6/10

Samuel SORIN
contact@samuelsorin.fr

Programmation Orientée Objet (POO)

POO - introduction *(0_python_vs_php)*

La Programmation Orientée Objet est un **Style de programmation** qui permet de regrouper au même endroit le comportement (les fonctions) et les données (les structures) qui sont faites pour aller ensemble.

Permet de faire de belles API, de créer du code réutilisable et facile à manipuler.

C’est surtout bien pour ceux qui réutiliseront le code plus tard.

POO - Les objets *(exo0.py 1)*

Un objet est un… truc. Un machin. Un bidule.
En Python, absolument tout est un objet.

On décrit un objet en créant une classe : La classe est le moyen de décrire à quoi va ressembler un objet.

```
class DescriptionDeLObject:
    pass
```

POO - Les objets *(exo0.py 1)*

Puis pour utiliser cet objet, on “l’instancie” :

```
mon_objet = DescriptionDeLObject()
```

La variable ***mon_objet*** contient un objet issu de la classe ***DescriptionDeLObject***, on dit qu’il contient une instance de ***DescriptionDeLObject***.

C’est pour cela que l’action de créer un objet à partir d’une classe est appelée instanciation.

POO - Les Méthodes *(exo0.py - 2)*

Les méthodes sont des fonctions déclarées à l'intérieur de la classe.
-> Méthode est juste un nom pour dire "cette fonction est dans une classe".
Les méthodes servent à retourner les valeurs à l'intérieur de l'objet, ou à les modifier.

```
class DescriptionDeLObject:
    def la_methode(objet_en_cours):
        print ('Je suis une methode')


mon_objet = DescriptionDeLObject()
mon_objet.la_methode()
```

Mais c'est quoi "***objet_en_cours***" dans <<***def la_methode(objet_en_cours):***>>?
C'est une spécificité de Python : quand vous appelez une méthode depuis un objet, l'objet est automatiquement passé en premier paramètre par Python. C'est automatique, et invisible.
Et c'est OBLIGATOIRE sinon, ça plante
Par convention *objet_en_cours* est nommé "***self"*** : <<***def la_methode(self):***>>

POO - Les Méthodes *(exo0.py 3)*

une méthode est juste une fonction attachée à un objet, on peut donc lui passer des paramètres.

```
class DescriptionDeLObject:
    def la_methode(self, param):
        print (f'Je suis une methode avec le paramètre {param}')


mon_objet = DescriptionDeLObject()
mon_objet.la_methode('toto')
```

POO - Les Attributs *(exo0.py 4a)*

Les attributs sont juste des **variables** attachées à un objet.

```
class DescriptionDeLObject:
   mon_attribut = “toto”

mon_objet = DescriptionDeLObject()
mon_objet.mon_attribut
```

On peut modifier directement la valeur d’un attribut.
Ce changement ne sera visible que pour l’instance en cours d’utilisation

```
mon_objet.mon_attribut = 5555
```

POO - Les Attributs *(exo0.py 4b)*

Python est un langage dynamique, on peut ajouter n'importe quel attribut à n'importe quel objet en utilisant la syntaxe < objet >.< nom_attribut > :

```
>>> print mon_objet.un_attribut                    # l'attribut n'existe pas
génère une erreur

>>> mon_objet.un_attribut = "valeur de l'attribut"
>>> print(mon_objet.un_attribut)
valeur de l'attribut                               # Ouaou !!
```

POO - Les Attributs *(exo0.py 5)*

On peut accéder aux attributs depuis l'intérieur et l'extérieur d'un objet :
C'est là qu'on voit alors l'intérêt de **self** :

```
class DescriptionDeLObject:
    def afficher(self):
        # self est l'objet en cours, donc on à accès à ses attributs !
        print (self.un_attribut)

    def modifier(self):
        # on peut modifier les attributs depuis l'intérieur
        self.un_attribut = "autre valeur"
```

```
>>> mon_objet = DescriptionDeLObject()
>>> mon_objet.afficher() # l'attribut n'existe pas encore
Error...
>>> mon_objet.un_attribut = "maintenant il existe !"
>>> mon_objet.afficher()
maintenant il existe !
>>> ze_object.modifier() # l'attribut est modifié par la méthode
>>> ze_object.un_attribut
'autre valeur'
```

POO - un peu d'explication sur ***self*** *(exo0.py 6)*

En fait ***self*** correspond à l'objet en cours d'utilisation.

Donc quand on fait **self.mon_attribut**, il faut comprendre *"attribut de l'objet en cours d'utilisation"*.

De plus, ***self*** est passé de manière automatique et transparente en premier paramètre de chaques méthodes d'une class. Ça permet d'avoir accès à toutes les méthodes et attributs d'un objet en cours depuis n'importe où dans la class.

POO - Petite pause convention de nommage

Une **classe** doit être nommée en **CamelCase** : **NomDUneClasse**

Les **méthodes** et **attributs** sont écrits en **snake_case** comme pour les variables et fonctions :
- **nom_de_methode**,
- **nom_d_attribut**

L'objet en cours dans la méthode est TOUJOURS nommé **self : ma_methode(self)**

On note un attribut privé en ajoutant un underscore devant son nom : **_attributs**
Attention : rien n'est vraiment privé en python, c'est une convention de nommage qui veut juste dire "*ne pas le modifier directement*"

On note qu'un attribut est une constante en le mettant en majuscule. **ATTRIBUT_CONSTANTE** (en vrai ce n'est pas une constante, parsque les constante n'existe pas en Python, ça sert juste à avertir le développeur)

POO - Initialisation d’un objet : __init__ *(exo0.py 7)*

L’initialisation est définit par la méthode **__init__**

Ça permet de donner un état de départ à tout nouvel objet créé.

__init__ est appelé automatiquement et de manière invisible par Python (c’est une *“méthode magique”*)

__init__ reste tout de même une méthode ordinaire, il est donc possible de lui passer des paramètres.

```
class DescriptionDeLObject:
    def __init__(self, nom=”Sorin”):
        self.prenom = "Samuel"
        self.nom = nom
```

POO - petit résumé

La classe, c'est une description, un plan.

L'objet, c'est ce qu'on crée à partir de la description.
Quand on crée un objet, on dit qu'on crée une **instance** de la classe

Une méthode, c'est une fonction déclarée dans une classe *(qui est attachée à chaque objet produit, et on lui passe en premier paramètre l'objet en cours).*

Un attribut, c'est une variable attachée à un objet.

POO - Les attributs de classe *(exo0.py 7)*

Un attribut de classe est accessible par la classe, sans instance (ou avec).

Ils servent à quoi ces attributs ?
- à créer des “pseudo” constantes.
Par convention, on l’écrit tout en majuscule : **MON_ATTRIBUT**

Attention : *Si je modifie l’attribut au niveau de la classe, la classe et toutes les instances créées après ont la nouvelle valeur. Donc ne donnez pas de valeur par défaut dans les attributs de classe. Faites ça avec __init__*

POO - Les attributs de classe

```
class DescriptionDeLObject:
      ATTIBUT_DE_CLASSE = “valeur”

>>> print DescriptionDeLObject.ATTIBUT_DE_CLASSE
“valeur”

>>> mon_objet = DescriptionDeLObject()
>>> print mon_objet.ATTIBUT_DE_CLASSE
“valeur”
```

POO - exercice 1 *(exo - personne.py)*

Créer une classe personne. Cette classe permet d'avoir :

- *Attributs : âge et nom*
- *initialiser la classe*
- *changer l'âge et le nom*
- *afficher une phrase récapitulant l'identité de la personne*

Créer un scénario permettant de :

- *créér une personne qui s'appelle Bob et qui a 25 ans*
- *créer une personne qui s'appelle Toto et qui a 10 ans*
- *afficher les infos de Bob et Toto*
- *modifier l'age de Toto*
- *ré-afficher les infos de Bob et Toto*
- *ajouter dynamiquement la couleur de cheveux de Bob et l'afficher dans le recap*

POO - exercice 2 *(2_calculatrice)*

Créer une classe calculatrice permettant d'additionner, soustraire, multiplier et diviser 2 nombres entiers

- faire un scénario comportant chacune des opération
  exple: print(ma_class.add(5, 5))
- Refactoriser cette class en mettant les 2 nombres en init
- faite un scénario avec 2 nombre défini lors de l'instanciation
- faite un second scénario qui modifie le 1er nombre après le second scénario

POO - Les properties *(exo0.py 8)*

Une propriété est juste un “déguisement” qu’on met sur une méthode pour qu’elle ressemble à un attribut. On l’utilise en faisant **@property** devant une méthode.

Une propriété est en fait un “décorateur”. *(En python, un décorateur est une fonction qui modifie le comportement d'autres fonctions.)*

On peut donc “intercepter” la récupération, la suppression et la modification d’un attribut facilement. Cela permet d’exposer une belle API à base d’attribut, mais derrière faire des traitements complexes.

C’est pour cette raison qu’on n’a jamais de getter et de setter explicite en Python : on utilise les attributs tels quel, et si le besoin se présente, on en fait des propriétés.

Explications plus complètes : https://www.freecodecamp.org/news/python-property-decorator/

POO - Les properties

```
class LesVisiteursEnAmerique:

    def __init__(self):
        self._replique = 'okay'        # c'est privé, pas touche, y a un underscore

    @property
    def replique(self):
        print(“get”)
        return self._replique

    @replique.setter                   # le décorateur a le même nom que la méthode suivi de “.setter”
    def replique(self, value):         # value est la valeur à droite du '=' quand on set
        print(f”set to {value}”)       # utilisation : ma_class.replique = “Super Okay”
        self._replique = value

    @replique.deleter                  # le décorateur a le même nom que la méthode suivi de “.deletter”
    def replique(self):                # utilisation : del ma_class.replique
        print('delete')
        self._replique = None
```

POO - L'héritage de classe *(exo0.py 9)*

L'héritage permet de créer de nouvelles classes mais avec une base existante.

```
class Voiture:
        def __init__(self, nom):
                self.nom = nom
        def roule(self):
                print(“vroooom”)
        def essence(self):
                print(“je met de l’essence”)


class VoitureSport(Voiture):
        def roule(self):
                print(“Super vroooom”)
                print(“Super vroooom”)
```

```
>>> ma_voiture=Voiture('Twingo')
>>> ma_voiture.nom
'Twingo'
>>> ma_voiture.roule()
“vroooom”
>>> ma_voiture.essence()
“je met de l’essence”


>>> super_voiture = VoitureSport('Ferrari')
>>> super_voiture.nom
'Ferrari'
>>> super_voiture.roule()
“Super vroooom”
“Super vroooom”
>>> super_voiture.essence()
“je met de l’essence”
```

POO - exercice 3 *(3_voiture)*

Créer une class Voiture avec

- un nom de voiture
- une méthode roule() qui ajoute le nombre de km parcouru et enlève de l'essence et qui fait biper la voiture si elle a moins de 10 litre de carburant
- une méthode niveau_carburant() qui permet d'afficher le carburant restant
- une methode km_parcouru() qui affiche le nombre de km parcouru

Créer un scénario avec 2 véhicules qui roule sur plusieurs km et remettent de l'essence

POO - exercice 3b *(3_voiture)*

Créer une class voiture de sport qui hérite de la class voiture.
Les spécificité de la voiture de sport :

- elle est par défaut de couleur rouge
- elle consomme 2 fois plus de carburant
- son klaxon fait "Pouette Pouette"

POO - exercice 4 *(4_heritage)*

Héritage multiple:

Créer une class Papa, Maman et une class enfant qui hérite à la fois de la class Papa et Maman

- Les classes Papa et Maman doivent avoir des attibuts *“couleur_yeux”*, *“couleur_cheveux”*, *“nom”*, *“prenom”*. Chacunes de leurs valeurs doivent être différentes.
- Créer une méthode dans la class Papa
- Instancier la class enfant puis afficher les différents attributs et méthodes.
- refaites la même chose en changeant l’ordre d’héritage dans la class enfant

POO - exercice 5 *(5_fight_club)*

Créer une classe Personnage

- Créer 2 personnages. Ceux-ci ont la capacité de se combattre l'un/l'autre. Chaque coup leur retire 15 points de vie (pv). Ils en ont 100 au départ.
- Créer un scénario

Évolution 1

- On ajoute la capacité de changer d'arme.
- Un coup avec épée retire 20 points de vie
- Un coup avec une Cure-dent retire 1 points de vie

Créer un scénario avec une attaque :

- A chaque tour afficher l'état du personnage (nom, pb) ainsi que l'arme utilisée
- Si un personnage arrive à moins de 50 point de vie, il change d'arme.

POO - exercice 5 suite

Évolution 2
On ajoute la capacité de se protéger.

- créer la classe protection avec un nom et une capacité de protection
- créer un scénario où, si un personnage à une protection, on déduit les points de dégats.
- Afficher les dégâts pour chaque perso, à chaque tour

Évolution 3 ***(après cours héritage)***
Créer une nouvelle arme qui a la faculté d'être empoisonné en plus du reste

POO - exercice 6 *(6_blog)*

Reprendre l’exercice du blog avec Flask.

- Refactoriser toute la partie d’accès (select, insert, update …) à la base de données en objet
- Créer une classe article qui hérite de cette nouvelle class pour gérer votre blog

POO - Pour aller plus loin

http://sametmax.com/le-guide-ultime-et-definitif-sur-la-programmation-orientee-objet-en-python-a-lusage-des-debutants-qui-sont-rassures-par-les-textes-detailles-qui-prennent-le-temps-de-tout-expliquer-partie-1/#comment-5151

https://openclassrooms.com/fr/courses/4302126-decouvrez-la-programmation-orientee-objet-avec-python