### El método \_\_init\_\_() de una clase hija

Cuando escribas una nueva clase basada en una clase existente, a menudo querrás llamar al método \_\_init\_\_() de la clase padre. Esto inicializará los atributos definidos en el método \_\_init\_\_() de la clase padre y los pondrá a disposición de la clase hija.

Por ejemplo, modelemos un coche eléctrico. Un coche eléctrico es sólo una clase específica de coche, así que podemos basar nuestra nueva clase ElectricCar en la clase Car que escribimos antes. Así sólo tendremos que escribir código para los atributos y comportamientos específicos de los coches eléctricos.

Empecemos haciendo una versión sencilla de la clase ElectricCar, que hace todo lo que hace la clase Car:

**electric\_car.py**

❶ 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  
 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."""  
 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  
  
❷ class ElectricCar(Car):  
 """Represent aspects of a car, specific to electric vehicles."""  
  
❸ def \_\_init\_\_(self, make, model, year):  
 """Initialize attributes of the parent class."""  
❹ super().\_\_init\_\_(make, model, year)  
  
  
❺ my\_leaf = ElectricCar('nissan', 'leaf', 2024)  
print(my\_leaf.get\_descriptive\_name())

Empezaremos con Car ❶. Cuando creas una clase hija, la clase padre debe formar parte del archivo actual y debe aparecer antes que la clase hija en el archivo. A continuación definimos la clase hija, ElectricCar ❷. El nombre de la clase padre debe incluirse entre paréntesis en la definición de una clase hija. El método \_\_init\_\_() recoge la información necesaria para crear una instancia de Car ❸.

La función super() ❹ es una función especial que te permite llamar a un método de la clase padre. Esta línea indica a Python que llame al método \_\_init\_\_() desde Car, lo que proporciona a una instancia de ElectricCar todos los atributos definidos en ese método. El nombre *super* proviene de la convención de llamar a la clase padre *superclass* y a la clase hija *subclass*.

Comprobamos si la herencia funciona correctamente intentando crear un coche eléctrico con el mismo tipo de información que proporcionaríamos al crear un coche normal. Creamos una instancia de la clase ElectricCar y la asignamos a my\_leaf ❺. Esta línea llama al método \_\_init\_\_() definido en ElectricCar, que a su vez le dice a Python que llame al método \_\_init\_\_() definido en la clase padre Car. Proporcionamos los argumentos 'nissan', 'leaf', y 2024.

Aparte de \_\_init\_\_(), aún no hay atributos ni métodos particulares de un coche eléctrico. En este momento sólo nos estamos asegurando de que el coche eléctrico tiene los comportamientos Car adecuados:

2024 Nissan Leaf

La instancia de ElectricCar funciona igual que una instancia de Car, así que ahora podemos empezar a definir atributos y métodos específicos de los coches eléctricos.

[anterior](c09_13.html)[Subtema 14 de 34: (Ver todo)](c09.html)[siguiente](c09_15.html)