Corrigé – Programmation orientée objet (POO)

Exercice 1

1. Les attributs de la classe `Identite` sont `nom`, `prenom` et `naissance`.
2. Les membres de la classe `Identite` sont `nom`, `prenom`, `naissance`, `__init__` et age.
3. Ligne 16 : `hugo = Identite('Bonneau', 'Jean', 1970)`.
4. 50

Exercice 2 – *(TP - Compte bancaire)*

Voici le programme complet, suivi des réponses détaillées aux questions :

```
1  class CompteBancaire:
2      """ CompteBancaire(int, str, str, int) ->
3      Classe représentant un compte bancaire
4      """
5      def __init__(self, numero_compte, nom, prenom, solde):
6          self.numero_compte = numero_compte
7          self.nom = nom
8          self.prenom = prenom
9          self.solde = solde
10
11     def __str__(self):
12         return "N° Compte : {} | Prénom : {} | Nom : {} | Solde : {}"\
13                .format(self.numero_compte, self.prenom, self.nom, self.solde)
14
15     def depot(self, somme):
16         """ depot(int) -> int
17         Ajouter le montant 'somme' au compte bancaire.
18         """
19         self.solde += somme
20         return self.solde
21
22     def retrait(self, somme):
23         """ retrait(int) -> int
24         Ajouter le montant 'somme' au compte bancaire.
25         """
26         self.solde -= somme
27         return self.solde
28
29 compte_ines = CompteBancaire(123, "Si", "Inès", 10000)
30 compte_jordi = CompteBancaire(321, "Nateur", "Jordi", 5000)
31 print(compte_ines)
32 print(compte_jordi)
33
34 # Virement de 100 euros de Inès à Jordi
35 montant_a_virer = 100
36 compte_ines.retrait(montant_a_virer)
37 compte_jordi.depot(montant_a_virer)
```

1. Lignes 1-9 du programme ci-dessus
2. Lignes 11-12 (et 17-18 pour le test)
3. Lignes 26-29
4. (a) Lignes 14-2

   (b) Lignes 31-34

Exercice 3 – *(TP - Do you want a date?)*

Voici le programme complet, suivi des réponses détaillées aux questions :

```
class Date:
    """ Date(int, int, int) -> Date
    Classe représentant une date (jour, mois, année).
    """

    def __init__(self, jour, mois, annee):
        if jour < 1 or jour > 31 or not isinstance(jour, int):
            raise ValueError("Le jour doit être un entier compris entre 1 et 31.")
        elif mois < 1 or mois > 12 or not isinstance(mois, int):
            raise ValueError("Le mois doit être un entier compris entre 1 et 12.")
        elif not isinstance(annee, int):
            raise ValueError("L'année doit être un entier.")
        else:
            self.jour = jour
            self.mois = mois
            self.annee = annee

    def __str__(self):
        """ __str__() -> str
        Permet d'afficher un objet de type 'Date' sous forme d'une chaîne de caractères de type "25 mai 1999
        """
        nom_mois = {1: "janvier", 2: "février", 3: "mars", 4: "avril", 5: "mai", 6: "juin",
                7: "juillet", 8: "aout", 9: "septembre", 10: "octobre", 11: "novembre",
                12: "decembre"}

        return "{} {} {}".format(self.jour, nom_mois[self.mois], self.annee)

    def __lt__(self, date):
        """ __lt__(Date) -> bool
        Compare la date 'self' à la date 'date' passée en paramètre. Renvoie 'True' si 'self' est avant 'dat
        """
        if self.annee < date.annee:
            return True

        if self.annee == date.annee:
            if self.mois < date.mois:
                return True

            if self.mois == date.mois:
                return self.jour < date.jour

        return False

    # OU
#     def __lt__(self, date):
#         # Exploitation de l'évaluation paresseuse des booléens en Python
#         return self.annee < date.annee or \
#             self.annee == date.annee and (self.mois < date.mois or \
#             self.mois == date.mois and self.jour < date.jour)

date1 = Date(14, 7, 1789)
date2 = Date(13, 7, 1789)

print(date1)
print(date2)

print(date1.annee)

# TESTS <
assert Date(14, 7, 1789) < Date(15, 7, 1789)
```

```
61 assert Date(14, 7, 1789) < Date(14, 8, 1789)
62 assert Date(14, 7, 1789) < Date(14, 7, 1790)
63
64 assert Date(14, 7, 1789) < Date(15, 8, 1790)
```

1. Lignes 1-15
2. (a) Lignes 17-24 (b) Lignes 48-52 (c) Ligne 54. Les attributs sont publics.
3. (a) Lignes 26-46 (b) Lignes 56-61
4. Voir *docstrings* lignes 2, 18 et 27.
5. Voici le programme modifié de façon à ce que les attributs soient privés :

```
1  class DatePrivee:
2      """ Date(int, int, int) -> Date
3      Classe représentant une date (jour, mois, année).
4      """
5
6      def __init__(self, jour, mois, annee):
7          if jour < 1 or jour > 31 or not isinstance(jour, int):
8              raise ValueError("La saisie de jour n'est pas valide")
9          elif mois < 1 or mois > 12 or not isinstance(mois, int):
10             raise ValueError("La saisie du mois n'est pas valide")
11         elif not isinstance(annee, int):
12             raise ValueError("La saisie de l'année est incorrecte")
13         else:
14             self.__jour = jour
15             self.__mois = mois
16             self.__annee = annee
17
18     def __str__(self):
19         """ __str__() -> str
20         Permet d'afficher un objet de type 'Date' sous forme d'une chaîne de caractères de type "25 mai
21         """
22         nom_mois = {1: "janvier", 2: "février", 3: "mars", 4: "avril", 5: "mai", 6: "juin",
23                 7: "juillet", 8: "aout", 9: "septembre", 10: "octobre", 11: "novembre",
24                 12: "decembre"}
25         texte = "{} {} {}".format(self.__jour, nom_mois[self.__mois], self.__annee)
26         return texte
27
28     def __lt__(self, date):
29         """ __lt__(DatePrivee) -> bool
30         Compare la date 'self' à la date 'date' passée en paramètre. Renvoie 'True' si 'self' est avant
31         """
32         return self.__annee < date.__annee or \
33             self.__annee == date.__annee and (self.__mois < date.__mois or \
34             self.__mois == date.__mois and self.__jour < date.__jour)
35
36     def get_jour(self):
37         """ get_jour() -> int
38         Renvoie le jour.
39         """
40         return self.__jour
41
42     def set_jour(self, jour):
43         """ set_jour(int) -> None
44         Modifie le jour en lui donnant la nouvelle valeur 'jour'.
45         """
46         self.__jour = jour
47
48     def get_mois(self):
49         """ get_mois() -> int
50         Renvoie le mois.
```

```
51              """
52              return self.__mois
53
54          def set_mois(self, mois):
55              """ set_mois(int) -> None
56              Modifie le mois en lui donnant la nouvelle valeur 'mois'.
57              """
58              self.__mois = mois
59
60          def get_annee(self):
61              """ get_annee() -> int
62              Renvoie l'année.
63              """
64              return self.__annee
65
66          def set_annee(self, annee):
67              """ set_annee(int) -> None
68              Modifie l'année en lui donnant la nouvelle valeur 'annee'.
69              """
70              self.__annee = annee
71
72      date1 = DatePrivee(14, 7, 1789)
73      date2 = DatePrivee(13, 7, 1789)
74
75      # Comme les attributs sont privés, la ligne suivante génère une erreur :
76      # print(date1.__jour) # AttributeError: 'DatePrivee' object has no attribute '__jour'
77
78      # Pour accéder au jour, il faut écrire :
79      print(date1.get_jour())
```

(a) Lignes 13-15, test lignes 66-70

(b) Lignes 33-61

6. L'appel à la fonction `help(Date)` affiche la documentation de la classe, basée sur les *docstrings* écrites par les développeurs. Lorsqu'on travaille en équipe, et même pour soi-même quand on reprend un ancien programme, il est important d'avoir documenté ses programmes pour en faciliter l'utilisation.

Exercice 4 – *(TP)*

Voici le programme complet, suivi des réponses détaillées aux questions :

```
1   class Bim:
2       """ Bim(str, int, float) -> Bim
3       Classe représentant un bien immobilier
4       """
5
6       def __init__(self, nature, surface, prix_moyen):
7           self.nature = nature
8           self.surface = surface
9           self.prix_moyen = prix_moyen
10
11      def estime_prix(self):
12          return self.surface * self.prix_moyen
13
14      def modifie_prix_moyen(self, prix):
15          """ modifie_prix_moyen(float) -> None
16          Modifie le prix au métre carré
17          """
18          self.prix_moyen = prix
19
20      def est_moins_cher(self, autre_bien):
21          """ est_moins_cher(Bim) -> bool
22          param: autre_bien est un objet, instance la classe Bim
```

```
23          Renvoie True si le prix estimé de self est strictement inférieur à celui de autre_bien et False sino
24          """
25          return self.estime_prix() < autre_bien.estime_prix()
26
27  # La méthode est bien en dehors de la classe ! Elle ne s'applique pas à un objet.
28  def compte_maison(lst_biens):
29      nb = 0
30      for bien in lst_biens:
31          if bien.nature == 'maison':
32              nb += 1
33      return nb
34
35  bim = Bim("appartement", 30, 8000.)
36  bien1 = Bim('maison', 70, 2000.0)
37  print(bien1.est_moins_cher(bim))
38  print(compte_maison([bim, bien1]))
```

1. Lignes 6-8 et test ligne 34
2. 140000.0 de type `float` (la méthode `estime_prix` est appliquée à l'objet `bien1`)
3. La fonction devient :

```
        def estime_prix(self):
            """ estime_prix() -> float
            Estime le prix d'un bien immobilier en fonction de sa surface et du prix moyen,
            pondéré par la nature du bien.
            """
            coeff = 1
            if self.nature == 'maison':
                coeff = 1.1
            elif self.nature == 'bureau':
                coeff = 0.8

            return self.surface * self.prix_moyen * coeff
```

4. (a) Comme `prix_moy` n'est pas un attribut existant pour l'objet `bien1`, l'instruction ligne 17 crée un nouvel attribut (nommé `prix_moy`). C'est un des dangers liés à la permissivité de Python qui autorise à créer des attributs à la volée.

   (b) Il faut remplacer `prix_moy` par `prix_moyen` à la ligne 17.

5. (a) Ligne 24

   (b) Ligne 36

6. Lignes 26-32 et test ligne 37

Exercice 5 – *(TP - Les irréductibles fractions)*

```
1   class Fraction:
2       """ Fraction(int, int) -> Fraction
3       Représente une fraction.
4       """
5
6       def __init__(self, numerateur, denominateur):
7           if denominateur <= 0:
8               raise ValueError("Le dénominateur doit être strictement positif.")
9
10          self.numerateur = numerateur
11          self.denominateur = denominateur
12
13      def __str__(self):
14          """ __str__() -> str
15          Affiche la fraction sous la forme 'numerateur/denominateur' ou 'numerateur' si le dénominateur vaut
16          """
```

```
        if self.denominateur == 1:
            return str(self.numerateur)
        else:
            return "{}/{}".format(self.numerateur, self.denominateur)

    def __eq__(self, frac):
        """ __eq__(Fraction) -> bool
        Renvoie True si les fractions 'self' et 'frac' sont égales.
        """
        return self.numerateur * frac.denominateur == self.denominateur * frac.numerateur

    def __lt__(self, frac):
        """ __lt__(Fraction) -> bool
        Renvoie True si la fraction 'self' est strictement inférieure à 'frac'.
        """
        return self.numerateur * frac.denominateur < self.denominateur * frac.numerateur

    def __add__(self, frac):
        """ __add__(Fraction) -> Fraction
        Renvoie un objet 'Fraction' résulat de la somme de la fraction 'self' et de la fraction 'frac'.
        """
        if self.denominateur == frac.denominateur:
            return Fraction(self.numerateur + frac.numerateur, self.denominateur)
        else:
            return Fraction(self.numerateur * frac.denominateur + frac.numerateur * self.denominateur, self.

    def __mul__(self, frac):
        """ __mult__(Fraction) -> Fraction
        Renvoie un objet Fraction résulat du produit de la fraction 'self' et de la fraction 'frac'.
        """
        return Fraction(self.numerateur * frac.numerateur, self.denominateur * frac.denominateur)

    # BONUS, pas obligatoire !
    def rendre_irreductible(self):
        """ rendre_irreductible(Fraction) -> None
        Fonction bonus qui modifie un objet Fraction en l'écrivant sous forme irréductible.
        """
        if self.numerateur < self.denominateur:
            min = self.numerateur
        else:
            min = self.denominateur
        for i in range(2, min + 1):
            if self.numerateur % i == 0 and self.denominateur % i == 0:
                self.numerateur = self.numerateur // i
                self.denominateur = self.denominateur // i

f1 = Fraction(1, 2)
f2 = Fraction(1, 2)
f3 = Fraction(1, 3)
f4 = Fraction(2, 4)
f5 = Fraction(5, 1)

print(f1)

# BONUS
print(f4)
f4.rendre_irreductible()
print(f4)
```