HEH.be
Sciences
et technologies

Avenue V. Maistriau 8a
B-7000 Mons
+32 (0)65 33 81 54
scitech-mons@heh.be

WWW.HEH.BE

Programmations avancées - Pratique

Recueil exercices

Bachelier en informatique et systèmes – Finalité Télécommunications et réseaux –Bloc 1.

Depreter Johan – johan.depreter@heh.be
Desmet Erwin – erwin.desmet@heh.be
Scopel Fabrice – fabrice.scopel@heh.be

Table des matières

1. Introduction

Exercice 1

Si nous supposons que les pythons, les vipères et les cobras sont des sous-classes de la même superclasse, comment l'appelleriez-vous?

Exercice 2

Essayez de nommer quelques sous-classes de la classe python.

Exercice 3

Pouvez-vous nommer une classe simplement « class » ou « classe » ?

Exercice 4

Définir 3 types de classes différentes ainsi que leurs composants.

TD 1

Créer une classe « Serpent » qui possède les caractéristiques suivantes :

- Longueur
- Couleur
- Vitesse
- Venimeux

Elle possèdera également les méthodes suivantes :

- 'afficherInfo()', pour afficher les différentes infos de l'objet
- 'seDeplacer()', pour permettre à l'objet de se déplacer dans une direction
- 'manger()', pour définir que le serpent a mangé.

TD 2

Créer une classe « Voiture » qui possède les caractéristiques suivantes :

- Couleur
- Vitesse
- Portes
- Marque
- Demarree ?

Elle possèdera également les méthodes suivantes :

- 'afficherInfo()', pour afficher les différentes infos de l'objet
- 'avancer()', pour permettre à l'objet de se déplacer dans une direction
- 'demarrer()', pour permettre à la voiture de se mettre en route.

TP 1 – Les monstres

Créer une classe « Monstre » qui possède les caractéristiques suivantes :

- Race (Dragon, Gobelin, Fantomes, etc)
- Type (Humanoïdes, Mort-Vivants, Aquatiques, etc)
- Nom
- Age
- Dangereux ?

Elle possèdera également les méthodes suivantes :

- 'afficherInfo()', pour afficher les différentes infos de l'objet
- 'vieillir()', pour permettre au monstre de vieillir
- 'devenirDangereux(), pour permettre au monstre de devenir dangereux

TP 2 – Fil Rouge

Le but de cet exercice est de créer une classe 'Humain'.

Cette classe aura les caractéristiques suivantes :

- Nom
- Prénom
- Année de Naissance
- Enfants

Elle pourra également réaliser les actions suivantes :

- Calculer son age (méthode 'CalculerAge')
- Se présenter (méthode 'SePresenter')
- Reconnaître ses enfants (méthode 'ReconnaissanceParentale')

Instructions :

1. Créer la classe 'Humain'
2. Ajouter un constructeur qui prend en entrée le nom, le prénom et l'année de naissance.
3. Ajouter une méthode 'calculerAge' qui retourne l'âge en fonction de l'année de naissance.
4. Ajouter une méthode 'sePresenter' qui affiche le nom, le prénom et l'âge de la personne.
5. Ajouter une méthode 'reconnaissanceParentale' qui prend en entrée un 'Humain' et qui l'ajoute a la liste 'Enfants'
6. Créer plusieurs instances de 'Humain' et tester leurs méthodes.

Exercice 5

En supposant qu'il existe une classe nommée `Serpent`, écrivez la toute première ligne de la déclaration de classe `Python`, exprimant le fait que la nouvelle classe est en fait une sous-classe de Snake.

Exercice 6

Quelque chose manque dans la déclaration suivante – quoi?

```
classe Serpents
    def __init__():
        self.sound = 'Sssssss'
```

Exercice 7

Modifiez le code pour garantir que la propriété `venimeux` est privée.

```
classe Serpents
    def __init__(self):
        self.venimeux = Vrai
```

TD 3

Dans la classe 'Serpent', réaliser l'encapsulation de ses différents attributs ainsi que la création des mutateurs et des accesseurs.

TD 4

Dans la classe 'Voiture', réaliser l'encapsulation de ses différents attributs ainsi que la création des mutateurs et des accesseurs.

TP 3 – Fil Rouge

Dans l'exercice « TP 2 – Fil Rouge », réaliser l'encapsulation des différents attributs de la classe 'Humain' en rendant les propriétés privées. En effet, pour savoir ces informations auprès d'une personne, il faut lui demander d'abord.

TP 4 – Jeu de dés

Le but de cet exercice est de créer une classe 'd6'.

Cette classe aura les caractéristiques suivantes :

- Valeur

Elle pourra également réaliser les actions suivantes :

- Déterminer une valeur aléatoire (méthode 'lancerDe')

Une fois implémentée, créez une liste de 5 d6 et additionnez leur valeur, une fois « lancés ». Tant que le résultat de l'addition des différentes valeurs n'est pas d'au moins 24, recommencer le processus.

Affichez le résultat ainsi que le nombre de fois qu'il a fallu reproduire le processus.

TP complémentaires

Contextualisez un problème simple, permettant de mettre en pratique la création

de classe, l'implémentation de constructeur et de l'encapsulation.

2. Propriétés de la POO

Exercice 1

Essayer de prédire la sortie du code suivant :

```
class ExampleClass:
    a = 1
    def __init__(self):
        self.b = 2


example_object = ExampleClass()

print(hasattr(example_object, 'b'))
print(hasattr(example_object, 'a'))
print(hasattr(ExampleClass, 'b'))
print(hasattr(ExampleClass, 'a'))

```

Exercice 2

Lesquelles des propriétés de classe `Python` sont des variables d'instance et lesquelles sont des variables de classe ? Lesquels d'entre eux sont privés?

```
class Python:
    population = 1
    victims = 0
    def __init__(self):
        self.length_ft = 3
        self.__venomous = False
```

Exercice 3

Vous allez inverser l'attribut `__venomous` de l'objet `version_2`, en surpassant le fait que la propriété est privée. Comment allez-vous vous y prendre?

`version_2 = Python()`

Exercice 4

Écrivez une expression qui vérifie si l'objet `version_2` contient une propriété d'instance nommée `constrictor`.

TP 1 – Les attributs d'une voiture

Nous allons créer une classe 'Voiture' qui possède deux types d'attributs : des attributs d'instance et des attributs de classe. Les attributs d'instance sont

propres à chaque instance de la classe 'Voiture', tandis que les attributs de classe sont partagés entre toutes les instances de la classe 'Voiture'.

Voici la liste des attributs, la première étape sera de déterminer quels sont les attributs de classe et les attributs d'instance :

- Marque
- Modèle
- Nombre de voitures
- Année
- Couleurs disponibles
- Couleur
- Kilométrage

Nous allons ajouter également des méthodes à notre classe 'Voiture':

- '__init__': qui initialise les attributs d'instance de la classe 'Voiture'.
- 'changer_couleur': qui permet de changer la couleur de la voiture.
- 'afficher_informations': qui affiche les informations de la voiture (marque, modèle, année, couleur, kilométrage et nombre total de voitures).
-

Votre tâche consiste à créer une instance de la classe 'Voiture' et à tester les différentes méthodes de cette classe. Il est important d'avoir évidemment géré l'encapsulation des différents attributs.

Vous utiliserez également l'attribut '__dict__' afin de vérifier que vous avez implémenté correctement les attributs d'instance.

TP 2 – Le Chateau

Nous allons créer une classe 'Château' qui possède les attributs suivants :

- Nom
- Proprietaire
- annee_construction
- nombre_pieces
- nombre_chateaux

Déterminez lesquels sont des attributs d'instance et lesquels sont des attributs de classe.

Nous allons ajouter également des méthodes à notre classe 'Château':

- '__init__': qui initialise les attributs d'instance de la classe 'Château'.
- 'afficher_informations': qui affiche les informations du château (nom, propriétaire, année de construction et nombre de pièces).
- 'ajouter_pieces': qui permet d'ajouter un certain nombre de pièces au château.
-

Votre tâche consiste à créer plusieurs instances de la classe Chateau et à tester les différentes méthodes de cette classe. Vous allez également utiliser l'attribut __dict__ pour afficher les attributs d'instance de chaque instance.

Exercice 5

La déclaration de la classe `Snake` est donnée ci-dessous. Enrichissez la classe avec une méthode nommée `increment()`, en ajoutant `1` à la propriété `__victims`.

```
class Snake:
    def __init__(self):
        self.victims = 0
```

Exercice 6

Redéfinissez le constructeur de classe `Snake` afin qu'il ait un paramètre pour initialiser le champ `victims` avec une valeur passée à l'objet pendant la construction.

Exercice 7

Pouvez-vous prédire la sortie du code suivant ?

```
class Snake:
    pass

class Python(Snake):
    pass

print(Python.__name__, 'is a', Snake.__name__)
print(Python.__bases__[0].__name__, 'can be', Python.__name__)
```

TP 3 – Le Timer

Objectifs de cet exercice :

- améliorer les compétences de l'élève dans la définition des classes à partir de zéro;
- définition et utilisation de variables d'instance ;
- définition et utilisation de méthodes.

Scénario

Nous avons besoin d'une classe capable de compter les secondes. Facile? Pas autant que vous le pensez, car nous allons avoir des attentes spécifiques, c'est bien connu que nous sommes de professeurs exigeants !

Lisez attentivement les consignes car le code que vous écrivez sera utilisé pour lancer des fusées transportant des missions internationales vers Mars. C'est une grande responsabilité. Nous comptons sur vous !

Votre classe s'appellera `Timer`. Son constructeur acceptera trois arguments représentant les **heures** (une valeur de la plage [0..23] - nous utiliserons l'heure en code militaire), les **minutes** (de la plage [0..59]) et les **secondes** (de la plage [0..59]).

Zéro est la valeur par défaut pour tous les paramètres ci-dessus. Il n'est pas nécessaire d'effectuer des contrôles de validation dans ce cas.

La classe elle-même devra fournir les caractéristiques suivantes:

- les objets de la classe doivent être « imprimables », c'est-à-dire qu'ils doivent pouvoir se convertir implicitement en chaînes de caractères de la forme suivante: « hh:mm:ss », avec des zéros non significatifs ajoutés lorsque l'une des valeurs est inférieure à 10;

- La classe doit être équipée de méthodes sans paramètres appelées next_second() et previous_second() incrémentant le temps stocké à l'intérieur des objets de +1 ou -1 seconde respectivement.

Utilisez les conseils suivants pour mieux finaliser votre projet :

- Toutes les propriétés de l'objet doivent être privées ;

- Envisagez d'écrire une fonction distincte (pas une méthode !) pour formater la chaîne de temps.

Complétez le modèle que nous avons fournissons ci-dessous. Exécutez votre code et vérifiez si la sortie est identique à :

```
23:59:59
00:00:00
23:59:59
```

Modèle :

```
1  class Timer:
2      def __init__( ??? ):
3          #
4          # Write code here
5          #
6
7      def __str__(self):
8          #
9          # Write code here
10         #
11
12     def next_second(self):
13         #
14         # Write code here
15         #
16
17     def prev_second(self):
18         #
19         # Write code here
20         #
21
22
23 timer = Timer(23, 59, 59)
24 print(timer)
25 timer.next_second()
26 print(timer)
27 timer.prev_second()
28 print(timer)
```

TP 4 – La gestion de la semaine

Objectifs de cet exercice :

- améliorer les compétences de l'élève dans la définition des classes à partir de zéro;
- définition et utilisation de variables d'instance ;
- définition et utilisation de méthodes.

Scénario

Votre tâche consiste à implémenter une classe appelée Weeker. Oui, vos yeux ne vous trompent pas – ce nom vient du fait que les objets de cette classe pourront stocker et manipuler les jours de la semaine. It's amazing !

Le constructeur de classe accepte un argument : une chaîne de caractère. La chaîne représente le nom du jour de la semaine et les seules valeurs acceptables doivent provenir de l'ensemble suivant : Lun Mar Mer Jeu Ven Sam Dim (et oui, même à la HEH il n'existe que 7 jours possible !)

L'appel du constructeur avec un argument extérieur à cet ensemble devrait déclencher l'exception WeekDayError (définissez-la vous-même).

La classe devrait fournir les caractéristiques suivantes:

- les objets de la classe doivent être « imprimables », c'est-à-dire qu'ils doivent pouvoir se convertir implicitement en chaînes de caractères de la même forme que les arguments constructeurs;
- La classe doit être équipée de méthodes possédant un paramètre appelées add_days(n) et subtract_days(n), n étant un nombre entier et mettant à jour le jour de la semaine stocké à l'intérieur de l'objet de la manière qui reflète le changement de date du nombre de jours indiqué, en avant ou en arrière.

- Tous les attributs de l'objet doivent être privés ;

Complétez le modèle que nous avons fourni avec votre code et exécutez le pour voir si votre sortie ressemble à la nôtre.

```
Mon
Tue
Sun
Sorry, I can't serve your request.
```

Modèle :

```
class WeekDayError(Exception):
    pass

class Weeker:
    # Write code here.

    def __init__(self, day):
        # Write code here

    def __str__(self):
        # Write code here.

    def add_days(self, n):
        # Write code here.

    def subtract_days(self, n):
        # Write code here.

try:
    weekday = Weeker('Mon')
    print(weekday)
    weekday.add_days(15)
    print(weekday)
    weekday.subtract_days(23)
    print(weekday)
    weekday = Weeker('Monday')
except WeekDayError:
    print("Sorry, I can't serve your request.")
```

TP 5 – Le point

Objectifs

- améliorer les compétences de l'élève dans la définition des classes à partir de zéro;
- définition et utilisation de variables d'instance ;
- définition et utilisation de méthodes.

Scénario

Visitons un endroit très spécial - un avion avec le système de coordonnées cartésiennes (vous pouvez en apprendre plus sur ce concept en allant à vos cours de maths… c'est tellement dommage de pas aller).

Chaque point situé sur le plan peut être décrit comme une paire de coordonnées habituellement appelées x et y. Nous nous attendons à ce que vous soyez capable d'écrire une classe Python qui stocke les deux coordonnées sous forme de nombres à virgule flottante. De plus, nous voulons que les objets de cette classe évaluent les distances entre n'importe lesquels des points situés sur le plan.

La tâche est plutôt facile si vous utilisez la fonction nommée hypot() (disponible via le module mathématique) qui évalue la longueur de l'hypoténuse d'un triangle rectangle (plus de détails ici: https://en.wikipedia.org/wiki/Hypotenuse) et ici: https://docs.python.org/3.7/library/math.html#trigonometric-functions.

Voici comment nous imaginons la classe :

- Elle s'appelle `Point`;
- Son constructeur accepte deux arguments (**x** et **y**), tous deux par défaut à zéro;
- toutes les propriétés doivent être privées;
- La classe contient deux méthodes sans paramètre appelées `getX`() et `gety()`, qui renvoient chacune des deux coordonnées (les coordonnées sont stockées en privé, elles ne sont donc pas accessibles directement depuis l'objet) ;
- La classe fournit une méthode appelée `distance_from_xy(x,y)`, qui calcule et renvoie la distance entre le point stocké à l'intérieur de l'objet et l'autre point donné sous forme de pair de nombres flottants;
- la classe fournit une méthode appelée `distance_from_point(point)`, qui calcule la distance (comme la méthode précédente), mais l'emplacement de l'autre point est donné comme un autre objet de classe Point;

Complétez le modèle que nous avons fourni et vérifiez si votre sortie ressemble à la nôtre.

```
1.4142135623730951
1.4142135623730951
```

Modèle :

```python
import math

class Point:
    def __init__(self, x=0.0, y=0.0):
        # Write code here

    def getx(self):
        # Write code here

    def gety(self):
        # Write code here

    def distance_from_xy(self, x, y):
        # Write code here

    def distance_from_point(self, point):
        # Write code here

point1 = Point(0, 0)
point2 = Point(1, 1)
print(point1.distance_from_point(point2))
print(point2.distance_from_xy(2, 0))
```

TP 6 – Les joueurs

Nous allons créer une classe 'Joueur' qui possède des attributs d'instance et un attribut de classe. Les attributs d'instance sont nom, age, score et niveau. L'attribut de classe est nombre_joueurs qui représente le nombre total de joueurs créés.

Nous allons ajouter également des méthodes à notre classe 'Joueur' :

- '__init__': qui initialise les attributs d'instance de la classe 'Joueur'.
- 'afficher_informations': qui affiche les informations du joueur (nom, âge, score et niveau).
- 'augmenter_score': qui permet d'augmenter le score du joueur d'un certain nombre de points.

Enfin, nous allons créer une sous-classe 'JoueurExpert' qui hérite de la classe 'Joueur'. Cette sous-classe possède un nouvel attribut d'instance bonus et une nouvelle méthode 'afficher_bonus' qui affiche le bonus du joueur expert.

Votre tâche consiste à créer une instance de chaque classe 'Joueur' et 'JoueurExpert' et à tester les différentes méthodes de ces classes. Vous allez également utiliser l'attribut '__dict__' pour afficher les attributs d'instance de chaque instance.

De plus, il vous est demandé d'utiliser '__bases__' afin de comprendre ce qu'il est sensé afficher.

3. L'héritage

Exercice 1

En partant du code suivant, répondez aux questions :

```
1  class Dog:
2      kennel = 0
3      def __init__(self, breed):
4          self.breed = breed
5          Dog.kennel += 1
6      def __str__(self):
7          return self.breed + " says: Woof!"
8
9
10 class SheepDog(Dog):
11     def __str__(self):
12         return super().__str__() + " Don't run away, Little Lamb!"
13
14
15 class GuardDog(Dog):
16     def __str__(self):
17         return super().__str__() + " Stay where you are, Mister Intruder!"
18
19
20 rocky = SheepDog("Collie")
21 luna = GuardDog("Dobermann")
```

Question 1 : Quelle est la sortie attendue du morceau de code suivant ?

```
print(rocky)
print(luna)
```

Question 2 : Quelle est la sortie attendue du morceau de code suivant ?

```
print(issubclass(SheepDog, Dog), issubclass(SheepDog, GuardDog))
print(isinstance(rocky, GuardDog), isinstance(luna, GuardDog))
```

Question 3 : Quelle est la sortie attendue du morceau de code suivant ?

```
print(luna is luna, rocky is luna)
print(rocky.kennel)
```

Question 4 : Définissez la sous-classe de `SheepDog` nommée `LowlandDog` et équipez-la d'une méthode `__str__()` remplaçant une méthode héritée du même nom. La méthode `__str__()` du nouveau chien doit renvoyer la chaîne « Woof! Je n'aime pas les montagnes! » .

TP 1 – Fil Rouge

En reprenant la classe 'Humain' créée lors d'un exercice précédent, nous allons maintenant imaginer qu'il existe plusieurs types d'humains : les sorciers et les moldus.

Nous allons donc créer 2 nouvelles classes qui hériteront de la classe 'Humain'. Commençons par les moldus. En plus des caractéristiques des humains, les moldus ont les caractéristiques suivantes :

- Travail
- Connaissance de la magie

Ils sont également capables de réaliser les actions suivantes :

- Se présenter (méthode 'SePresenter')
- Découvrir la magie (méthode 'DecouvrirMagie')

Instructions :

1. Créer la classe 'Moldu'
2. Ajouter un constructeur qui prend le nom, le prénom, l'année de naissance et le travail et qui a comme valeur par défaut qu'il ne connait pas la magie.
3. Surcharger la méthode 'SePresenter' de la classe 'Human' pour ajouter le travail dans les informations données.
4. Ajouter la méthode 'DécouvrirMagie'.
5. Créer plusieurs instances de 'Moldu' et tester leurs méthodes.

Viens ensuite les sorciers. En plus des caractéristiques des humains, les sorciers ont les caractéristiques :

- Baguette magique
- Maison (Gryffondor, Poufsouffle, Serdaigle, Serpentard)
- Sortilèges connus

Ils sont également capables de réaliser les actions suivantes :

- Lancer un sort (méthode 'LancerSort')
- Apprendre un sort (méthode 'ApprendreSort')
- Se présenter (méthode 'SePresenter')

Instructions :

1. Créer la classe 'Sorcier'
2. Ajouter un constructeur qui prend le nom, le prénom, l'année de naissance, la baguette magique et la maison.
3. Surcharger la méthode 'SePresenter' de la classe 'Human' pour ajouter la maison dans les informations données.
4. Ajouter la méthode 'LancerSort' qui prend en entrée le nom d'un sortilège et qui vérifie si le sorcier connaît le sortilège. Si oui, afficher une phrase indiquant que le sort a été lancé. Si non, afficher une erreur.
5. Ajouter la méthode 'ApprendreSort' qui prend en entrée le nom d'un sortilège et qui l'ajoute à la liste des sorts connus.
6. Créer plusieurs instances de 'Sorcier' et tester leurs méthodes.

TP 2 – Mons Dynamics

L'exercice consiste à mettre au point une classe 'Robot' afin de passer un test pour intégrer une nouvelle start-up à Mons : Mons Dynamics qui se veut être à la pointe de la robotique (pas encore aussi connue que sa cousine de Boston, mais ça va bientôt changer !)

Cette classe devra vous permettre d'implémenter des classes enfants 'Drone' et 'RobotDanseur'.

La classe 'Robot' va posséder plusieurs attributs : nom, numéro de série (calculé en fonction du nombre de robot fabriqué), batterie.

'Drone' possèdera les attributs suivants : portée de vol et s'il y a, ou non, une caméra embarquée.
'RobotDanseur' possèdera les attributs suivants : niveau de maitrise, partenaire (un autre robot danseur), les danses qu'il connait.

Tous les objets robots devront bien entendu pouvoir se déplacer en indiquant ce qu'ils sont, par où ils vont et la méthode de déplacement. Utilisez donc une méthode 'seDeplacer' qui prendra en paramètre la direction dans laquelle le robot est sensés se déplacer.

Il y a également une méthode 'seRecharger' qui permettra de rajouter 10% à la batterie commune à tous les robots.

Les 'Drones' devront pouvoir réaliser un vol stationnaire et donc avoir une méthode 'volStationnaire' qui prendra en paramètre l'altitude à laquelle il se trouve.

Les 'RobotsDanseurs' devront avoir l'occasion de réaliser une danse avec un partenaire avec lequel il partage un type de danse.

Il est important de réaliser l'encapsulation et de protéger vos données. N'hésitez pas non plus à réaliser des tests afin que les données soient logiques (Batterie 100% max, si pas de batterie pas de déplacement, etc)

N'oubliez pas également de redéfinir __str__ afin de pouvoir afficher les informations dans un print.

Créez ensuite différents types de robots afin de tester les différentes méthodes. De plus, créez plusieurs objets 'RobotDanseur' afin de pouvoir les faire danser entre-eux.

TP 3 – Combats de super-héros

Dans cet exercice, nous allons créer une hiérarchie de classes pour modéliser des super-héros et des super-vilains avec des pouvoirs, et les faire s'affronter dans un combat.

Tout d'abord, nous allons définir une classe de base pour tous les personnages, appelée ‘Personnage’.

Cette classe contient une méthode d'initialisation qui prend un nom, une puissance et un nombre de points de vie. Elle contient également une méthode "attaquer" qui permet à un personnage d'attaquer un autre personnage en lui infligeant des dégâts égaux à sa puissance. La méthode "perdre_point_de_vie" permet à un personnage de perdre des points de vie en recevant des dégâts, et la méthode "est_ko" permet de vérifier si le personnage est KO (c'est-à-dire s'il n'a plus de points de vie).

Ensuite vont venir les 2 classes enfants ‘SuperHeros’ et ‘SuperVilain’.

La classe SuperHero hérite de la classe Personnage et ajoute des attributs pour représenter le nom secret du héros et son état de mort ou de vie. Elle surcharge également la méthode "attaquer" pour infliger le double de dégâts aux super-vilains. Enfin, elle ajoute une méthode "utiliser_pouvoir_special" qui augmente temporairement la puissance du héros de 10 par défaut.

La classe SuperVilain hérite également de la classe Personnage et ajoute des attributs pour représenter le nom de code du vilain, les héros qu’il a combattu ainsi que le but qu’il a en tant que super-vilain.

Faites combattre différents super-héros (Superman, Batman et super-vilains pour découvrir qui est le plus fort.

Bonus : Imaginez comment mettre en place la création d’équipe de super-héros et de super-vilains afin de pouvoir mettre en place des combats d’arènes à plusieurs. Vous pouvez également essayer d’implémenter de vrais pouvoirs qui aurait des effets particuliers sur l’adversaire.

TP 4 – Triangle

Objectifs

- améliorer les compétences de l'élève dans la définition des classes à partir de zéro;
- Utiliser le principe de composition (avoir une instance de classe dans une autre).

Scénario

Nous allons maintenant intégrer la classe `Point` de l'exercice précédent dans une autre classe. De plus, nous allons mettre trois points dans une classe, ce qui nous permettra de définir un triangle. Comment pouvons-nous le faire?

La nouvelle classe s'appellera `Triangle` et voici la liste de nos caractéristiques:

- le constructeur accepte trois arguments - tous sont des objets de la classe `Point`;
- les points sont stockés à l'intérieur de l'objet sous forme de liste privée ;
- La classe fournit une méthode sans paramètre appelée `perimeter()`, qui calcule le périmètre du triangle décrit par les trois points; le périmètre est une somme de toutes les longueurs de segments (nous le mentionnons par sureté, bien que nous soyons sûrs que vous le connaissez parfaitement grâce votre UE Math.)

Complétez le modèle que nous avons fourni,et vérifiez si le périmètre évalué est le même que le nôtre : `3.414213562373095`

Ci-dessous, vous pouvez copier le code de classe `Point` que nous avons utilisé dans le laboratoire précédent :

```
class Point:
    def __init__(self, x=0.0, y=0.0):
        self.__x = x
        self.__y = y
```

```
import math

class Point:
    #
    # The code copied from the previous lab.
    #


class Triangle:
    def __init__(self, vertice1, vertice2, vertice3):
        #
        # Write code here
        #

    def perimeter(self):
        #
        # Write code here
        #

triangle = Triangle(Point(0, 0), Point(1, 0), Point(0, 1))
print(triangle.perimeter())
```

TP 5 – Equipe de Basketball

Dans cet exercice, le but va être de faire en sorte de mettre au point un programme permettant de répertorier des équipes de basket avec leur joueur.

On va donc devoir créer 2 classes 'Joueur' et 'Equipe'.

La classe 'Joueur' doit avoir les attributs suivants :
- nom : le nom du joueur (une chaîne de caractères)
- age : l'âge du joueur (un entier)
- taille : la taille du joueur en centimètres (un entier)
- poste : le poste de jeu du joueur (une chaîne de caractères)

La classe 'Equipe' doit avoir les attributs suivants :
- nom : le nom de l'équipe (une chaîne de caractères)
- joueurs : une liste d'objets de la classe Joueur qui représentent les joueurs de l'équipe.

La classe Équipe doit avoir les méthodes suivantes :

- add_joueur(joueur) : ajoute un joueur à l'équipe
- remove_joueur(joueur) : retire un joueur de l'équipe
- moyenne_age() : calcule la moyenne d'âge des joueurs de l'équipe
- joueur_plus_grand() : renvoie le joueur le plus grand de l'équipe
- joueur_plus_petit() : renvoie le joueur le plus petit de l'équipe
- joueur_au_poste(poste) : renvoie le joueur de l'équipe qui occupe le poste spécifié.

Créez ensuite un objet 'Equipe' qui sera l'équipe des Profs et un deuxième objet 'Equipe' qui représentera votre groupe. Peuplez ces équipes d'objets 'Joueur'.

TP 6 – Fil Rouge

Dans un exercice précédent, nous avons créé une classe 'Sorcier'. Cette dernière pouvait lancer des sorts.

Nous allons maintenant créer une classe 'Sortilege'. Un sortilège possède les caractéristiques suivantes :
- Formule
- Effet
- Impardonnable

Nous pourrons effectuer les opérations suivantes :
- Être lancé (méthode 'SeLancer')
- Afficher les informations du sort (méthode 'Description')

Instructions :
1. Créer la classe 'Sortilege'
2. Ajouter un constructeur qui prend la formule, l'effet et s'il s'agit d'un sortilège impardonnable ou non.
3. Ajouter la méthode 'SeLancer' qui affiche une phrase indiquant que le sort

(en donnant sa formule) a été lancé avec succès.
4. Ajouter la méthode 'Description' qui affiche la formule et l'effet du sortilège.
5. Modifier la méthode 'LancerSort' de la classe 'Sorcier' pour qu'elle prenne une instance de 'Sortilege' en entrée et qu'elle fasse appel à la méthode 'SeLancer' du sortilège.
6. Modifier la méthode 'AjouterSort' de la classe 'Sorcier' pour qu'elle prenne une instance de 'Sortilege' en entrée et qu'elle l'ajoute à la liste des sorts connus.
7. Créer des instances des différentes classes et tester leurs méthodes.

4. Intégration

TP 1 – Système de gestion de location de véhicules

Le but de cet exercice est d'implémenter un système de gestion de voitures de location fonctionnel.

Etape 1 : Définition des classes

- Classe 'Vehicle' : représente un véhicule, avec les attributs suivants : 'id', 'make', 'model', 'year', 'daily_rate'.
- Classe 'Car' : représente une voiture, hérite de la classe 'Vehicle'. Elle possède les attributs 'num_doors' et 'top_speed'.
- Classe 'Truck' : représente un camion, hérite de la classe 'Vehicle'. Elle possède les attributs 'payload_capacity' et 'num_axles'
- Classe 'RentalAgreement' : représente un contrat de location, avec les attributs suivants : 'vehicle', 'start_date', 'end_date', 'daily_rate'.
- Classe 'RentalSystem' : représente le système de location, avec les attributs suivants : 'vehicles', 'agreements'.

Etape 2 : Implémentation des classes

- La classe 'Vehicle' doit avoir les méthodes suivantes :
  - '__str__' : retourne une chaîne de caractères représentant le véhicule sous la forme "id : make model year (daily_rate $/jour)"
  - 'check_odometer' : retourne le kilométrage du véhicule.
- La classe 'Car' doit avoir une méthode '__str__' qui retourne une chaîne de caractères représentant la voiture sous la forme "Voiture : make model year (daily_rate $/jour)".
- La classe 'Truck' doit avoir une méthode __str__ qui retourne une chaîne de caractères représentant le camion sous la forme "Camion : make model year (daily_rate $/jour)".
- La classe 'RentalAgreement' doit avoir une méthode 'total_cost' qui retourne le coût total de la location en fonction de la durée de la location.
- La classe 'RentalSystem' doit avoir les méthodes suivantes :
  - 'add_vehicle' : permet d'ajouter un véhicule au système.
  - 'remove_vehicle' : permet de supprimer un véhicule du système.
  - 'list_vehicles' : permet de lister tous les véhicules disponibles dans le système.
  - 'create_agreement' : permet de créer un contrat de location pour un

véhicule donné.
- 'list_agreements' : permet de lister tous les contrats de location du système
- 'return_vehicle' : permet de marquer un véhicule comme retourné dans le système

Etape 3 : Eprouver le système

- Créer plusieurs instances de véhicules (voiture ET camions)
- Instancier 'RentalSystem'.
- Ajouter les véhicules au système
- Créer des contrats de locations pour différents véhicules et différentes durées.
- Afficher les informations des véhicules.

TP 2 – Flotte spatiale

Partie 1 : Définition des classes
La première classe est la classe Vaisseau, elle doit contenir les attributs suivants:
- nom : le nom du vaisseau (une chaîne de caractères)
- modele : le modèle du vaisseau (une chaîne de caractères)
- longueur : la longueur du vaisseau en mètres (un nombre)
- largeur : la largeur du vaisseau en mètres (un nombre)
- hauteur : la hauteur du vaisseau en mètres (un nombre)
- vitesse : la vitesse maximale du vaisseau en km/h (un nombre)
- integrite : le pourcentage d'integrite du vaisseau (nombre entre 0 et 100 – ce nombre est toujours à 100 à la instanciation de l'objet).

La classe Vaisseau doit également contenir les méthodes suivantes :
- __str__ : affiche les détails du vaisseau (nom, modèle, longueur, largeur, hauteur, vitesse)
- accelerer(vitesse_supplementaire) : augmente la vitesse du vaisseau de la valeur "vitesse_supplementaire".

La deuxième classe est la classe Chasseur, elle doit hériter de la classe Vaisseau et doit contenir les attributs suivants :
- armes : une liste des armes du chasseur
- furtif : booleen permettant de savoir s'il est passé en furtif.

La classe Chasseur doit également contenir les méthodes suivantes :
- afficher_armes() : affiche les armes du chasseur
- activer_furtivite() : active la furtivite, ce qui le rend insensible au dégats.
- attaquer(vaisseau, arme) : réduit l'intégrité d'un vaisseau en fonction de l'arme utilisée

La troisième classe est la classe VaisseauDeCombat, elle doit hériter de la classe Vaisseau et doit contenir les attributs suivants :
- armure : le niveau d'armure du vaisseau de combat (un nombre)

La classe VaisseauDeCombat doit également contenir les méthodes suivantes :
- afficher_armure() : affiche le niveau d'armure du vaisseau de combat

- activer_bouclier() : active la capacité de bouclier du vaisseau de combat, ce qui augmente son armure de 100.

La quatrième classe est la classe FlotteSpatiale, elle doit contenir les attributs suivants :
- nom : le nom de la flotte (une chaîne de caractères)
- vaisseaux : une liste des vaisseaux de la flotte (une liste d'objets Vaisseau)

La classe FlotteSpatiale doit également contenir les méthodes suivantes :
- ajouter_vaisseau(vaisseau) : ajoute un vaisseau à la flotte
- supprimer_vaisseau(vaisseau) : supprime un vaisseau de la flotte
- afficher_vaisseaux() : affiche la liste des vaisseaux dans la flotte
- attaquer_flotte(flotte_cible) : permet à la flotte de lancer une attaque de tous ses vaisseaux vers les vaisseaux de la flotte cible (cibles aléatoires)

Partie 2 : Test du système
Créez un objet de type Vaisseau avec les caractéristiques suivantes :

nom : "Faucon Millenium"
modèle : "YT-1300"
longueur : 34.75
largeur : 25.61
hauteur : 8.25
vitesse : 1050

Affichez les détails du vaisseau, puis accélérez-le de 100 km/h en utilisant la méthode accelerer() et affichez à nouveau ses détails.

Créez un objet de type Chasseur avec les caractéristiques suivantes :

nom : "X-wing"
modèle : "T-65B"
longueur : 12.5
largeur : 10.5
hauteur : 2.5
vitesse : 1050
armes : ["canons laser", "torpilles à proton"]

Affichez les détails du chasseur et les armes du chasseur à l'aide de la méthode afficher_armes(). Puis, activez la capacité d'évasion du chasseur en utilisant la méthode activer_furtivite().

Créez un objet de type VaisseauDeCombat avec les caractéristiques suivantes :

nom : "Destroyer Stellaire"
modèle : "Classe Impériale"
longueur : 1600
largeur : 900
hauteur : 460
vitesse : 975
armure : 250

Affichez les détails du vaisseau de combat et le niveau d'armure à l'aide de la méthode afficher_armure(). Puis, activez la capacité de bouclier du vaisseau de combat en utilisant la méthode activer_bouclier().

Créez ensuite une deux flottes, en rajoutant d’autres vaisseaux dans chacune d’entre-elles afin de mettre en place un combat spatial.

5. UML

Exercices : Modélisation des exigences

1. L'hippodrome

Lors des courses hippiques dans un hippodrome, on a bien évidemment la possibilité de suivre les courses de chevaux mais on a aussi celle de parier.
Quels vont être les différents acteurs qui vont interagir avec ces différents services ?
Pouvez-vous construire le diagramme des cas d'utilisation.

2. Le club équestre

Un club de poneys va offrir diverses prestations tel que l'hébergement des chevaux, les balades et des cours.
Tous les clients ont le droit de partir faire des balades et de devenir adhérents.
En revanche seuls ceux qui sont déjà adhérents peuvent suivre des cours et héberger des chevaux.
Quels vont être les différents acteurs qui vont interagir dans notre club avec les différents services ?
Pouvez-vous construire le diagramme de cas d'utilisation ?

3. Le manège à chevaux de bois

Les clients d'un manège peuvent faire un tour de carrousel contre une somme.
Quels vont être les différents acteurs liés à ce service ?
Pouvez-vous construire le diagramme de cas d'utilisation ?
Pouvez-vous donner la forme textuelle de ce diagramme ?

Exercices : Modélisation de la dynamique

1.

Lors d'une course de chevaux, il faut un billet pour pouvoir entrer dans l'hippodrome. Construisez-nous donc un diagramme de séquence qui aura pour but de représenter la phase d'achat.
Quels seront les objets du système que vous allez ainsi découvrir ?

2.

Essayez de construire le diagramme de séquence de la commande de produits sur le site e-commerce de la centrale d'achats.
Quels seront les objets du système que vous allez ainsi découvrir ?

Exercices : Modélisation des objets

1.

Les chevaux ont une structure hiérarchique. Si nous considérons les classes «Stallion», «Mare», « Foal », « Horse », « Filly», « Male » et « Female ».
Si en plus, nous considérons les associations Daddy et Mommy.
Essayez de nous établir la hiérarchie des classes en n'oubliant pas d'y faire apparaitre les deux associations.

Pour cela, il faudra utiliser les contraintes {incomplete}, {complete}, {disjoint} et {overlapping}

En complément, rajoutez-y la classe `Herd`. Trouvez une manière d'ajouter l'association de composition entre Troupeau et les autres classes.
Vocabulaire : Herd= troupeau ; Foal=Poulain ; Horse=Cheval ; Fill=Pouliche ; Stallion=Etalon ; Mare=Jument ; Dady=Père ; Momy=Mère

2.
Essayez de produire le diagramme de classe des différents aspects statiques qui sont liés au paragraphe ci-dessous.

Un Marchand de poneys peut vendre plusieurs sortes de produits pour ceux-ci : Matériel d'entretien, Matériel pour monter, Alimentations, Matériel pour de ferrures.
Les commandes vont contenir un ensemble de ces produits avec une quantité.
Un devis est parfois établi avant que la commande soit faite.
Si jamais le stock est à 0, le client peut décider de se faire livrer en plusieurs fois.
Chaque livraison donnera quant à elle lieu à une facture.

Exercices : Modélisation du cycle de vies des objets

1.
Quels sont les différents états que peux prendre un ticket de Tiercé ?
Pouvez-vous construire le diagramme d'états-transitions qui correspond à la classe `TicketTiercé`.

2.
Quels sont selon vous les différents états que peux avoir une course d'équidés ?
Pouvez-vous construire le diagramme d'états-transitions qui correspond à la classe `CoursePoneys`.

3.
Un carrousel de poneys en bois peu prendre différents états ? Pouvez-vous construire le diagramme d'états-transitions qui y correspond à celui-ci?

Exercices : Modélisation des activités

1.
Imaginons que nous voulons assister à un spectacle de poneys. Pouvez-vous construire le diagramme d'activités qui correspondra à l'achat du ticket pour celui-ci ?

2.
Quel serait le diagramme d'activités pour la supervision d'une caisse vendant des tickets de tiercé ? (Concentrez-vous sur la vente de tickets mais ne prenez pas en compte les gains à rendre).

6. Exercices complets

Dans les exercices de ce chapitre, il vous sera demandé à chaque fois de réaliser le diagramme de classe. Les problématiques qui vous seront présentées seront parfois moins détaillées que dans les précédents Labo.

TP 1 – Gestion d'un parc d'attraction

Suite à l'achat d'un gigantesque terrain, M. X (qui souhaite rester anonyme) souhaite construire un parc d'attraction qui a pour thème principal l'informatique.

Il a imaginé tout un tas d'attractions et souhaite mettre en place un logiciel lui permettant de gérer ces dernières ainsi que la vente de billet.

Il est venu vous trouver afin de réaliser le diagramme de classe de sa problématique ainsi que les scripts qui lui permettront d'effectuer une gestion informatique de celui-ci.

Avec l'aide d'un analyste, il a listé les classes qui lui semblait nécessaire pour la bonne réalisation du projet :

- 'Parc' : représente le parc d'attractions et contient les informations sur les billets vendus, l'argent encaissé, la liste des attractions ainsi que les méthodes pour vendre les entrées d'une attraction, ajouter une attraction, désactiver une attraction et la supprimer.
- 'Attraction' : représente une attraction générique du parc, avec ses caractéristiques (nom, file d'attente, billet d'entrée, nombre de places, en activité) et les méthodes pour réduire la file d'attente et pour afficher le statut de l'attraction
- 'Manege', 'Montagnes Russes' et 'Carrousel' : représentent des attractions spécifiques du parc avec leurs caractéristiques propres :
    - 'Manege' : nombre de cabines, …
    - 'MontagneRusse' : vitesse maximale, …
    - 'Carrousel' : hauteur maximale, …

  Ainsi que les méthodes spécifiques pour démarrer, arrêter et vérifier le nombre de places encore disponibles
- 'Billet' : représente un billet générique pour une attraction avec ses informations (code, date d'achat, tarif, …) et les méthodes pour vérifier les détails du billet.

Une fois que vous avez réalisé le diagramme de classe qui vous semble le plus adéquat, réalisez le code en Python.

TP 2 – HEHRoom

Vous travaillez pour hôtel appelé "HEHRoom". HEHRoom a besoin d'un système pour gérer la réservation des clients et l'enregistrement/le départ de ces derniers. Les clients peuvent réserver des chambres pour une période spécifique et enregistrer leurs heures d'arrivée et de départ à l'hôtel. Le personnel de HEHRoom peut ajouter de nouvelles chambres à son catalogue, afficher les réservations et les réservations existantes et mettre à jour le statut de chaque réservation. Réalisez le diagramme de cas d'utilisation de cette situation. Par la suite, réalisez le diagramme de classe qui correspond le mieux au besoin. Une fois cela réalisé, implémentez le code en Python.

TP 3 – HEH Lan

Vous faites partie de l'organisation de la LAN party se déroulant à l'école. Il vous a été demandé de réaliser un système permettant de gérer l'inscription des joueurs et leur participation au(x) tournoi(s), ainsi que le suivi des scores et leurs classements. Les joueurs peuvent s'inscrire seul ou en équipe et choisir les tournois auxquels ils décident de s'inscrire. Les organisateurs, quant à eux, peuvent créer et configurer des tournois pour différents jeux vidéo. Réalisez le diagramme de cas d'utilisation de cette situation. Par la suite, réalisez le diagramme de classe qui correspond le mieux au besoin. Une fois cela réalisé, implémentez le code en Python.