Chapitre 1 : Programmation orientée objet TRAVAUX PRATIQUES

I. TP1 : utiliser une classe existante

Utiliser la classe suivante :

```
#
class Eleve:
      #
      matiere1="Programmation"
      matiere2="Algorithmique"
      matiere3="Projet"
      #
      def __init__(self,pNom,pPrenom,pDate,pNote1,pNote2,pNote3):
      #
            self.nom=pNom
            self.prenom=pPrenom
            self.date=pDate
            self.note_mat1=pNote1
            self.note_mat2=pNote2
            self.note_mat3=pNote3
```

1) Ajouter de la documentation

Ajouter la documentation et visualiser la documentation de la classe Eleve. Si vous donnez cette classe à un autre programmeur, il faut qu'il puisse savoir comment l'utiliser sans avoir à en décortiquer le code...

2) Créer trois nouveaux élèves ayant les caractéristiques suivantes

Elève 1 :	Elève 2 :	Elève 3 : vous
• Nom: Dubois • Prénom : Camille • Date: 01/07/2003 • Programmation: 7 • Algorithmique : 14 • Projet : 11	• Nom: Durand • Prénom : Adrien • Date: 01/11/2003 • Programmation: 13 • Algorithmique : 8 • Projet : 17	• Nom: • Prénom : • Date: • Programmation: • Algorithmique : • Projet :

3) Ajouter une méthode moyenne

Il serait intéressant de pouvoir obtenir la moyenne de l'élève avec l'instruction eleve1.moyenne(). Pour cela, créer une fonction à l'intérieur de la classe Eleve qui renvoie la moyenne de l'élève

4) Écrire une fonction moy_matieres(liste_eleve) (hors de la classe Eleve) qui prend en paramètre une liste et qui renvoie les moyennes par matières dans un dictionnaire. La liste sera ici constituée des trois élèves.

II. TP2 : création d'un mini jeu

1) Dans un nouveau fichier MonPersonnage.py, créer la classe Personnage dont la documentation est la suivante:

```
class Personnage:
    """
    Personnage d'un jeu de type hack 'n slash

    Attributs d'instance:
        nom : chaine de caractères, nom du personnage
        pv : entier positif ou nul, points de vie du personnage
        degats : entier >0, dégat maximum du personnage
    Méthodes:
        init() constructeur de la classe Personnage
        attaque() : renvoie les dégâts faits à l'adversaire
        nombre aléatoire compris entre 1 et degats avec randint()
    """
```

2) Créer 2 objets perso1 et perso2 qui sont des instances de la classe Personnage et afficher leurs attributs.
3) Proposer une bataille entre ces deux personnages :

- perso1 attaque le premier, si perso2 a ses points de vie inférieurs ou égaux à 0, vous affichez : perso2.nom " est au tapis" , sinon vous affichez : perso2.nom," a subi ",degats, " points de dégats et est à "perso2.pv," points de vie"

- puis perso2 attaque, si perso1 a ses points de vie inférieurs ou égaux à 0, vous affichez : perso1.nom est au tapis, sinon vous affichez : perso1.nom," a subi ",degats, " points de dégats et est à ",perso1.pv," points de vie"

Le jeu se termine lorsque l'un des joueurs est au tapis.

III. TP3 : Mini projet POO avec attributs privés

Le but de ce mini projet est de réaliser de plusieurs façon une classe permettant d'afficher l'heure

1) Première méthode

Créer une **classe** nommée **Heure1** avec un constructeur ayant comme paramètres **heureInit = 0**, **minuteInit = 0** et comme attributs privés: **__heure, __minute.**

Créer une méthode **getHeure()** qui renvoie **__heure**.

Créer une méthode **getMinute()** qui renvoie **__minute**.

Créer une méthode **setHeure()** ayant comme paramètre **nouvelleHeure** et change la valeur de **__heure**.

Créer une méthode **setMinute()** ayant comme paramètre **nouvelleMinute** et change la valeur de **__minute**.

Créer une méthode **incrementer()** qui permet d'incrémenter **__minute** de 1. Si **__minute** est égal à 60 alors mettre **__minute** à 0 et ajouter 1 à **__heure** ; si **__heure** est égal à 24 alors mettre **__heure** à 0.

Créer une méthode **__str__()** permettant d'afficher au format *heure : minute.*

2) Utilisation de la première classe

Créer une instance nommée **h1** de la classe **Heure1** avec les arguments 22 et 12.

Puis afficher l'heure avec la méthode **getHeure()** appliquée sur l'objet h1.

Puis afficher les minutes avec la méthode **getMinute()** appliquée sur l'objet h1.

Puis mettre **__heure** à 23 et **__minute** à 59 et afficher h1.

Puis **incrémenter** de 1 minute l'objet h1 et afficher h1.

3) Seconde méthode

Créer une nouvelle classe nommée **Heure2** avec un constructeur ayant comme paramètres heureInit = 0, minuteInit = 0 et comme attribut uniquement **__minutes**.

Créer une méthode **getHeure()** qui renvoie la valeur des heures.

Créer une méthode **getMinute()** qui renvoie la valeur des minutes.

Créer une méthode **setHeure()** ayant comme paramètre **nouvelleHeure** et change la valeur de **__minutes**.

Créer une méthode **setMinute()** ayant comme paramètre **nouvelleMinute** et change la valeur de **__minutes**.

Créer une méthode **incrementer()** qui permet d'incrémenter **__minutes** de 1.

Créer une méthode **__str__()** permettant d'afficher au format *heure : minute.*

4) Utilisation de la seconde méthode

Créer une instance nommée **h2** de la classe **Heure2** avec les arguments 22 et 12 ; puis faire comme dans la question 2 pour l'objet h2.

5) Troisième méthode

On va maintenant créer une classe **Heure3** avec une autre interface.

Créer une nouvelle classe nommée **Heure3** avec un constructeur ayant 3 paramètres hh =12, mm =0, ss =0 et 3 attributs **heure**, **minute**, **seconde**.

Créer une méthode **afficheHeure()** qui affiche l'heure au format heure :minute :seconde.

Créer une méthode **setHeure()** ayant comme paramètre **h** et change la valeur de **heure.**

6) Utilisation de la troisième méthode

Créer une instance nommée **h3** de la classe **Heure3** sans argument et appliquer la méthode afficheHeure() à cet objet.

Mettre l'heure à 13 puis afficher la nouvelle heure.

Créer une instance nommée **recreation** de la classe **Heure3** avec les arguments 9, 50, 18, puis afficher la nouvelle heure.

Remarques:

les classes Heure1 et Heure2 ont la **même interface** c'est-à-dire les **mêmes méthodes** getHeure() , getMinute(), setHeure(), setMinute() et incrementer() mais elles sont implémentées de manière différente.

IV. TP4 : Sujet 24-NSI-39, Exercice 2

CODE DONNE EN PIECE JOINTE On définit une classe gérant une adresse IPv4.

On rappelle qu'une adresse IPv4 est une adresse de longueur 4 octets, notée en décimale à point, en séparant chacun des octets par un point. On considère un réseau privé avec une plage d'adresses IP de 192.168.0.0 à 192.168.0.255.

On considère que les adresses IP saisies sont valides. Les adresses IP 192.168.0.0 et 192.168.0.255 sont des adresses réservées. Le code ci-dessous implémente la classe AdresseIP.

```
class AdresseIP:
    def __init__(self, adresse):
        self.adresse =...

    def liste_octets(self):
        """renvoie une liste de nombres entiers,
        la liste des octets de l'adresse IP"""
        # Note : split découpe la chaine de caractères
        # en fonction du séparateur
        return [int(i) for i in self.adresse.split(".")]

    def est_reservee(self):
        """renvoie True si l'adresse IP est une adresse
        réservée, False sinon"""
        reservees = [ ... ]
        return ...

    def adresse_suivante(self):
        """renvoie un objet de AdresseIP avec l'adresse
        IP qui suit l'adresse self si elle existe et None sinon"""
        octets = ...
        if ... == 254:
            return None
        octet_nouveau = ... + ...
        return AdresseIP('192.168.0.' + ...)
```

Compléter le code ci-dessus et instancier trois objets : adresse1, adresse2, adresse3 avec respectivement les arguments suivants :
'192.168.0.1', '192.168.0.2', '192.168.0.0'

Vérifier que :

```
>>> adresse1.liste_octets()
[192, 168, 0, 1]
>>> adresse1.est_reservee()
False
>>> adresse3.est_reservee()
True
>>> adresse2.adresse_suivante().adresse #acces valide à adresse ici car on sait que l'adresse suivante existe
'192.168.0.3'
```