premier_code_poo_avec_python

January 29, 2022

1 Premier code en POO avec Python : classe et objet

2 Créer une classe Point

[1]:
```
class Point:
    pass
```

Cette classe permet de créer des objets de type *Point*.

[2]:
```
p = Point()
```

[3]:
```
type(p)
```

[3]:
```
__main__.Point
```

[4]:
```
print(p)
```

```
<__main__.Point object at 0x000001F903A4B820>
```

3 Définir les attributs de la classe Point

[5]:
```
class Point:

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

La méthode `__init__` est le *constructeur* de la classe Point. Cette méthode est executée quand on instancie un objet de cette classe.

Le mot clé `self` fait référence à l'instance de la classe.

Les arguments x et y sont maintenant obligatoires pour instancier un objet.

[6]:
```
# Appel du constructeur __init__avec les arguments x=1 et y=2
a = Point(1, 2)
```

Il est possible d'accéder aux attributs de l'objet en utilisant l'opérateur `.`, par exemple, `a.x`.

[7]:
```
print(a.x, a.y)
```

```
1 2
```

[8]:
```
b = Point(3, 4)
print(b.x, b.y)
```

```
3 4
```

Il est possible de modifier directement les attributs de l'objet.

[9]:
```
b.x = 7
b.y = 8
print(b.x, b.y)
```

```
7 8
```

Remarque. Dans la plupart des langages orientés objets classiques, comme C++ ou Java, l'accès direct aux attributs (par exemple, b.x = 7) est fortement déconseillé, voire interdit, pour respecter le principe d'encapsulation. Dans ces langages, il est habituellement nécessaire de définir des méthodes spécifiques (getters et setters) pour manipuler les attributs. En Python, les attributs des objets sont par défaut publiques et peuvent être accédés directement. Il existe toutefois des conventions en Python qui permettent d'indiquer explicitement qu'un attribut est interne à l'objet et qu'il ne doit pas être accédé directement même si techniquement cela reste possible.

[10]:
```
# Le code ci-dessous produit une erreur car x et y sont manquants
# p = Point()
```

Il est possible de définir des valeurs par défaut pour les attributs.

[11]:
```
class Point:

    def __init__(self, x = 0, y = 0):
        self.x = x
        self.y = y
```

[12]:
```
# Un point par défaut avec x=0 et y=0
p = Point()
print(p.x, p.y)
```

```
0 0
```

4 Définir une représentation de l'objet

[13]:
```
print(p)
```

```
<__main__.Point object at 0x000001F903A928B0>
```

[14]:
```
print(p.x, p.y)
```

```
0 0
```

[15]:
```
point_representation = f"Point [x={p.x}, y={p.y}]"
print(point_representation)
```

```
Point [x=0, y=0]
```

La méthode spéciale `__repr__` permet d'indiquer une chaîne de caractères qui sert à représenter une classe.

[16]:
```
class Point:

    def __init__(self, x = 0, y = 0):
        self.x = x
        self.y = y

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

[17]:
```
p = Point()
print(p)
```

```
Point [x=0, y=0]
```

5 Définir une méthode

[18]:
```
class Point:

    def __init__(self, x = 0, y = 0):
        self.x = x
        self.y = y

    def reset(self):
        self.x = 0
        self.y = 0

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

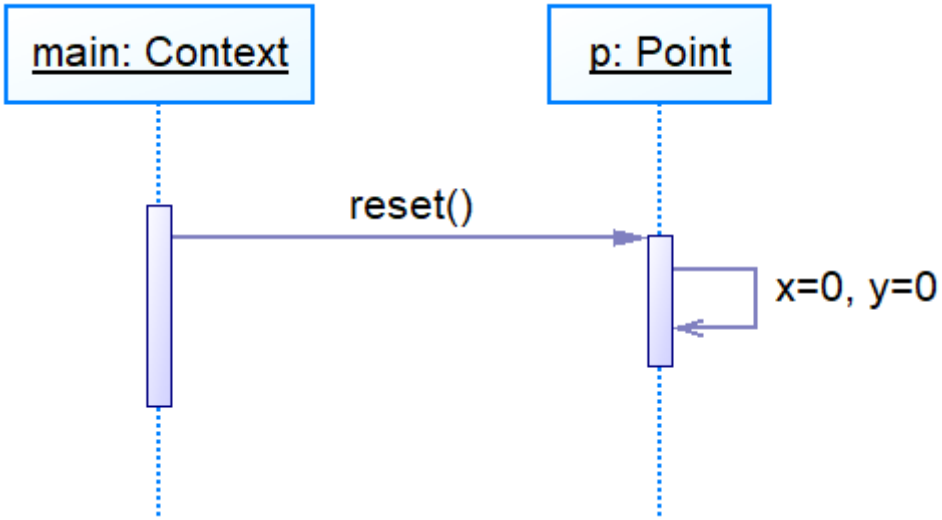
[19]:
```
p = Point(3, 4)
print(p)
```

```
Point [x=3, y=4]
```

[20]:
```
p.reset()
print(p)
```

```
Point [x=0, y=0]
```

Vocabulaire de la POO. Dans la ligne `p.reset()`, le programme principal `__main__` (parfois, on parle en général d'un Client ou d'un Contexte) envoie un *message* `reset()` à l'objet `p`.



6 Annotations des méthodes et documentation docstring

[21]:
```python
class Point:
    """Implementation of a geometric point"""

    def __init__(self, x:float = 0, y:float = 0) -> None:
        self.x = x
        self.y = y

    def reset(self) -> None:
        """Reset the point to the origin"""
        self.x = 0
        self.y = 0

    def __repr__(self) -> str:
        return f"Point [x={self.x}, y={self.y}]"
```

[22]:
```python
print(Point.__doc__)
```

```
Implementation of a geometric point
```

[23]:
```python
print(Point.reset.__doc__)
```

```
Reset the point to the origin
```

7 Exercice (corrigé)

Créer un programme qui exécute les étapes suivantes et répondre aux questions. 1. Créer le point A (-1.0, 3.0) 2. Créer le point B (3.5, 2.1) 3. Calculer la distance entre les points A et B 4. Déplacer le point A de dx=0.5 et dy=1.5 5. Est-ce que la distance entre A et B a augmenté ?

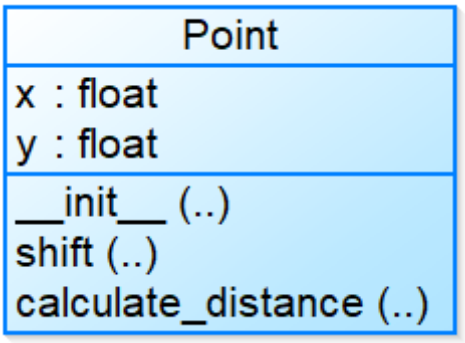
7.1 Conception

Pour résoudre cet exercice, il est nécessaire de créer la classe Point. Dans le programme, il aura deux objets de cette classe : les points A et B.

Quels attributs ?

Les points sont définis dans l'espace 2D. La classe Point possède alors deux attributs : x et y.

Quelles méthodes ? - Un constructeur qui permet d'instancier un objet avec des coordonnées x et y - Une méthode qui permet de calculer la distance entre deux points - Une méthode qui permet de déplacer un point de (dx, dy)



7.2 Implémentation de la classe Point

[24]:
```python
from math import hypot

class Point:

    def __init__(self, x:float = 0, y:float =0) -> None:
        self.x = x
        self.y = y

    def shift(self, dx: float = 0, dy: float = 0) -> None:
        self.x = self.x + dx
        self.y = self.y + dy

    def calculate_distance(self, other: "Point") -> float:
        return hypot(self.x - other.x, self.y - other.y)

    def __repr__(self) -> str:
        return f"Point [x={self.x}, y={self.y}]"
```

7.3 Implémentation du programme

[25]:
```
# Créer le point A (-1.0, 3.0)
a = Point(-1.0, 3.0)
print("A:", a)

# Créer le point B (3.5, 2.1)
b = Point(3.5, 2.1)
print("B:", b)

# Calculer la distance entre les points A et B
original_distance = a.calculate_distance(b)
print("Original distance:", original_distance)

# Déplacer le point A de dx=0.5 et dy=1.5
a.shift(dx=0.5, dy=1.5)
print("Shifted A:", a)

# Est-ce que la distance entre A et B a augmenté ?
updated_distance = a.calculate_distance(b)
print("Updated distance:", updated_distance)
print("Distance increased?", updated_distance>original_distance)
```

```
A: Point [x=-1.0, y=3.0]
B: Point [x=3.5, y=2.1]
Original distance: 4.589117562233507
Shifted A: Point [x=-0.5, y=4.5]
Updated distance: 4.66476151587624
Distance increased? True
```

[ ]: