Principe du polymorphisme

En programmation orientée objet, le polymorphisme est la capacité d'une même instruction ou expression à avoir des effets différents en fonction du type d'objet sur lequel elle est appliquée.

Il existe deux types de polymorphisme en Python :

* Le polymorphisme d'héritage, qui permet à une classe fille d'hériter des méthodes et des attributs de sa classe mère.
* Le polymorphisme ad hoc, qui permet à une même méthode d'avoir des comportements différents en fonction du type d'objet sur lequel elle est appelée.

Redéfinition des méthodes

La redéfinition des méthodes est une forme de polymorphisme d'héritage. Elle permet à une classe fille de redéfinir le comportement d'une méthode héritée de sa classe mère.

Par exemple, la classe Animal définit une méthode crier() qui affiche le cri d'un animal. La classe Chat est une sous-classe de Animal et redéfinit la méthode crier() pour afficher le miaulement d'un chat.

Python

class Animal:

def crier(self):

print("Je suis un animal !")

class Chat(Animal):

def crier(self):

print("Miaou !")

chat = Chat()

chat.crier()

Surcharge des méthodes

La surcharge des méthodes est une forme de polymorphisme ad hoc. Elle permet à une classe de définir plusieurs méthodes portant le même nom, mais avec des signatures différentes.

Par exemple, la classe Forme définit deux méthodes aire(), une pour calculer l'aire d'un cercle et une autre pour calculer l'aire d'un carré.

Python

class Forme:

def aire(self, rayon):

return 3.14 \* rayon \*\* 2

def aire(self, longueur, largeur):

return longueur \* largeur

cercle = Forme()

cercle.aire(10)

# 314.1592653589793

carre = Forme()

carre.aire(10, 10)

# 100

Surcharge des opérateurs

La surcharge des opérateurs est une forme de polymorphisme ad hoc. Elle permet à une classe de définir des comportements spécifiques pour les opérateurs mathématiques et autres.

Par exemple, la classe Point définit un comportement spécifique pour l'opérateur +.

Python

class Point:

def \_\_init\_\_(self, x, y):

self.x = x

self.y = y

def \_\_add\_\_(self, autre\_point):

return Point(self.x + autre\_point.x, self.y + autre\_point.y)

point1 = Point(1, 2)

point2 = Point(3, 4)

point3 = point1 + point2

print(point3.x, point3.y)

# 4 6