Classe python

March 15, 2023

1 Programmation orientée objet

La programmation orientée objet est l'une des approches les plus efficaces pour écrire des logiciels. Dans la programmation orientée objet, vous écrivez des classes qui représentent des situations du monde réel, et vous créez des objets basés sur ces classes.

Lorsque vous écrivez une classe, vous définissez le comportement général que toute une catégorie d'objets peut avoir. Lorsque vous créez des objets individuels à partir de la classe, chaque objet est automatiquement équipé du comportement général; vous pouvez alors donner à chaque objet les traits uniques que vous désirez. Vous serez étonné de voir à quel point des situations du monde réel peuvent être modélisées avec une programmation orientée objet.

La création d'un objet à partir d'une classe s'appelle l'instanciation et vous travaillez avec des instances d'une classe. Dans ce cours, vous allez écrire des classes et créer des instances de ces classes. Vous spécifierez le type d'informations qui peuvent être stockées dans les instances, et vous définirez les actions pouvant être entreprises avec ces instances. Vous allez également écrire des classes qui étendent les fonctionnalités des classes existantes, afin que des classes similaires puissent partager du code efficacement.

1.1 Créer et utiliser une classe

Vous pouvez modéliser presque tout en utilisant des classes. Commençons par écrire un simple classe, Chien, qui représente un chien - pas un chien en particulier, mais n'importe quel chien.

Que savons-nous de la plupart des chiens de compagnie ? Ils ont tous un nom et un âge. Nous savons également que la plupart des chiens s'assoient et se retournent.

Ces deux informations (nom et âge) et ces deux comportements (s'asseoir et se retourner) seront inclus dans notre classe Chien, car ils sont communs à la plupart des chiens. Cette classe permet à Python de savoir comment créer un objet représentant un chien. Une fois notre classe écrite, nous l'utiliserons pour créer des instances individuelles, chacune représentant un chien. Chaque instance créée à partir de la classe Dog stockera deux attribut : name et age, et nous donnerons à chaque instance deux méthodes : sit () et de roll_over ():

[2]:
```
class Dog:
    """A simple attempt to model a dog."""
    def __init__(self, name, age):
        """Initialize name and age attributes."""
        self.name = name
```

```
        self.age = age
    def sit(self):
        """Simulate a dog sitting in response to a command."""
        print(f"{self.name} is now sitting.")
    def roll_over(self):
        """Simulate rolling over in response to a command."""
        print(f"{self.name} rolled over!")
```

Chaque fonction qui fait partie d'une classe est une méthode. Tout ce que vous avez appris sur les fonctions s'applique également aux méthodes; la seule différence pratique pour le moment est la façon dont nous appellerons les méthodes.

La méthode __init__ () est une méthode spéciale que Python exécute automatiquement chaque fois que nous créons une nouvelle instance basée sur la classe Dog. Cette méthode comporte deux traits de soulignement au début et deux traits de soulignement à la fin, une convention qui permet d'éviter que les noms de méthode par défaut de Python n'entrent en conflit avec vos noms de méthode. Assurez-vous d'utiliser deux traits de soulignement de chaque côté de __init__ (). Si vous n'en utilisez qu'une de chaque côté, la méthode ne sera pas appelée automatiquement ce qui peut entraîner des erreurs difficiles à identifier.

Nous définissons la méthode __init__ () pour avoir trois paramètres: self, name et âge. Le paramètre self est requis dans la définition de la méthode, et il doit apparaitre en premier avant les autres paramètres. Il doit être inclus dans la définition car lorsque Python appellera cette méthode plus tard (pour créer une instance de Dog), l'appel de méthode passera automatiquement l'argument self.

Chaque appel de méthode associé à une instance passe automatiquement self, qui est une référence à l'instance elle-même; il donne à chaque instance un accès aux attributs et méthodes de la classe. Lorsque nous créons une instance de Dog, Python appellera la méthode__init__ () à partir de la classe Dog. Nous passerons à Dog () un nom et un âge comme arguments; self est transmis automatiquement, nous n'avons donc pas besoin de le transmettre. Chaque fois que nous voulons créer une instance à partir de la classe Dog, nous devons transmettre des valeurs uniquement pour les deux derniers paramètres, le nom et l'âge.

La ligne self.name = name prend la valeur associée au nom du paramètre et l'affecte au nom de la variable, qui est ensuite attaché à l'instance en cours de création. Le même processus se produit avec self.age = age.

Les variables accessibles via des instances comme celle-ci sont appelées attributs. La classe Dog a deux autres méthodes définies: sit () et roll_over ().

Étant donné que ces méthodes n'ont pas besoin d'informations supplémentaires pour s'exécuter, nous les définissons simplement comme ayant un paramètre, self. Les instances que nous créerons ultérieurement auront accès à ces méthodes. En d'autres termes, ils pourront s'asseoir et se retourner. Pour l'instant, sit () et roll_over () ne font pas grand-chose. Elle affichent simplement un message indiquant que le chien est assis ou se retourne. Mais le concept peut être étendu à des situations réalistes : si cette classe faisait partie d'un jeu vidéo réel, ces méthodes contiendraient du code pour faire asseoir et se retourner un chien animé. Si cette classe était écrite pour contrôler un robot, ces méthodes dirigeraient les mouvements qui amèneraient un chien robotique à s'asseoir et à se retourner.

[5]:
```
my_dog = Dog('Willie', 6)
print(f"My dog's name is {my_dog.name}.")
print(f"My dog is {my_dog.age} years old.")
```

```
My dog's name is Willie.
My dog is 6 years old.
```

La classe Dog que nous utilisons ici est celle que nous venons d'écrire dans la précédente Exemple. Nous demandons à Python de créer un chien dont le nom est 'Willie' et dont l'âge est 6 ans.

Lorsque Python lit cette ligne, il appelle la méthode __init__ () dans la classe Dog avec les arguments 'Willie' et 6. La méthode __init__ () crée une instance représentant ce chien particulier et définit les attributs nom et âge en utilisant les valeurs que nous avons fournies.

Python renvoie ensuite une instance représentant ce chien. Nous affectons cette instance à la variable my_dog. La convention de dénomination est utile ici: nous pouvons généralement supposer qu'un nom en majuscule comme Dog fait référence à une classe et un nom en minuscule comme my_dog fait référence à une instance créée à partir d'une classe.

1.1.1 Appeler des méthodes

[6]:
```
my_dog.sit()
my_dog.roll_over()
```

```
Willie is now sitting.
Willie rolled over!
```

1.1.2 Créer plusieurs instances

[7]:
```
my_dog = Dog('Willie', 6)
your_dog = Dog('Lucy', 3)

print(f"My dog's name is {my_dog.name}.")
print(f"My dog is {my_dog.age} years old.")
my_dog.sit()

print(f"\nYour dog's name is {your_dog.name}.")
print(f"Your dog is {your_dog.age} years old.")
your_dog.sit()
```

```
My dog's name is Willie.
My dog is 6 years old.
Willie is now sitting.

Your dog's name is Lucy.
Your dog is 3 years old.
Lucy is now sitting.
```

1.2 Exercice 1

1. Restaurant: Créer une classe appelée Restaurant. La méthode __init__ () pour le restaurant doit stocker deux attributs : un restaurant_name et un cuisine_type. Créez une méthode appelée describe_restaurant () qui affiche ces deux informations, et une méthode appelée open_restaurant () qui affiche un message indiquant que le restaurant est ouvert.

Créez une instance appelée restaurant à partir de votre classe. Affichez les deux attributs individuellement, puis appelez les deux méthodes.

2. Trois restaurants : Créer trois différentes instances de la classe Restaurant, et appelez describe_restaurant () pour chaque exemple.

3. Utilisateurs : créez une classe appelée Utilisateur. Créez deux attributs appelés first_name et last_name, puis créez plusieurs autres attributs qui sont généralement stockés dans un profil utilisateur. Créez une méthode appelée describe_user () qui affiche un résumé des informations de l'utilisateur. Créez une autre méthode appelée greet_user () qui affiche un message d'accueil personnalisé à l'utilisateur.

Créez plusieurs instances représentant différents utilisateurs et appelez les deux méthodes pour chaque utilisateur.

2 Classes et Instances

Vous pouvez utiliser des classes pour représenter de nombreuses situations du monde réel. Une fois que vous écrivez une classe, vous passerez la plupart de votre temps à travailler avec des instances créées à partir de cette classe. L'une des premières tâches que vous voudrez faire est de modifier les attributs associé à une instance particulière. Vous pouvez modifier les attributs d'un directement ou écrivez des méthodes qui mettent à jour les attributs de manière spécifique.

[11]:
```
class Car:
    """A simple attempt to represent a car."""
    def __init__(self, make, model, year):
        """Initialize attributes to describe a car."""
        self.make = make
        self.model = model
        self.year = year
    def get_descriptive_name(self):
        """Return a neatly formatted descriptive name."""
        long_name = f"{self.year} {self.make} {self.model}"
        return long_name.title()
```

[12]:
```
my_new_car = Car('audi', 'a4', 2019)
print(my_new_car.get_descriptive_name())
```

```
2019 Audi A4
```

2.1 Définir une valeur par défaut à un attribut

[19]:
```
class Car2:
    """A simple attempt to represent a car."""
    def __init__(self, make, model, year):
        """Initialize attributes to describe a car."""
        self.make = make
        self.model = model
        self.year = year
        self.odometer_reading = 0
    def get_descriptive_name(self):
        """Return a neatly formatted descriptive name."""
        long_name = f"{self.year} {self.make} {self.model}"
        return long_name.title()
    def read_odometer(self):
        """Print a statement showing the car's mileage."""
        print(f"This car has {self.odometer_reading} miles on it.")
```

[21]:
```
my_new_car = Car2('audi', 'a4', 2019)
print(my_new_car.get_descriptive_name())
my_new_car.read_odometer()
```

```
2019 Audi A4
This car has 0 miles on it.
```

2.2 Modifier la valeur d'un attribut

2.2.1 Modifier la valeur d'un attribut directement

[22]:
```
my_new_car.odometer_reading = 23
my_new_car.read_odometer()
```

```
This car has 23 miles on it.
```

2.2.2 Modifier la valeur d'un attribut en utilsant une méthode

[30]:
```
class Car3:
    """A simple attempt to represent a car."""
    def __init__(self, make, model, year):
        """Initialize attributes to describe a car."""
        self.make = make
        self.model = model
        self.year = year
        self.odometer_reading = 0
    def get_descriptive_name(self):
        """Return a neatly formatted descriptive name."""
        long_name = f"{self.year} {self.make} {self.model}"
        return long_name.title()
```

```
    def read_odometer(self):
        """Print a statement showing the car's mileage."""
        print(f"This car has {self.odometer_reading} miles on it.")
    def update_odometer(self, mileage):
        """Set the odometer reading to the given value."""
        self.odometer_reading = mileage
```

[31]:
```
my_new_car2 = Car3('audi', 'a3', 2020)
print(my_new_car2.get_descriptive_name())
```

```
2020 Audi A3
```

[33]:
```
my_new_car2.update_odometer(23)
my_new_car2.read_odometer()
```

```
This car has 23 miles on it.
```

Nous pouvons étendre la méthode update_odometer () pour faire un traitement supplémentaire. On veut s'assurer que personne n'essaye de faire reculer la lecture du compteur kilométrique:

[35]:
```
class Car4:
    """A simple attempt to represent a car."""
    def __init__(self, make, model, year):
        """Initialize attributes to describe a car."""
        self.make = make
        self.model = model
        self.year = year
        self.odometer_reading = 0
    def get_descriptive_name(self):
        """Return a neatly formatted descriptive name."""
        long_name = f"{self.year} {self.make} {self.model}"
        return long_name.title()
    def read_odometer(self):
        """Print a statement showing the car's mileage."""
        print(f"This car has {self.odometer_reading} miles on it.")
    def update_odometer(self, mileage):
        """
        Set the odometer reading to the given value.
        Reject the change if it attempts to roll the odometer back.
        """
        if mileage >= self.odometer_reading:
            self.odometer_reading = mileage
        else:
            print("You can't roll back an odometer!")
```

Parfois, on souhaite incrémenter la valeur d'un attribut d'une certaine valeur plutôt que de définir une valeur entièrement nouvelle.

[36]:
```python
class Car5:
    """A simple attempt to represent a car."""
    def __init__(self, make, model, year):
        """Initialize attributes to describe a car."""
        self.make = make
        self.model = model
        self.year = year
        self.odometer_reading = 0
    def get_descriptive_name(self):
        """Return a neatly formatted descriptive name."""
        long_name = f"{self.year} {self.make} {self.model}"
        return long_name.title()
    def read_odometer(self):
        """Print a statement showing the car's mileage."""
        print(f"This car has {self.odometer_reading} miles on it.")
    def update_odometer(self, mileage):
        """
        Set the odometer reading to the given value.
        Reject the change if it attempts to roll the odometer back.
        """
        if mileage >= self.odometer_reading:
            self.odometer_reading = mileage
        else:
            print("You can't roll back an odometer!")
    def increment_odometer(self, miles):
        """Add the given amount to the odometer reading."""
        self.odometer_reading += miles
```

[40]:
```python
my_used_car = Car5('subaru', 'outback', 2015)
print(my_used_car.get_descriptive_name())

my_used_car.update_odometer(23500)
my_used_car.read_odometer()

my_used_car.increment_odometer(100)
my_used_car.read_odometer()
```

```
2015 Subaru Outback
This car has 23500 miles on it.
This car has 23600 miles on it.
```

2.3 Exercice 2

1. Nombre de clients servis : reprenez par votre programme de l'exercice 1.1. Ajoutez un attribut appelé number_served avec une valeur par défaut de 0. - Créez une instance appelée restaurant de cette classe. Imprimer le nombre de clients que le restaurant a servi, puis modifiez cette valeur et imprimez-la à nouveau. - Ajoutez une méthode appelée set_number_served () qui vous permet de définir le nombre des clients qui ont été servis. Appelez cette méthode avec un nouveau numéro

et imprimer à nouveau la valeur. - Ajoutez une méthode appelée increment_number_served () qui vous permet d'incrémenter le nombre de clients qui ont été servis. Appelez cette méthode.

2. Tentatives de connexion : ajoutez un attribut appelé login_attempts à votre classe utilisateur de l'exercice 1.3. Ecrire une méthode appelée increment_login_attempts () qui incrémente la valeur de login_attempts de 1. Écrivez une autre méthode appelée reset_login_attempts () qui réinitialise la valeur de login_attempts à 0. Créez une instance de la classe User et appelez increment_login_attempts () plusieurs fois. Imprimez la valeur de login_attempts pour vous assurer qu'elle a été incrémentée correctement, puis appelez reset_login_attempts (). Imprimer à nouveau login_attentats pour assurez-vous qu'il a été remis à 0.

3 Héritage

[42]:
```
class ElectricCar(Car5):
    """Represent aspects of a car, specific to electric vehicles."""
    def __init__(self, make, model, year):
        """Initialize attributes of the parent class."""
        super().__init__(make, model, year)
```

[43]:
```
my_tesla = ElectricCar('tesla', 'model s', 2019)
print(my_tesla.get_descriptive_name())
```

```
2019 Tesla Model S
```

[44]:
```
class ElectricCar2(Car5):
    """Represent aspects of a car, specific to electric vehicles."""
    def __init__(self, make, model, year):
        """
        Initialize attributes of the parent class.
        Then initialize attributes specific to an electric car.
        """
        super().__init__(make, model, year)
        self.battery_size = 75
    def describe_battery(self):
        """Print a statement describing the battery size."""
        print(f"This car has a {self.battery_size}-kWh battery.")
```

[50]:
```
my_tesla2 = ElectricCar2('tesla', 'model s', 2020)
print(my_tesla2.get_descriptive_name())
my_tesla2.describe_battery()
```

```
2020 Tesla Model S
This car has a 75-kWh battery.
```

4 Utiliser une instance comme attribut

[51]:
```
class Battery:
    """A simple attempt to model a battery for an electric car."""
    def __init__(self, battery_size=75):
        """Initialize the battery's attributes."""
        self.battery_size = battery_size
    def describe_battery(self):
        """Print a statement describing the battery size."""
        print(f"This car has a {self.battery_size}-kWh battery.")
```

[52]:
```
class ElectricCar3(Car5):
    """Represent aspects of a car, specific to electric vehicles."""
    def __init__(self, make, model, year):
        """
        Initialize attributes of the parent class.
        Then initialize attributes specific to an electric car.
        """
        super().__init__(make, model, year)
        self.battery = Battery()
```

[53]:
```
my_tesla3 = ElectricCar3('tesla', 'model s', 2019)
print(my_tesla3.get_descriptive_name())
my_tesla3.battery.describe_battery()
```

```
2019 Tesla Model S
This car has a 75-kWh battery.
```

[54]:
```
class Battery2:
    """A simple attempt to model a battery for an electric car."""
    def __init__(self, battery_size=75):
        """Initialize the battery's attributes."""
        self.battery_size = battery_size
    def describe_battery(self):
        """Print a statement describing the battery size."""
        print(f"This car has a {self.battery_size}-kWh battery.")
    def get_range(self):
        """Print a statement about the range this battery provides."""
        if self.battery_size == 75:
            range = 260
        elif self.battery_size == 100:
            range = 315
        print(f"This car can go about {range} miles on a full charge.")
```

[55]:
```
class ElectricCar4(Car5):
    """Represent aspects of a car, specific to electric vehicles."""
    def __init__(self, make, model, year):
```

```
        """
        Initialize attributes of the parent class.
        Then initialize attributes specific to an electric car.
        """
        super().__init__(make, model, year)
        self.battery = Battery2()
```

[56]:
```
my_tesla4 = ElectricCar4('tesla', 'model s', 2019)
print(my_tesla4.get_descriptive_name())
my_tesla4.battery.describe_battery()
my_tesla4.battery.get_range()
```

```
2019 Tesla Model S
This car has a 75-kWh battery.
This car can go about 260 miles on a full charge.
```

5 Exercice 3

1. Stand de crème glacée : Un stand de crème glacée est un type spécifique de restaurant. Écrivez une classe appelée IceCreamStand qui hérite de la classe Restaurant que vous avez écrite dans l'exercice 2.1. Ajoutez un attribut appelé saveurs qui stocke une liste de saveurs de crème glacée. Écrivez une méthode qui affiche ces saveurs. Créez une instance de IceCreamStand et appelez cette méthode.

2. Admin : un administrateur est un type particulier d'utilisateur. Écrivez une classe appelée Admin qui hérite de la classe User que vous avez écrite dans l'exercice 2.2. - Ajouter un attribut, des privilèges, qui stocke une liste de chaînes de caractères comme "peut ajouter un message", "peut supprimer un message", "peut interdire l'utilisateur", et ainsi de suite. - Écrivez une méthode appelée show_privileges () qui répertorie l'ensemble des privilèges. - Créez une instance d'Admin et appelez votre méthode.

3. Privilèges: écrivez une classe Privileges distincte. La classe devrait en avoir un attribut, privilèges, qui stocke une liste de chaînes de caractères. Déplacez la méthode show_privileges () vers cette classe. Créer une instance Privileges comme attribut dans la classe Admin. Créez une nouvelle instance d'Admin et utilisez votre méthode pour montrer ses privilèges.