Epreuve d'Informatique - AF2 – 10 mai 2025 - 2h

Ecrire directement sur les feuilles de l'énoncé.

Rendre toutes les feuilles (même vierges) avec votre nom dessus.

Documents autorisés : notes de cours et T.D., polys de cours.

Vous veillerez à soigner la présentation des programmes et à choisir des noms de variables « significatifs ». Le nombre de lignes précisé permet d'avoir une idée de la complexité du programme demandé, mais il n'est qu'approximatif et peut varier d'une solution à l'autre. Idem pour le temps indiqué qui doit vous aider à gérer le temps passé sur chaque exercice.

AF2 – *Compétences en programmation- savoir traduire un algorithme "simple" en un programme*

Exercice I - Compétition (environ 15 minutes)

Un groupe de joueurs se retrouvent toutes les semaines pour faire une partie d'un jeu de société et décident de stocker leurs résultats. À chaque partie, une fiche de résultat, appelée *"partie_du_jour.txt"* est complétée par chaque joueur indiquant la date de la partie puis le score et l'identité de chaque participant :

```
12/05/2025
12 : Colonel Moutarde
18 : Madame Pervenche
11 : Madame Leblanc
17 : Professeur Violet
21 : Mademoiselle Rose
15 : Docteur Olive
20 : Sergent Legris
```

Les joueurs suivent leurs résultats semaine après semaine dans un fichier *"suivi.txt"* indiquant la date de la partie, le joueur gagnant et son score :

```
31/03/2025 : Madame Pervenche : 19
07/04/2025 : Docteur Olive : 18
14/04/2025 : Madame Pervenche : 23
21/04/2025 : Mademoiselle Rose : 17
28/04/2025 : Sergent Legris : 20
05/05/2025 : Docteur Olive : 22
```

1) Lecture

On vous demande d'écrire un programme (pas de fonction demandée), qui à partir du fichier *"partie_du_jour.txt"* recherche la date de la partie, son gagnant (plus haut score) et le score de ce dernier. *(environ 9 lignes)*

2) Ecriture

Continuer le programme précédent qu'il **complète** le fichier de suivi *"suivi.txt"* avec le résultat de la dernière partie stockée dans le fichier *"partie_du_jour.txt"*. *(environ 3 et 4 lignes)*

Exercice II - Identité (environ 30 minutes)

En France, chaque personne est identifiée par son numéro INSEE. Ce numéro contient treize chiffres (exemple : "265081416802533"), construit de la façon suivante :

- Code sexe ("1" = homme, "2" = femme)
- Année naissance (2 derniers chiffres du millésime : 1980 → "80")
- Mois naissance (sur 2 chiffres : janvier="01", décembre="12")
- Code département naissance (sur 2 chiffres, exemple "75" pour Paris, "99" pour l'étranger, pour la Corse, les lettres A et B seront remplacées par des zéros)
- Code commune + ordre de naissance (sur 6 chiffres)

1) INSEE

a) Définir la classe INSEE et son constructeur *__init__* qui modélise un numéro INSEE. Les instances de cette classe ont pour attributs, tous passés en paramètres sous forme de chaînes de caractères : le code sexe, l'année de naissance (2 derniers chiffres), le mois de naissance (2 chiffres), le code du département, le code commune. *(7 lignes)*

b) Créer les méthodes accesseurs des attributs mois et année. *(4 lignes)*

c) Créer la méthode *__eq__* qui indique si l'année de naissance et le mois de naissance sont identiques. *(entre 2 et 8 lignes)*

d) Écrire une fonction *anniv_du_mois* qui reçoit en paramètres une liste d'instances de numéros INSEE et d'un numéro de mois et retourne la liste des instances dont le mois correspond. *(entre 6 lignes)*

2) Identite

a) Créer une classe *Identite* et son constructeur *_init__* qui modélise un individu avec les attributs *nom_, prenom_, jour_de_naissance_*, *numero_insee_ (qui est une instance de la classe NumINSEE).* Tous les attributs sont fournis en paramètres à la méthode *__init__* (*6 lignes*)

b) Définir la méthode *__str__* qui permet d'afficher l'identité d'une personne sous la forme suivante *(environ 4 lignes)* :

```
Prénom Nom
Date de naissance : jour mois année
```

Indications : le jour et le mois de naissance doivent être récupérés dans le numéro INSEE

c) Proposer un programme principal *main* permettant de créer une identité et de l'afficher. (3 lignes)

d) Définir une méthode *jumeau* qui retourne *True* si un individu est le jumeau d'un autre individu (né le même jour, même mois, même année). Cette méthode **utilisera la comparaison de deux numéros INSEE** citée précédemment. (*entre 4 et 5 lignes*)

3) Identité numérique

a) Définir une sous-classe *Identite_numerique* de la classe *Identite* permettant de modéliser l'identité numérique d'une personne. Les instances de cette sous-classe dispose d'un attribut supplémentaire *adresse_mail* permettant de définir une adresse mail. (*4 lignes*)

b) Repréciser pour la sous-classe *Identite_numerique,* la méthode __str__ définie précédemment pour qu'elle affiche en plus l'adresse mail. *(3 lignes)*

Exercice III - Un nouveau Robot (environ 10 minutes)

Nous fournissons ci-dessous quelques extraits de la classe *Robot* utilisée dans le projet des robots pompiers.

```
class Robot:
    def __init__(self, carte, ligne, colonne, nom, reservoir=1000, larguee, puissance):
      """ … """
        self.l_, self.c_ = ligne, colonne
        self.nom_ = nom
        self.reservoir_ = reservoir
        self.qte_ = larguee
        self.puissance_ = puissance
        self.carte_ = carte

    def agir(self, feu):
        """ … """
        lfeu, cfeu = feu.getPosition()
        l, c = self.getPosition()

        if (l == lfeu and abs(cfeu - c) == 1) or (c == cfeu and abs(lfeu - l) == 1):
            return self.larguer(feu)

        return self.deplacer(feu)

    def larguer(self,feu):
        """ … """
        # Mise à jour de l'intensité
        feu.setIntensite(min(self.reservoir_/self.qte_, 1)*self.puissance_)

        # Mise à jour du réservoir
        self.reservoir_ -= self.qte_

        # Revoie l'état du réservoir (False si vide)
        return self.reservoir_ > 0

    def est_valide(self, lcible, ccible):
        """  …    """
        return 0 <= lcible < len(self.carte_) and 0 <= ccible < len(self.carte_[0])

class Drone(Robot):
    def __init__(self, carte, ligne, colonne):
        """ … """
        Robot.__init__(self, carte, ligne, colonne, "D", 400, 100, 0.1)

class Parcelle:
    def __init__(self, nature, altitude):
        """ …        """
        self.nature_ = nature
        self.altitude_ = altitude

    def getNature(self):
        """ …         """
        return self.nature_
```

On doit modifier le comportement des drones, ils ne peuvent pas se déplacer sur une parcelle sur laquelle il y a déjà un robot.

Pour ce faire, en plus des attributs habituels, il faut fournir à chaque robot de la classe Drone la liste des robots présents sur la carte.

1) Classe Drone

Redéfinir la classe *Drone* et son constructeur *__init__* pour prendre en compte ce nouvel attribut. (*≈3 lignes*)

Vous pouvez écrire les modifications directement sur le code proposé sur la page précédente.

2) Adaptation des méthodes

Réécrire la(les) méthode(s) qui doivent être adaptées pour s'adapter à ce comportement particulier des drones.