Programmation orientée objets (Exercices)

S1 - Langages et programmation

Les exercices précédés du symbole 🖵 *sont à faire sur machine, en sauvegardant le fichier si nécessaire.*

Les exercices précédés du symbole ✎ *doivent être résolus par écrit.*

🖵 Exercice 1

On considère la classe suivante :

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

    def deplace(self, dx, dy):
        self.x = self.x + dx
        self.y = self.y + dy

    def symetrique(self):
        return Point(-self.x, -self.y)

    def __repr__(self):
        return f"Point({self.x}, {self.y})"
```

1. Quelle instruction entrer dans la console pour créer le point a d'abscisse 2 et d'ordonnée 4 ?
2. Quels sont les attributs et les méthodes de cette classe ? Dresser le diagramme de classe de cette classe.
3. La méthode spéciale `__repr__` permet de définir comment l'objet sera affiché dans la console Python.

   Qu'affichent les instructions suivantes dont la sortie a été effacée ?

   ```
   >>> b = Point(1, 2)
   >>> b
   ...
   >>> b.deplace(3, 5)
   >>> b
   ...
   ```

4. Définir une méthode `abscisse` qui renvoie l'abscisse du point.
5. Recommencer avec la méthode `ordonnee`.

Figure 1: Diagramme de classe de la classe Date

Exercice 2

Soit la classe `Date` définie par le diagramme de classe (Figure 1).

1. Implémenter cette classe en Python.
2. Créer deux dates le 20 janvier 2012 et le 14 février 2022.
3. Dans la méthode d'initialisation d'instance de la classe, prévoir un dispositif pour éviter les dates impossibles (du genre 32/14/2020). Dans ce cas, la création doit provoquer une erreur, chose possible grâce à l'instruction `raise` (documentation à rechercher !).
4. Ajouter une méthode `__repr__` et une méthode `__str__`permettant d'afficher la date sous la forme "25 janvier 1989". Les noms des mois seront définis en tant qu'attribut de classe à l'aide d'une liste.
5. Ajouter une méthode `__lt__` qui permet de comparer deux dates. L'expression `d1 < d2` (`d1` et `d2` étant deux objets de type `Date`) doit grâce à cette méthode renvoyer `True` ou `False` .

Exercice 3 (Bac 2022, extrait)

Simon souhaite créer en Python le jeu de cartes « la bataille » pour deux joueurs. Les questions qui suivent demandent de reprogrammer quelques fonctions du jeu. On rappelle ici les règles du jeu de la bataille :

Préparation :

- Distribuer toutes les cartes aux deux joueurs.
- Les joueurs ne prennent pas connaissance de leurs cartes et les laissent en tas face cachée devant eux.

Déroulement :

- À chaque tour, chaque joueur dévoile la carte du haut de son tas.
- Le joueur qui présente la carte ayant la plus haute valeur emporte les deux cartes qu'il place sous son tas.
- Les valeurs des cartes sont : dans l'ordre de la plus forte à la plus faible : As, Roi, Dame, Valet, 10, 9, 8, 7, 6, 5, 4, 3 et 2 (la plus faible).

Si deux cartes sont de même valeur, il y a "bataille".

- Chaque joueur pose alors une carte face cachée, suivie d'une carte face visible sur la carte dévoilée précédemment.
- On recommence l'opération s'il y a de nouveau une bataille sinon, le joueur ayant la valeur la plus forte emporte tout le tas.

Lorsque l'un des joueurs possède toutes les cartes du jeu, la partie s'arrête et ce dernier gagne.

Pour cela Simon crée une classe Python `Carte`. Chaque instance de la classe a deux attributs : un pour sa valeur et un pour sa couleur. Il donne au valet la valeur 11, à la dame la valeur 12, au roi la valeur 13 et à l'as la valeur 14. La couleur est une chaîne de caractères: "trefle", "carreau", "coeur" ou "pique".

Simon a écrit la classe Python `Carte` suivante, ayant deux attributs `valeur` et `couleur`, et dont le constructeur prend deux arguments: `val` et `coul`.

1. Recopier et compléter les pointillés des lignes ci-dessous.

```
class Carte:
    def __init__(self, val, coul):
        ... .valeur = ...
        ... . ... = coul
```

2. Parmi les propositions ci-dessous quelle instruction permet de créer l'objet « 7 de cœur » sous le nom c7 ?

   - `c7. init (self, 7, "coeur")`
   - `c7 = Carte(self, 7, "coeur")`
   - `c7 = Carte ( 7, "coeur")`
   - `from Carte import 7, "coeur"`

3. On souhaite créer le jeu de cartes. Pour cela, on écrit une fonction `initialiser()` :

   - sans paramètre
   - qui renvoie une liste de 52 objets de la classe `Carte` représentant les 52 cartes du jeu.

   Voici une proposition de code. Recopier et compléter les lignes suivantes pour que la fonction réponde à la demande :

```
def initialiser() :
    jeu = []
    for c in ["coeur", "carreau", "trefle", "pique"]:
        for v in range( ... ) :
            carte_cree = ...
            jeu.append(carte_cree)
    return jeu
```

4. Écrire une fonction `comparer(cartel, carte2)` qui prend en paramètres deux objets de la classe `Carte`. Cette fonction renvoie :

   - 0 si la force des deux cartes est identique,
   - 1 si la carte cartel est strictement plus forte que carte2
   - -1 si la carte carte2 est strictement plus forte que cartel

Voir le corrigé

Exercice 4 (Bac 2022)

Un fabricant de brioches décide d'informatiser sa gestion des stocks. Il écrit pour cela un programme en langage Python. Une partie de son travail consiste à développer une classe Stock dont la première version est la suivante :

```
class Stock:
    def __init__(self):
        self.qt_farine = 0 # quantité de farine initialisée à 0 g
        self.nb_oeufs = 0 # nombre d'œufs (0 à l'initialisation)
        self.qt_beurre = 0 # quantité de beurre initialisée à 0 g
```

1. Écrire une méthode `ajouter_beurre(self, qt)` qui ajoute la quantité `qt` de beurre à un objet de la classe `Stock`.

   On admet que l'on a écrit deux autres méthodes `ajouter_farine` et `ajouter_oeufs` qui ont des fonctionnements analogues.

2. Écrire une méthode `afficher(self)` qui affiche la quantité de farine, d'œufs et de beurre d'un objet de type `Stock`. L'exemple ci-dessous illustre l'exécution de cette méthode dans la console :

```
>>> mon_stock = Stock()
>>> mon_stock.afficher()
farine: 0
oeuf: 0
beurre: 0
>>> mon_stock.ajouter_beurre(560)
>>> mon_stock.afficher()
farine: 0
oeuf: 0
beurre: 560
```

3. Pour faire une brioche, il faut 350 g de farine, 175 g de beurre et 4 oeufs. Écrire une méthode `stock_suffisant_brioche(self)` qui renvoie un booléen : VRAI s'il y a assez d'ingrédients dans le stock pour faire une brioche et FAUX sinon.

4. On considère la méthode supplémentaire `produire(self)` de la classe `Stock` donnée par le code suivant :

```
def produire(self):
    res = 0
    while self.stock_suffisant_brioche():
        self.qt_beurre = self.qt_beurre - 175
        self.qt_farine = self.qt_farine - 350
        self.nb_oeufs = self.nb_oeufs - 4
        res = res + 1
    return res
```

   On considère un stock défini par les instructions suivantes :

```
>>> mon_stock=Stock()
>>> mon_stock.ajouter_beurre(1000)
>>> mon_stock.ajouter_farine(1000)
>>> mon_stock.ajouter_oeufs(10)
```

   1. On exécute ensuite l'instruction : `>>> mon_stock.produire()`. Quelle valeur s'affiche dans la console ? Que représente cette valeur ?
   2. On exécute ensuite l'instruction : `>>> mon_stock.afficher()`. Que s'affiche-t-il dans la console ?

5. L'industriel possède n lieux de production distincts et donc n stocks distincts.

   On suppose que ces stocks sont dans une liste dont chaque élément est un objet de type `Stock`. Écrire une fonction Python `nb_brioches(liste_stocks)` possédant pour unique paramètre la liste des stocks et qui renvoie le nombre total de brioches produites.

Voir le corrigé