

# 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 : {}".format(
13             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 :

```
1 class Date:
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("Le jour doit être un entier compris entre 1 et 31.")
9         elif mois < 1 or mois > 12 or not isinstance(mois, int):
10            raise ValueError("Le mois doit être un entier compris entre 1 et 12.")
11        elif not isinstance(annee, int):
12            raise ValueError("L'année doit être un entier.")
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
21        ↪ mai 1999"
22        """
23        nom_mois = {1: "janvier", 2: "février", 3: "mars", 4: "avril", 5: "mai", 6: "juin",
24                    7: "juillet", 8: "août", 9: "septembre", 10: "octobre", 11: "novembre",
25                    12: "décembre"}
26
27        return "{} {} {}".format(self.jour, nom_mois[self.mois], self.annee)
28
29    def __lt__(self, date):
30        """ __lt__(Date) -> bool
31        Compare la date 'self' à la date 'date' passée en paramètre. Renvoie 'True' si 'self' est
32        ↪ avant 'date', 'False' sinon.
33        """
34        if self.annee < date.annee:
35            return True
36
37        if self.annee == date.annee:
38            if self.mois < date.mois:
39                return True
40
41            if self.mois == date.mois:
42                return self.jour < date.jour
43
44        return False
45
46    # OU
47    def __lt__(self, date):
48        # Exploitation de l'évaluation paresseuse des booléens en Python
49        return self.annee < date.annee or \
50            self.annee == date.annee and (self.mois < date.mois or \
51            self.mois == date.mois and self.jour < date.jour)
52
53    date1 = Date(14, 7, 1789)
54    date2 = Date(13, 7, 1789)
55
56    print(date1)
57    print(date2)
58
59    print(date1.annee)
```

```

59 # TESTS <
60 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
21              ↪ "25 mai 1999"
22              """
23          nom_mois = {1: "janvier", 2: "février", 3: "mars", 4: "avril", 5: "mai", 6: "juin",
24                     7: "juillet", 8: "août", 9: "septembre", 10: "octobre", 11: "novembre",
25                     12: "décembre"}
26          texte = "{} {} {}".format(self.__jour, nom_mois[self.__mois], self.__annee)
27          return texte
28
29      def __lt__(self, date):
30          """ __lt__(DatePrivee) -> bool
31              Compare la date 'self' à la date 'date' passée en paramètre. Renvoie 'True' si 'self'
32              ↪ est avant 'date', 'False' sinon.
33              """
34          return self.__annee < date.__annee or \
35                 self.__annee == date.__annee and (self.__mois < date.__mois or \
36                 self.__mois == date.__mois and self.__jour < date.__jour)
37
38      def get_jour(self):
39          """ get_jour() -> int
40              Renvoie le jour.
41              """
42          return self.__jour
43
44      def set_jour(self, jour):
45          """ set_jour(int) -> None
46              Modifie le jour en lui donnant la nouvelle valeur 'jour'.
47              """
48          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 mètre 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
24     ↪ False sinon
25     """
26     return self.estimate_prix() < autre_bien.estimate_prix()
27
28 # La méthode est bien en dehors de la classe ! Elle ne s'applique pas à un objet.
29 def compte_maison(lst_biens):
30     nb = 0
31     for bien in lst_biens:
32         if bien.nature == 'maison':
33             nb += 1
34     return nb
35
36 bim = Bim("appartement", 30, 8000.)
37 bien1 = Bim('maison', 70, 2000.0)
38 print(bien1.est_moins_cher(bim))
39 print(compte_maison([bim, bien1]))

```

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

```

def estimate_prix(self):
    """ estimate_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, numérateur, dénominateur):
7         if dénominateur <= 0:
8             raise ValueError("Le dénominateur doit être strictement positif.")
9
10        self.numérateur = numérateur
11        self.dénominateur = dénominateur
12

```

```

13 def __str__(self):
14     """ __str__() -> str
15     Affiche la fraction sous la forme 'numérateur/dénominateur' ou 'numérateur' si le
16     ↪ dénominateur vaut 1.
17     """
18     if self.dénominateur == 1:
19         return str(self.numérateur)
20     return "{}/{ {}".format(self.numérateur, self.dénominateur)
21
22 def __eq__(self, frac):
23     """ __eq__(Fraction) -> bool
24     Renvoie True si les fractions 'self' et 'frac' sont égales.
25     """
26     return self.numérateur * frac.dénominateur == self.dénominateur * frac.numérateur
27
28 def __lt__(self, frac):
29     """ __lt__(Fraction) -> bool
30     Renvoie True si la fraction 'self' est strictement inférieure à 'frac'.
31     """
32     return self.numérateur * frac.dénominateur < self.dénominateur * frac.numérateur
33
34 def __add__(self, frac):
35     """ __add__(Fraction) -> Fraction
36     Renvoie un objet 'Fraction' résultat de la somme de la fraction 'self' et de la fraction
37     ↪ 'frac'.
38     """
39     if self.dénominateur == frac.dénominateur:
40         return Fraction(self.numérateur + frac.numérateur, self.dénominateur)
41     else:
42         return Fraction(self.numérateur * frac.dénominateur + frac.numérateur *
43             ↪ self.dénominateur, self.dénominateur * frac.dénominateur)
44
45 def __mul__(self, frac):
46     """ __mult__(Fraction) -> Fraction
47     Renvoie un objet Fraction résultat du produit de la fraction 'self' et de la fraction
48     ↪ 'frac'.
49     """
50     return Fraction(self.numérateur * frac.numérateur, self.dénominateur * frac.dénominateur)
51
52 # BONUS, pas obligatoire !
53 def rendre_irréductible(self):
54     """ rendre_irréductible(Fraction) -> None
55     Fonction bonus qui modifie un objet Fraction en l'écrivant sous forme irréductible.
56     """
57     if self.numérateur < self.dénominateur:
58         min = self.numérateur
59     else:
60         min = self.dénominateur
61     for i in range(2, min + 1):
62         if self.numérateur % i == 0 and self.dénominateur % i == 0:
63             self.numérateur = self.numérateur // i
64             self.dénominateur = self.dénominateur // i
65
66 f1 = Fraction(1, 2)
67 f2 = Fraction(1, 2)
68 f3 = Fraction(1, 3)
69 f4 = Fraction(2, 4)
70 f5 = Fraction(5, 1)
71 print(f1)
72 # BONUS
73 print(f4)
74 f4.rendre_irréductible()
75 print(f4)

```