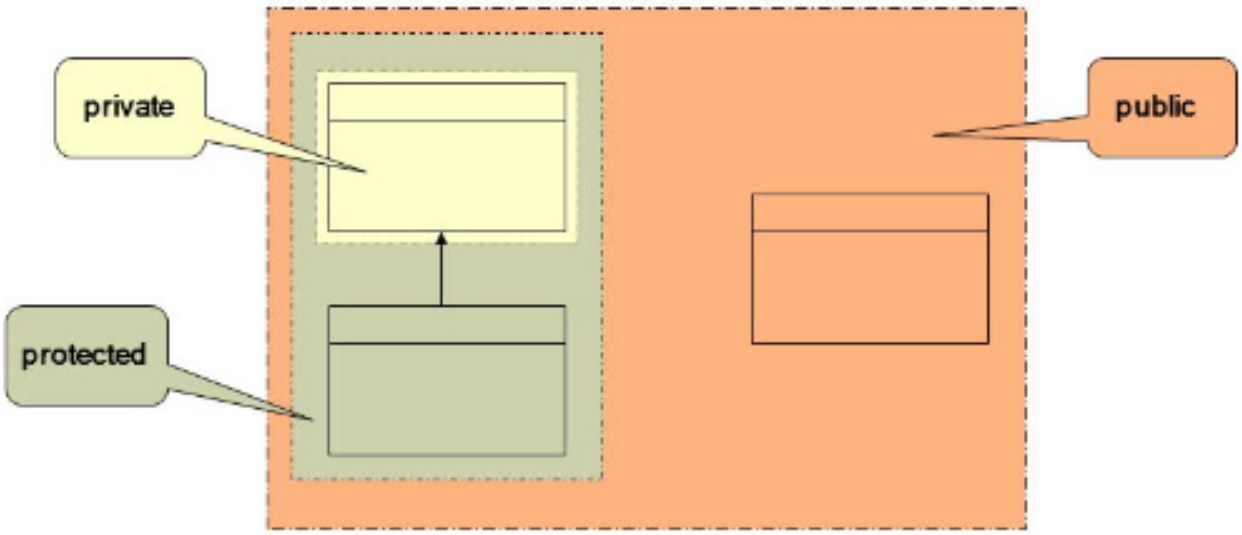
La Plateforme

Jour 2 : Encapsulation / Abstraction

La programmation c'est class

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 1

Créer une classe **Rectangle** avec des attributs privés, **longueur** et **largeur** initialisées 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 2

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, Ce dernier doit être un nombre entier positif, sinon la valeur n'est pas changée et un message d'erreur est affiché.

Job 3

Récupérer la classe **Livre**.

Ajouter l'attribut **privé** suivant :

- **disponible** est initialisé par défaut à 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, vérifiez 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 emprunté (sans utiliser l'attribut disponible), puis change la valeur de l'attribut disponible.

Job 4

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 imprimez 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 **privée** Cette méthode retourne "**Excellent**" si le nombre de crédits est égal ou supérieur 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 5

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

Cette classe doit posséder des mutateurs et assesseurs 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.

Ajoutez 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 réservoir est supérieure à **5** la voiture peut démarrer.

Job 6

Créer une classe **Commande** avec les attributs **privés,** numéro de commande, liste de plats commandés et statut de la commande (en cours, terminée ou annulée). Ajouter des méthodes permettant d'ajouter un plat à la commande , annuler une commande, calculer le total d'une commande qui doit être en **privé** et afficher une commande avec son total à payer, ainsi qu'une méthode permettant de calculer la TVA. Utiliser l'encapsulation et l'abstraction pour créer cette classe de manière que les attributs ne puissent pas être modifiés de l'extérieur. La liste des plats commandés doit être représentée sous forme de dictionnaire avec les noms des plats, le prix ainsi que le statut de la commande.

Sur votre script doit apparaître l'ensemble des méthodes appelées tout au long de cet exercice.

Rendu

Le projet est à rendre sur https://github.com/prenom-nom/**runtrack-python-poo**. Pour chaque jour, créer un dossier nommé "**jourXX**" ou **XX** est le numéro du jour 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.

Compétences visées

- Maîtriser l’architecture POO en Python
- Maîtriser l’encapsulation
- Maîtriser l’abstraction

Base de connaissances

- Les classes - Documentation officielle
- Apprenez la programmation orientée objet avec Python
- L’abstraction
- L’encapsulation