# TD9. Classes, instances et références

Bases de la Programmation Impérative – Chapitre 3 : references
version du 22 novembre 2022

L'objectif de ce TD est d'enfoncer le clou à propos des concepts de classe, d'instance et de référence introduit dans le dernier CM.

## Préambule : éléments de langage

## Question 1

Donner une définition du terme **classe**.

## Question 2

Donner une définition du terme **instance**.

## Question 3

Donner une définition du terme **référence**.

## Question 4

Donner une définition du terme **variable** ainsi qu'une ligne de code, dans n'importe quel langage, permettant de créer une variable de type entier.

## Exercice 1 : en Python, toutes les variables sont des références !

Et nous allons le prouver !

## Question 1

Nous savons maintenant que les variables Python sont toujours des **références** vers des **instances**. Comme nous l'avons fait dans le premier cours (diapositive 21 ici) et les premiers TD, exécutons pas à pas le programme suivant en modifiant sur papier un tableau contenant les variables accessibles à chaque étape du programme ainsi que les instances référencées par ces variables. Ce tableau indiquera pour chaque variable : son nom, son type, sa portée et sa valeur. La valeur sera donc dessinée par une flèche vers l'instance associée.

Autrement dit, les schémas que nous allons faire dans ce TD allient les tableaux de variables vus en début d'année avec les schémas d'instances tels que dessinés par le module traceur que nous utiliserons en TP.

```
1  #!/usr/bin/env python3
2
3  """Un petit programme pour nous, un grand programme pour l'interpréteur"""
4
5  i = 42
6  j = i
7  k = 41
8  k += 1
```

## Question 2

Que se passe-t-il si l'on rajoute une ligne `k = 17` ?

## Question 3

Dessiner l'état du programme ci-dessous, c'est à dire le tableau des variables et les instances en mémoire, après exécution de :

- la première ligne ;
- des deux premières lignes ;
- des quatre lignes.

```
#!/usr/bin/env python3

"""Un petit programme pour nous, un grand programme pour l'interpréteur"""

list1 = [3, 2, 1, "go", "feu", "partez"]
list2 = list1
list2[1] = 42
list2[2] = 17
```

# Exercice 2 : et donc les paramètres sont aussi des références !

Nous avons vu que les paramètres des fonctions sont des variables **locales** aux fonctions. Comme toutes les variables Python sont des références, les paramètres le sont aussi et c'est ce que nous allons voir dans cet exercice.

## Question 1

Dessiner l'état du programme ci-dessous, c'est à dire le tableau des variables et les instances en mémoire :

- après exécution de la ligne 12 ;
- à l'entrée de la fonction `add_1`, c'est à dire juste avant l'exécution de la ligne 7 ;
- après exécution de la ligne 7 ;
- après exécution de la ligne 13.

```
#!/usr/bin/env python3

"""Que se passe-t-il quand on passe des arguments à une fonction ?"""

def add_1(entier):
    """Une fonction qui ajoute 1, super !!"""
    entier += 1
    return entier

def main():
    """Point d'entrée du programme."""
    entier = 6
    entier = add_1(entier)
    print(f"{entier = }")

if __name__ == "__main__":
    main()
```

## Question 2

Dessiner l'état du programme ci-dessous, c'est à dire le tableau des variables et les instances en mémoire :

- après exécution de la ligne 11 ;
- à l'entrée de la fonction `add_1`, c'est à dire juste avant l'exécution de la ligne 7 ;
- après exécution de la ligne 7 mais avant de sortir de la fonction `add_1` ;
- après exécution de la ligne 12.

```
1  #!/usr/bin/env python3
2
3  """Que se passe-t-il quand on passe des arguments à une fonction ?"""
4
5  def add_1(entiers):
6      """Une fonction qui ajoute 1, super !!"""
7      entiers.append(1)
8
9  def main():
10     """Point d'entrée du programme."""
11     entiers = [3, 2]
12     add_1(entiers)
13     print(f"{entiers = }")
14
15 if __name__ == "__main__":
16     main()
```

## Exercice 3 : première classe

Nous allons maintenant voir comment définir nos propres classes. On considère le programme suivant :

```
1  #!/usr/bin/env python3
2
3  """Programme manipulant des points via deux entiers"""
4
5  def manipule_points():
6      """Quelques opérations sur des points"""
7
8      x1, y1 = 3, 5
9      x2, y2 = 4, 4
10     milieu_x, milieu_y = (x1 + x2) / 2, (y1 + y2) / 2
11     print("le milieu est (", milieu_x, milieu_y, ")")
12     print("on projette sur la droite y=5")
13     milieu_y = 5
14     print("nous sommes maintenant en (", milieu_x, milieu_y, ")")
15
16 if __name__ == "__main__":
17     manipule_points()
```

## Question 1

Pourquoi ne pas réécrire ce programme en utilisant des `namedtuple` ?

## Question 2

Écrire une classe `Point` composée de deux attributs `x` et `y` et fournissant :

- un constructeur prenant une abscisse et une ordonnée en paramètres ;
- un opérateur de conversion en chaine de caractères `__str__`.

## Question 3

Réécrire le code à l'aide de la classe `Point`

## Question 4

Dessiner l'état du programme ci-dessous après exécution de la ligne `milieu.y = 5`. Comme nous nous intéressons ici non pas à l'évolution des variables et des instances au cours du temps, mais simplement à comprendre ce qu'il se passe à un instant donné, nous ne dessinerons pas le tableau des variables mais directement les variables et instances en mémoire comme les dessinerait le module `traceur`.

# Exercice 4 : deuxième classe

## Question 1

Proposer une classe `Triangle` utilisant un tableau contenant trois références vers des instances de la classe `Point`. Penser à définir l'opérateur `__str__`.

## Question 2

Dessiner l'état du programme ci-dessous, après exécution des quatre lignes. Comme nous nous intéressons ici non pas à l'évolution des variables et des instances au cours du temps, mais simplement à comprendre ce qu'il se passe à un instant donné, nous ne dessinerons pas le tableau des variables mais directement les variables et instances en mémoire comme les dessinerait le module `traceur`.

```
p1 = Point(1,2)
p2 = Point(3,4)
p3 = Point(5,6)
t1 = Triangle(p1, p2, p3)
```

## Question 3

Dessiner l'état du programme ci-dessous juste après le dernier appel à la fonction `print` du programme ci-dessous. Comme nous nous intéressons ici non pas à l'évolution des variables et des instances au cours du temps, mais simplement à comprendre ce qu'il se passe à un instant donné, nous ne dessinerons pas le tableau des variables mais directement les variables et instances en mémoire comme les dessinerait le module `traceur`.

```
#!/usr/bin/env python3

"""Programme à analyser"""

from points_classe import Point
from triangles_classe import Triangle

def fonction_a_analyser():
```

```
    """Que fait cette fonction ?"""

    p1 = Point(0 ,0)
    p2 = Point(3, 0)
    p3 = Point(0, 3)
    p4 = Point(0 ,0)
    p5 = Point(4, 0)
    p6 = Point(0, 4)
    t1 = Triangle(p1, p2, p3)
    t2 = Triangle(p4, p5, p6)
    print(t1, t2)
    t1.points[0].x = 5
    print(t1, t2)
    t1.points[0] = p4
    print(t1, t2)
    t1.points[0].x = 5
    print(t1, t2)


if __name__ == "__main__":
    fonction_a_analyser()
```

## Exercice 5 : égaux ou identiques ? (pour aller plus loin)

## Question 1

Qu'affiche le programme ci-dessous ?

```
#!/usr/bin/env python3

"""Un petit programme mystère"""

i = 41
j = 42
k = i + 1
print(i is j)
print(i == j)
print(i is k)
print(i == k)
print(j is k)
print(j == k)
print()

a = 1234 * 5678
b = 5678 * 1234
c = b
print(a is b)
print(a == b)
print(a is c)
print(a == c)
print(b is c)
print(b == c)
print()

print(-9 is -9)
print()

r = "41"
s = "42"
t = "4" + str(2)
```

```
print(r is s)
print(r == s)
print(r is t)
print(r == t)
print(s is t)
print(s == t)
```

## Question 2

Qu'affiche le programme ci-dessous ?

```
#!/usr/bin/env python3

"""Un autre programme mystère"""

from points_classe import Point

p1 = Point(42, 42)
p2 = Point(42, 42)
print(p1 == p2)
print(p1 is p2)
```