Herencia en Python 3

1. ¿Qué es la herencia en Python 3?

La herencia es el mecanismo que se utiliza para crear jerarquías de clases relacionadas. Estas clases relacionadas compartirán una interfaz común que se definirá en la clase base. Las clases derivadas de la clase base pueden especializar la interfaz proporcionando una implementación particular cuando corresponda.

```
class Coche():
    """Esta clase representa un coche."""

    def __init__(self, modelo, potencia, consumo):
        """Inicializa los atributos de instancia.

        Argumentos posicionales:
        modelo -- string que representa el modelo del coche
        potencia -- int que representa la potencia en cv
        consumo -- int que representa el consumo en litros/100km
        """
        self.modelo = modelo
        self.potencia = potencia
        self.consumo = consumo
        self._km_actuales = 0

    def especificaciones(self):
        """Muestra las especificaciones del coche."""
        print("Modelo:", self.modelo,
              "\nPotencia: {} cv".format(self.potencia),
              "\nConsumo: {} l/100km".format(self.consumo),
              "\nKilometros actuales:", self._km_actuales)

    @property
    def kilometros(self):
        return self._km_actuales

    @kilometros.setter
    def kilometros(self, kilometros):
        """Actualiza los kilometros actuales del coche."""
        if kilometros > self._km_actuales:
            self._km_actuales = kilometros
        else:
            print("ERROR: No se puede establecer un valor de km inferior al actu

    def consumo_total(self):
        """Muestra el consumo total del coche desde el kilometro 0."""
        consumo_total = (self._km_actuales / 100) * self.consumo
```

```
        consumo_total = (self._km_actuales / 100) * self.consumo
        print("El consumo total es de {} litros".format(consumo_total))
```

¿Qué sucede si quiero representar un coche electrico?

```
class Coche():
    """Esta clase representa un coche."""

    def __init__(self, modelo, potencia, consumo):
        """Inicializa los atributos de instancia.

        Argumentos posicionales:
        modelo -- string que representa el modelo del coche
        potencia -- int que representa la potencia en cv
        consumo -- int que representa el consumo
        """
        self.modelo = modelo
        self.potencia = potencia
        self.consumo = consumo
        self._km_actuales = 0
        self._combustible = "l/100km"

    def especificaciones(self):
        """Muestra las especificaciones del coche."""
        print("Modelo:", self.modelo,
              "\nPotencia: {} cv".format(self.potencia),
              "\nConsumo: {} {}".format(self.consumo, self._combustible),
              "\nKilometros actuales:", self._km_actuales)

    @property
    def kilometros(self):
        return self._km_actuales

    @kilometros.setter
    def kilometros(self, kilometros):
        """Actualiza los kilometros actuales del coche."""
        if kilometros > self._km_actuales:
            self._km_actuales = kilometros
        else:
            print("ERROR: No se puede establecer un valor de km inferior al actu

    def consumo_total(self):
        """Muestra el consumo total del coche desde el kilometro 0."""
        consumo_total = (self._km_actuales / 100) * self.consumo
        print("El consumo total es de {} litros".format(consumo_total))


tesla = Coche("tesla model 3", 300, 15) # consumo: 15Kwh/100km


tesla.especificaciones()
```

```
Modelo: tesla model 3
Potencia: 300 cv
```

```
Consumo: 15 l/100km
Kilometros actuales: 0
```

En este punto podría crear un metodo *setter* que me permitiese modificar el valor del combustible, sin embargo, esto conduciría a una clase muy compleja y con mucho código. Este tipo de problemas es mejor resolverlos utilizando herencia.

```
class CocheElectrico(Coche):
    """Esta clase representa un coche electrico."""

    def __init__(self, modelo, potencia, consumo):
        """Inicializa los atributos de la clase padre."""
        super().__init__(modelo, potencia, consumo)
        self._combustible = "KWh/100km"


tesla = CocheElectrico("tesla model 3", 300, 15)


tesla.especificaciones()
```

```
Modelo: tesla model 3
Potencia: 300 cv
Consumo: 15 KWh/100km
Kilometros actuales: 0
```

2. Definición de atributos y métodos propios en la clase hija

Otra de las cosas que podemos hacer es extender el comportamiento de la clase padre añadiendo nuevos métodos y atributos en la clase hija.

```
class CocheElectrico(Coche):
    """Esta clase representa un coche electrico."""

    def __init__(self, modelo, potencia, consumo, capacidad_bateria):
        """Inicializa los atributos de la clase padre."""
        super().__init__(modelo, potencia, consumo)
        self._combustible = "KWh/100km"
        self._capacidad_bateria = capacidad_bateria

    def detalles_bateria(self):
        """Muestra los detalles de la bateria del coche electrico."""
        print("El tamaño de la batería es: {} KWh".format(self._capacidad_bateri


tesla = CocheElectrico("tesla model 3", 300, 15, 50)


tesla.especificaciones()
```

```
Modelo: tesla model 3
Potencia: 300 cv
Consumo: 15 KWh/100km
Kilometros actuales: 0
```

```
tesla.detalles_bateria()
```

```
El tamaño de la batería es: 50 KWh
```

3. Sobreescribir métodos de la clase padre

En algunas ocasiones, es posible que alguno de los métodos de la clase padre no encaje bien con la clase hija que se ha definido. En estos casos, podemos sobreescribir el método de la clase padre dentro de la clase hija.

```
tesla.kilometros = 100
```

```
tesla.consumo_total()
```

```
El consumo total es de 15.0 litros
```

```
class CocheElectrico(Coche):
    """Esta clase representa un coche electrico."""

    def __init__(self, modelo, potencia, consumo, capacidad_bateria):
        """Inicializa los atributos de la clase padre."""
        super().__init__(modelo, potencia, consumo)
        self._combustible = "KWh/100km"
        self._capacidad_bateria = capacidad_bateria

    def detalles_bateria(self):
        """Muestra los detalles de la bateria del coche electrico."""
        print("El tamaño de la batería es: {} KWh".format(self._capacidad_bateri

    def consumo_total(self):
        """Muestra el consumo total del coche desde el kilometro 0."""
        consumo_total = (self._km_actuales / 100) * self.consumo
        print("El consumo total es de {} kWh".format(consumo_total))
```

```
tesla = CocheElectrico("tesla model 3", 300, 15, 50)
```

```
tesla.kilometros = 14523452
```

```
tesla.consumo_total()
```

```
El consumo total es de 2178517.8 kWh
```

4. Objetos dentro de una clase

Es posble que en algunos casos de uso, determinadas propiedades de una clase tenga suficiente entidad como para convertirse en una clase propia. En estos casos, podemos asginar un objeto de esta segunda a clase a un atributo de la primera.

```
class Bateria:
    """Esta clase va a representar una bateria de un coche electrico."""

    def __init__(self, capacidad, tipo_pila, num_pilas, peso):
        self._capacidad = capacidad
        self._tipo_pila = tipo_pila
        self._num_pilas = num_pilas
        self._peso = peso

    def especificaciones(self):
        print("Capacidad {} kWh".format(self._capacidad),
             "\nTipo de pilas:", self._tipo_pila,
             "\nNumero de pilas:", self._num_pilas,
             "\nPeso de la bateria: {} kg".format(self._peso))


bateria_tesla_modelS = Bateria(80, 2170, 203_136, 480)


bateria_tesla_modelS.especificaciones()
```

```
Capacidad 80 kWh
Tipo de pilas: 2170
Numero de pilas: 203136
Peso de la bateria: 480 kg
```

```
class CocheElectrico(Coche):
    """Esta clase representa un coche electrico."""

    def __init__(self, modelo, potencia, consumo, bateria):
        """Inicializa los atributos de la clase padre."""
        super().__init__(modelo, potencia, consumo)
        self._combustible = "KWh/100km"
        self._bateria = bateria

    def detalles_bateria(self):
        """Muestra los detalles de la bateria del coche electrico."""
        self._bateria.especificaciones()

    def consumo_total(self):
        """Muestra el consumo total del coche desde el kilometro 0."""
        consumo_total = (self._km_actuales / 100) * self.consumo
        print("El consumo total es de {} kWh".format(consumo_total))
```

```
tesla = CocheElectrico("tesla model S", 450, 20, bateria_tesla_modelS)
```

```
tesla.detalles_bateria()
```

```
     Capacidad 80 kWh
     Tipo de pilas: 2170
     Numero de pilas: 203136
     Peso de la bateria: 480 kg
```

```
tesla_2
```