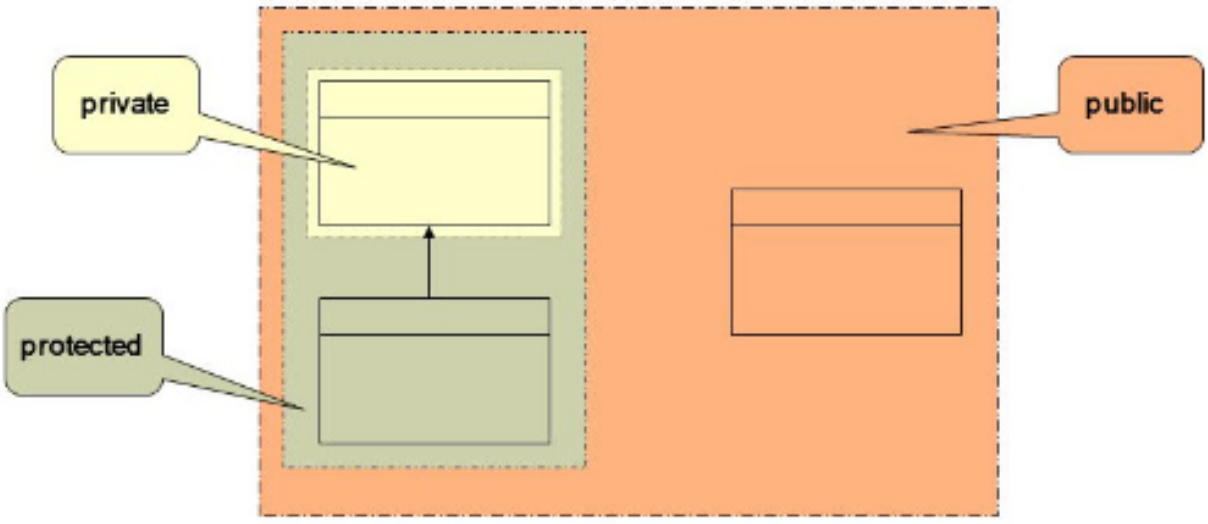
Créer une méthode ”**nommer**” qui prend en paramètre le nom de l’animal. Nommer votre animal et afficher en console le nom de l’animal.

Résultat attendu :

```
L'animal se nomme Luna
```

Il est temps de compartimenter nos données !

En programmation, il existe une façon de garantir l'intégrité des données contenues dans l'objet, l’encapsulation.
Elle permet de définir des niveaux de visibilité des éléments de la classe. Ces niveaux de visibilité définissent les droits d’accès aux données selon que l’on y accède par une méthode de la classe elle-même, d’une classe héritière, ou bien d’une classe quelconque.



Job 6

Créer une classe “**Rectangle**” avec des attributs privés, “**longueur**” et “**largeur**” initialisé dans le constructeur.
Créer des **assesseurs** et **mutateurs** afin de pouvoir afficher et modifier les attributs de la classe.

Créer un rectangle avec les valeurs suivantes : longueur 10 et largeur 5. Changer la valeur de la longueur et de la largeur et vérifier que les modifications soient bien prises en compte.

Job 7

Créer la classe '**Livre**’ qui prend en attributs **privés** un titre, un auteur et un nombre de pages.
Créer les **assesseurs** et **mutateurs** nécessaires afin de pouvoir modifier et afficher les attributs. Pour changer le nombre de pages, le nombre doit être un nombre entier positif sinon la valeur n’est pas changée et un message d’erreur est affiché.

Job 8

Récupérer la classe “**Livre**”.

Ajouter l'attribut **privé** suivant :

- “**disponible**” initialisé par défaut a True.

Créer une méthode nommée “**vérification**” qui vérifie si le livre est disponible, c'est-à-dire que la valeur de son attribut disponible est égale à True. Cette méthode doit retourner True ou False.

Ajouter une méthode “**emprunter**” qui rend le livre indisponible, autrement dit la valeur de “**disponible**” devient False. Bien sûr, pour pouvoir emprunter un livre, il faut que

celui-ci soit disponible, vérifier donc que celui-ci soit disponible sans utiliser l'attribut "**disponible**".

Après avoir emprunté un livre, il faut pouvoir le rendre. Créer une méthode "**rendre**" qui dans un premier temps vérifie si le livre a bien été emprunter (sans utiliser l'attribut "disponible"), puis change la valeur de l'attribut "disponible".

Job 9

Créer une classe nommée "**Student**" qui a comme attributs **privés** un nom, un prénom, un numéro d'étudiants et un nombre de crédits par défaut à zéro. La méthode "**add_credits**" permet d'ajouter un nombre de crédits au total de celui de l'étudiant. Ce **mutateur** doit s'assurer que la valeur passée en argument est supérieure à zéro pour garantir l'intégrité de la valeur.

Instancier un objet représentant l'étudiant John Doe qui possède le numéro d'étudiant 145. Ajoutez-lui des crédits à trois reprises et imprimer son total de crédits en console.

Résultat attendu :

```
Le nombre de credits de John Doe est de 30 points
```

Pour des mesures de confidentialité, l'établissement ne souhaite pas divulguer la façon dont elle évalue le niveau de ses étudiants. Pour répondre à cette problématique, créer une méthode nommée "**studentEval**" qui sera **privé**. Cette méthode retourne "Excellent" si le nombre de crédits est égale ou supérieure à 90, "Très bien." si le nombre est supérieur ou égal à 80, "Bien" si le nombre est supérieur ou égale à 70, "Passable" si égale ou supérieure à 60 et "insuffisant" si inférieur à 60.
Ajouter l'attribut **privé** nommé "**level**" dans le constructeur de la classe "**Student**" qui prend en valeur la méthode "**studentEval**". Créer une méthode **studentInfo** qui écrit en console les informations de l'étudiant (nom, prénom, identifiant et son niveau).

Résultat attendu :

```
Nom = John
Prénom = Doe
id = 145
Niveau = Bien
```

Job 10

Créer une classe “**Voiture**” qui a pour attributs **privés** une marque, un modèle , une année, un kilométrage et un attribut nommé “**en_marche**” initialisé à défaut sur False.

Cette classe doit posséder des mutateurs et assesseur pour chaque attribut.

Créer une méthode “**demarrer**” qui change la valeur de l’attribut “**en_marche**” en True, une méthode “**arreter**” qui change la valeur de l’attribut “**en_marche**” en False.

Ajouter à la classe “Voiture” l’attribut **privé** “**reservoir**” qui est initialisé à **50** par défaut dans le constructeur. Créer une méthode **privée** “**verifier_plein**” qui retourne la valeur de l’attribut “**reservoir**”. Cette méthode est appelée dans la méthode “**demarrer**” . Si la valeur du reservoir est supérieur à **5** la voiture peut demarrer.

Sur vos scripts doit apparaître l’ensemble des méthodes appelées tout au long des exercices.

Rendu

Le projet est à rendre sur https://github.com/prenom-nom/**runtrack-python-poo**. Pour chaque jour, créer un dossier nommé “jourXX” et pour chaque job, créer un fichier “jobXX” ou XX est le numéro du job.
N’oubliez pas d’envoyer vos modifications dès qu’une étape est avancée ou terminée et utilisez des commentaires explicites.
Pensez à mettre vos repository en public !

Compétences visées

- Maîtriser l’architecture POO en Python

Base de connaissances

- Les classes - Documentation officielle
- Apprenez la programmation orientée objet avec Python
- Tutoriel Class python
- Class python
- Passage de référence
- L'héritage