PRO-UT2-A4 Herencia múltiple

Herencia múltiple en Python

Python permite herencia múltiple. Se utiliza cuando la clase hija tiene características de **más de una clase madre**. En este caso, la clase hija hereda los miembres de más de una clase madre:

```
class MotherClass1:
    pass

class MotherClass2:
    pass

class ChildClass(MotherClass1, MotherClass2):
    pass
```

Ejemplo:

```
class SeaAnimal:
    def __init__(self, name):
        self.name = name

    def swim(self):
        return "Swimming"

class LandAnimal:
    def __init__(self, name):
        self.name = name

    def walk(self):
        return "Walking..."

class Penguin(SeaAnimal, LandAnimal):
    def __init__(self, name):
        super().__init__(name)
```

En el caso anterior un Pingüino **es un** animal marino y **es un** animal terrestre y puede, por tanto, nadar y caminar.

```
linux = Penguin("Tux")
print(linux.walk())     # Walking...
print(linux.swim())     # Swimming
```

Orden de resolución

Al heredar de más de una clase madre se puede dar el caso de que dichas clases madres tengan métodos con el mismo nombre:

```
class B:
    def x(self):
        print('x: B')
```

```
class C:
    def x(self):
        print('x: C')


class D(B, C):
    pass


d = D()
d.x()
```

El resultado de ejecutar el programa anterior es:

```
x: B
```

Python resuelve este caso siguiendo las siguientes reglas:

- Si el método ha sido sobreescrito en la clase hija, este es el que se ejecuta.
- Si el método no existe en la clase hija pero si en más de una de las clases madres se ejecuta el de aquella que se haya especificado antes a la hora de heredar.

Lo mismo pasa si usamos para llamar usando `super()` un método que existe en más de una clase madre, se resuelve ejecutando el de aquella que se haya indicando que **hereda en primer lugar** (más a la izquierda):

```
class B:
    def x(self):
        print('x: B')


class C:
    def x(self):
        print('x: C')


class D(B, C):
    def x(self):
        super().x()
    pass


d = D()
d.x()         # 'x: B'
```

Usando métodos de más de una clase madre

Cómo vimos, el método `super()` nos perminte acceder a métodos de la clase madre, pero si tenemos más de un antecesor con **métodos con el mismo nombre**, debemos disponer de una forma de discriminar de cuál de ellos queremos reutilizar un miembro. Para ello reemplazamos el método `super()` por el nombre de la clase de la que queremos reutilizar el método:

```
class B:
```

```
    def x(self):
        print('x: B')

class C:
    def x(self):
        print('x: C')

class D(B, C):
    def x(self):
        C.x(self)    # Se ha de pasar self al método

d = D()
d.x()     # x: C
```

Además, cuando invocamos de esta forma un método de una de las clases madres hemos pasar como primer parámetro del mismo `self` para que la clase madre disponga del objeto sobre el que aplicar el método.

El funcionamiento es el mismo para cualquier método, en particular también para el constructor:

```
class SeaAnimal:
    def __init__(self, name):
        print("SeaAnimal INIT")
        self.name = name

    def swim(self):
        return "Swimming..."

class LandAnimal:
    def __init__(self, age):
        print("LandAnimal INIT")
        self.age = age

    def walk(self):
        print("Walking...")

class Penguin(SeaAnimal, LandAnimal):
    def __init__(self, name, age):
        SeaAnimal.__init__(self, name=name)
        LandAnimal.__init__(self, age=age)

penguin = Penguin("Kowalski", 3)
print(penguin.name, penguin.age)
```

Supuesto 1

Creamos la clase `CocheHibrido` que hereda de las clases `CocheCombustion` y `CocheElectrico` que vimos en la actividad anterior

```
class Vehiculo:
    def __init__(self, matricula, color, num_ruedas):
        self._matricula = matricula
        self._color = color
        self._num_ruedas = num_ruedas

    def get_matricula(self):
```

```
        return self._matricula

    def __str__(self):
        return f"Vehículo con matrícula {self.get_matricula()}, de color
{self._color} y {self._num_ruedas} ruedas."

class CocheCombustion(Vehiculo):
    def __init__(self, matricula, color, num_ruedas, cilindrada):
        super().__init__(matricula, color, num_ruedas)
        self._cilindrada = cilindrada

    def get_cilindrada(self):
        return self._cilindrada

    def descripcion(self):
        return f"Coche con motor de combustión de {self.get_cilindrada()} cc,
matrícula {self.get_matricula()}, de color {self._color} y {self._num_ruedas}
ruedas."

class CocheElectrico(Vehiculo):
    def __init__(self, matricula, color, num_ruedas, potencia):
        super().__init__(matricula, color, num_ruedas)
        self._potencia = potencia

    def get_potencia(self):
        return self._potencia

    def descripcion(self):
        return f"Coche con motor eléctrico de {self.get_potencia()}CV,
matrícula {self.get_matricula()}, de color {self._color} y {self._num_ruedas}
ruedas."

class Moto(Vehiculo):
    def descripcion(self):
        return f"Moto, matrícula {self.get_matricula()}, de color {self._color}
y {self._num_ruedas} ruedas."
```

Reescribimos los métodos de la clase `CocheHibrido` de forma que el resultado de los mismos sea coherente. Aprovechamos los métodos de las clases madre siempre que sea posible.

```
class Vehiculo:
    def __init__(self, matricula, color, num_ruedas):
        self._matricula = matricula
        self._color = color
        self._num_ruedas = num_ruedas

    def get_matricula(self):
        return self._matricula

    def __str__(self):
        return f"Vehículo con matrícula {self.get_matricula()}, de color
{self._color} y {self._num_ruedas} ruedas."

class CocheCombustion(Vehiculo):
    def __init__(self, matricula, color, num_ruedas, cilindrada):
        Vehiculo.__init__(self, matricula, color, num_ruedas)
        self._cilindrada = cilindrada
```

```
    def get_cilindrada(self):
        return self._cilindrada

    def descripcion(self):
        return f"Coche con motor de combustión de {self.get_cilindrada()} cc,
matrícula {self.get_matricula()}, de color {self._color} y {self._num_ruedas}
ruedas."

class CocheElectrico(Vehiculo):
    def __init__(self, matricula, color, num_ruedas, potencia):
        super().__init__(matricula, color, num_ruedas)
        self._potencia = potencia

    def get_potencia(self):
        return self._potencia

    def descripcion(self):
        return f"Coche con motor eléctrico de {self.get_potencia()}CV,
matrícula {self.get_matricula()}, de color {self._color} y {self._num_ruedas}
ruedas."

class Moto(Vehiculo):
    def descripcion(self):
        return f"Moto, matrícula {self.get_matricula()}, de color {self._color}
y {self._num_ruedas} ruedas."

class CocheHibrido(CocheCombustion, CocheElectrico):
    def __init__(self, matricula, color, num_ruedas, cilindrada, potencia):
        CocheCombustion.__init__(self, matricula, color, num_ruedas,
cilindrada)
        self._potencia = potencia

    def __str__(self):
        result = super().__str__()
        result += f"\n Una cilindrada de {self._cilindrada} CC y una potencia
de {self._potencia} CV"
        return result

ch = CocheHibrido("1234GHJ", "Gris", 4, 1500, 230)
print(ch)
```

tags: pro ut2 herencia múltiple oop poo