Programación Avanzada
Práctica 02 - Clases II

Pedro Fernando Flores Palmeros

Febrero 2020

1 Trabajando con Clases

1.1 Introducción

En la práctica se comenzó´ a trabajar con clases y se definió una clase Dog en la se definían dos atributos y dos metodos. A continuación, se propone una clase que servirá de base para los ejemplos que se mostraran más adelante.

```
class Car():
    """Clase tipo coche""" def __init__(self, make,
    model, year): """Inicializacion de los atributos"""
    self.make = make self.model = model self.year =
    year

    def get_descriptive_name(self):
        """ Imprime las caracteristicas en la pantalla""" long_name =
        str(self.year)+' '+self.make + ' '+self.model return long_name.title()


my_new_car = Car('audi','a4',2020)
print(my_new_car.get_descriptive_name())
```

Ejercicio 1: Capture el código y observe lo que aparece en la terminal, guarde el código como Ejercicio_01.py y suba el código a un nuevo repositorio en GitHub llamado Practica_02.

```python
class Car():
    """Clase tipo coche"""
    def __init__(self, make, model, year):
        """Inicializacion de los atributos"""
        self.make = make
        self.model = model
        self.year = year
    def get_descriptive_name(self):
        """Imprime las caracteristicas en la pantalla"""
        long_name = str(self.year)+' '+self.make + ' '+self.model
        return long_name.title()

my_new_car = Car('audi','a4',2020)
print(my_new_car.get_descriptive_name())
```

```
edwin@edwin-virtual-machine:~/Desktop$ python Ejercicio_01.py
2020 Audi A4
```

1.2 Utilizando valores predeterminados para algún atributo

Todos los atributos de una clase necesitan ser inicializados, ya sea con 0 o con una string vacía. En algunos casos no es necesario que el atributo que tienen un valor definido reciba algún valor a través del constructor ya que no tendría caso. En el caso del código anterior, se agregará un atributo predefinido odometer_reading, este atributo tendrá un valor de 0 y observe que no estará´ en como argumento para el constructor

---

```
class Car():
    """Clase tipo coche""" def __init__(self, make,
    model, year): """Inicializacion de los atributos"""
    self.make = make self.model = model self.year =
    year self.odometer_reading = 0

    def get_descriptive_name(self):
        """ Imprime las caracteristicas en la pantalla""" long_name =
        str(self.year)+' '+self.make + ' '+self.model return long_name.title()

    def read_odometer(sefl):
        """Imprime los kilometros recorrido por el auto""" print("This car has " +
        str(sefl.odometer_reading) + " miles on it")
```

```
my_new_car = Car('audi','a4',2020)
print(my_new_car.get_descriptive_name())
my_new_car.read_odometer()
```

Ejercicio 2:

- Capture el programa anterior y ejecútelo, analice la salida del programa.

```
class Car():
    """Clase tipo coche"""
    def __init__(self, make, model, year):
        """Inicializacion de los atributos"""
        self.make = make
        self.model = model
        self.year = year
        self.odometer_reading = 0
    def get_descriptive_name(self):
        """ Imprime las caracteristicas en la pantalla"""
        long_name = str(self.year)+' '+self.make + ' '+self.model
        return long_name.title()
    def read_odometer(sefl):
        """Imprime los kilometros recorrido por el auto"""
        print("This car has " + str(sefl.odometer_reading) + " miles on it")
my_new_car = Car('audi','a4',2020)
print(my_new_car.get_descriptive_name())
my_new_car.read_odometer()
```

```
edwin@edwin-virtual-machine:~/Desktop$ python Ejercicio_02.py
2020 Audi A4
This car has 0 miles on it
```

- borre self.odometer_reading del constructor y trate de definirlo dentro de read_odometer(self), trate de ejecutar el programa y anote la salida de la terminal. Escriba la razón del error que aparece.

```python
class Car():
    """Clase tipo coche"""
    def __init__(self, make, model, year):
        """Inicializacion de los atributos"""
        self.make = make
        self.model = model
        self.year = year

    def get_descriptive_name(self):
        """ Imprime las caracteristicas en la pantalla"""
        long_name = str(self.year)+' '+self.make + ' '+self.model
        return long_name.title()
    def read_odometer(sefl):
        self.odometer_reading = 0
        """Imprime los kilometros recorrido por el auto"""
        print("This car has " + str(sefl.odometer_reading) + " miles on it")
my_new_car = Car('audi','a4',2020)
print(my_new_car.get_descriptive_name())
my_new_car.read_odometer()
```

```
Traceback (most recent call last):
  File "Ejercicio_02.py", line 19, in <module>
    my_new_car.read_odometer()
  File "Ejercicio_02.py", line 14, in read_odometer
    self.odometer_reading = 0
NameError: global name 'self' is not defined
```

El error es debido a que el atributo no se definió en el constructor, entonces Python no lo detecta sino se define ahí

- Agregue dos atributos predefinidos a la clase Car
- Agregue el último programa al repositorio de la práctica 2 con el nombre de Ejercicio_02.py

1.3 Modificando el valor de los atributos

La manera más sencilla de modificar los atributos es de la misma manera en que se ingresa a ellos para imprimirlos en la terminal, tal como se muestra en el siguiente programa

---

```
class Car():
    """Clase tipo coche""" def __init__(self, make,
    model, year): """Inicializacion de los atributos"""
    self.make = make
        self.model = model self.year =
        year self.odometer_reading = 0

    def get_descriptive_name(self):
```

```
        """ Imprime las caracteristicas en la pantalla""" long_name =
        str(self.year)+' '+self.make + ' '+self.model return long_name.title()

    def read_odometer(sefl):
        """Imprime los kilometros recorrido por el auto""" print("This car has " +
        str(sefl.odometer_reading) + " miles on it")


my_new_car = Car('audi','a4',2020)
print(my_new_car.get_descriptive_name())
my_new_car.odometer_reading = 23
my_new_car.read_odometer()
```

Ejercicio 3

- Escribir el código del ejemplo y verificar el funcionamiento del mismo, subir el código al repositorio con el nombre de Ejercicio_3_1.py

```python
class Car():
    """Clase tipo coche"""
    def __init__(self, make, model, year):
        """Inicializacion de los atributos"""
        self.make = make
        self.model = model
        self.year = year
        self.odometer_reading = 0
    def get_descriptive_name(self):
        """ Imprime las caracteristicas en la pantalla"""
        long_name = str(self.year)+ ''+self.make +' '+self.model
        return long_name.title()
    def read_odometer(sefl):
        """Imprime los kilometros recorrido por el auto"""
        print("This car has " + str(sefl.odometer_reading) + " miles on it")
my_new_car = Car('audi','a4',2020)
print(my_new_car.get_descriptive_name())
my_new_car.odometer_reading = 23
my_new_car.read_odometer()
```

```
edwin@edwin-virtual-machine:~/Desktop$ python Ejercicio_03.py
2020Audi A4
This car has 23 miles on it
```

- Retomar el código que usted implementó en el Ejercicio 2 y modifique los valores predeterminados de los nuevos atributos, subir el código al repositorio con el nombre de Ejercicio_3_2.py

```python
class Car():
    """Clase tipo coche"""
    def __init__(self, make, model, year):
        """Inicializacion de los atributos"""
        self.make = make
        self.model = model
        self.year = year
        self.odometer_reading = 0
    def get_descriptive_name(self):
        """ Imprime las caracteristicas en la pantalla"""
        long_name = str(self.year)+' '+self.make + ' '+self.model
        return long_name.title()
    def read_odometer(sefl):
        """Imprime los kilometros recorrido por el auto"""
        print("This car has " + str(sefl.odometer_reading) + " miles on it")
my_new_car = Car('mustang','shelby',2020)
print(my_new_car.get_descriptive_name())
my_new_car.read_odometer()
```

```
edwin@edwin-virtual-machine:~/Desktop$ python Ejercicio_02.py
2020 Mustang Shelby
This car has 0 miles on it
```

1.4 Modificando los valores a través de los métodos

De manera general el usuario no deber´ıa de tener acceso a los atributos de una clase, ya que los puede modificar accidentalmente y con ello modificar´ıa el flujo del programa y consecuentemente la ecuci´on del mismo obteniendo resultados erro´neos. Para evitar que el usuario tenga de acceso de manera directa a los atributos, se generan m´etodos para modificar el atributo.

```
class Car():
    """Clase tipo coche""" def __init__(self, make,
    model, year): """Inicializacion de los atributos"""
    self.make = make self.model = model self.year =
    year self.odometer_reading = 0

    def get_descriptive_name(self):
        """" Imprime las caracteristicas en la pantalla""" long_name =
        str(self.year)+' '+self.make + ' '+self.model return long_name.title()
```

```
    def read_odometer(sefl):
        """Imprime los kilometros recorrido por el auto""" print("This car has " +
        str(sefl.odometer_reading) + " miles on it")

    def update_odometer(self, mileage):
        """Modifica el valor del metodo desde la funcion""" self.odometer_reading = mileage


my_new_car = Car('audi','a4',2020)
print(my_new_car.get_descriptive_name())
my_new_car.update_odometer(23)
my_new_car.read_odometer()
```

Ejercicio 4

- Escribir el código del ejemplo y verificar el funcionamiento del mismo, subir el código al repositorio con el nombre de Ejercicio_4_1.py

```python
class Car():
    """Clase tipo coche"""
    def __init__(self, make, model, year):
        """Inicializacion de los atributos"""
        self.make = make
        self.model = model
        self.year = year
        self.odometer_reading = 0
    def get_descriptive_name(self):
        """ Imprime las caracteristicas en la pantalla"""
        long_name = str(self.year)+' '+self.make + ' '+self.model
        return long_name.title()
    def read_odometer(sefl):
        """Imprime los kilometros recorrido por el auto"""
        print("This car has " + str(sefl.odometer_reading) + " miles on it")
    def update_odometer(self, mileage):
        """Modifica el valor del metodo desde la funcion"""
        self.odometer_reading = mileage
my_new_car = Car('audi','a4',2020)
print(my_new_car.get_descriptive_name())
my_new_car.update_odometer(23)
my_new_car.read_odometer()
```

```
edwin@edwin-virtual-machine:~/Desktop$ python Ejercicio_4_1.py
2020 Audi A4
This car has 23 miles on it
```

- Retomar el código que usted implementó en el Ejercicio 2 y modifique los valores predeterminados de los nuevos atributos a través de métodos que usted diseñe, subir el código al repositorio con el nombre de Ejercicio_4_2.py

```python
class Car():
    """Clase tipo coche"""
    def __init__(self, make, model, year):
        """Inicializacion de los atributos"""
        self.make = make
        self.model = model
        self.year = year
        self.odometer_reading = 0
    def get_descriptive_name(self):
        """ Imprime las caracteristicas en la pantalla"""
        long_name = str(self.year)+' '+self.make + ' '+self.model
        return long_name.title()
    def read_odometer(sefl):
        """Imprime los kilometros recorrido por el auto"""
        print("This car has " + str(sefl.odometer_reading) + " miles on it")
    def update_odometer(self, mileage):
        """Modifica el valor del metodo desde la funcion"""
        self.odometer_reading = mileage
    def change_make(self,new_make):
        self.make=new_make
    def change_model(self,new_model):
        self.model=new_model
    def change_year(self,new_year):
        self.year=new_year

my_new_car = Car('audi','a4',2020)
print(my_new_car.get_descriptive_name())
my_new_car.change_make('mustang')
my_new_car.change_model('shelby')
my_new_car.change_year(23)
my_new_car.update_odometer(23)
my_new_car.read_odometer()
```

```
edwin@edwin-virtual-machine:~/Desktop$ python Ejercicio_02.py
2020 Mustang Shelby
This car has 0 miles on it
```

Dentro de los métodos también puede haber operaciones lógicas o diferentes procesos, en este caso se agregará una condición para que, cuando se quiera modificar el kilometraje sólo se pueda agregar y no quitar

---

```
class Car():
    """Clase tipo coche""" def __init__(self, make,
    model, year): """Inicializacion de los atributos"""
    self.make = make self.model = model self.year =
    year self.odometer_reading = 0

    def get_descriptive_name(self):
        """ Imprime las caracteristicas en la pantalla""" long_name =
        str(self.year)+' '+self.make + ' '+self.model return long_name.title()

    def read_odometer(sefl):
        """Imprime los kilometros recorrido por el auto""" print("This car has " +

    str(sefl.odometer_reading) + " miles on it") def update_odometer(self, mileage):

        """Modifica el valor del metodo desde la funcion""" if mileage >=
        self.odometer_reading:
            self.odometer_reading = mileage
        else: print("You cannot do that")



my_new_car = Car('audi','a4',2020)
print(my_new_car.get_descriptive_name())
my_new_car.update_odometer(23)
my_new_car.read_odometer()
my_new_car.update_odometer(50)
my_new_car.read_odometer()
my_new_car.update_odometer(10)
my_new_car.read_odometer()
```

Ejercicio 5" Escribir el código del ejemplo y verificar el funcionamiento del mismo, subir el código al repositorio con el nombre de Ejercicio 5.py

```python
class Car():
    """Clase tipo coche"""
    def __init__(self, make, model, year):
        """Inicializacion de los atributos"""
        self.make = make
        self.model = model
        self.year = year
        self.odometer_reading = 0
    def get_descriptive_name(self):
        """ Imprime las caracteristicas en la pantalla"""
        long_name = str(self.year)+' '+self.make + ' '+self.model
        return long_name.title()
    def read_odometer(sefl):
        """Imprime los kilometros recorrido por el auto"""
        print("This car has "+str(sefl.odometer_reading)+ "miles on it")
    def update_odometer(self, mileage):
        """Modifica el valor del metodo desde la funcion"""
        if mileage >= self.odometer_reading:
            self.odometer_reading = mileage
        else: print("You cannot do that")
my_new_car = Car('audi','a4',2020)
print(my_new_car.get_descriptive_name())
my_new_car.update_odometer(23)
my_new_car.read_odometer()
my_new_car.update_odometer(50)
my_new_car.read_odometer()
my_new_car.update_odometer(10)
my_new_car.read_odometer()
```

```
edwin@edwin-virtual-machine:~/Desktop$ python Ejercicio_05.py
2020 Audi A4
This car has 23miles on it
This car has 50miles on it
You cannot do that
This car has 50miles on it
```

2 Tarea

- Del ejercicio de restaurantes de la práctica 1. Agregar un atributo llamado number_served con un valor predefinido de 0. Crear una instancia del Restaurante (Crear el objeto). Imprimir el número de clientes que se han atendido, modifique el atributo numer_served y vuelva a imprimir el número de cliente que se han atendido. (Tarea_1_1)

```
class restaurant():
    """Clase tipo restaurante"""
    def __init__(self,restaurant_name,restaurant_type,restaurant_OC):
        """Inicializacion de los atributos"""
        self.restaurant_name=restaurant_name
        self.restaurant_type=restaurant_type
        self.restaurant_OC=restaurant_OC
        self.client_serve = 0
    def get_descriptive_name(self):
        """ Imprime las caracteristicas en la pantalla"""
        long_name = self.restaurant_name+' '+self.restaurant_type + ' '+self.restaurant_OC
        return long_name.title()
    def read_number(sefl):
        """Imprime los clientes qeu se han atendido"""
        print("se han atendido "+str(sefl.client_serve)+ " clientes.")
my_new_restaurant=restaurant('Chiringuito','Mexicana','open')
print(my_new_restaurant.get_descriptive_name())
my_new_restaurant.read_number()
```

```
edwin@edwin-virtual-machine:~/Desktop$ python Tarea_1_1.py
Chiringuito Mexicana Open
se han atendido 0 clientes.
```

Agregar un método llamado set_number_served() que permita incrementar el número de cliente que se han servido. Llame este método modificando el número de clientes, agregando la lógica que el número de clientes sólo puede aumentar, no puede disminuir y cada vez que se modifique el número de clientes llamar a un método que se encargue de imprimir el número de clientes que se han atendido. (Tarea_1_2)

```python
class restaurant():
    """Clase tipo restaurante"""
    def __init__(self,restaurant_name,restaurant_type,restaurant_OC):
        """Inicializacion de los atributos"""
        self.restaurant_name=restaurant_name
        self.restaurant_type=restaurant_type
        self.restaurant_OC=restaurant_OC
        self.client_serve = 0
    def get_descriptive_name(self):
        """ Imprime las caracteristicas en la pantalla"""
        long_name = self.restaurant_name+' '+self.restaurant_type + ' '+self.restaurant_OC
        return long_name.title()
    def read_number(sefl):
        """Imprime los clientes qeu se han atendido"""
        print("se han atendido "+str(sefl.client_serve)+ " clientes.")
    def set_number_served(self,client):
        if client >= self.client_serve:
            self.client_serve = client
        else: print("You cannot do that")
my_new_restaurant=restaurant('Chiringuito','Mexicana','open')
print(my_new_restaurant.get_descriptive_name())
my_new_restaurant.read_number()
my_new_restaurant.set_number_served(23)
my_new_restaurant.read_number()
my_new_restaurant.set_number_served(50)
my_new_restaurant.read_number()
my_new_restaurant.set_number_served(10)
my_new_restaurant.read_number()
```

```
edwin@edwin-virtual-machine:~/Desktop$ python Tarea_1_2.py
Chiringuito Mexicana Open
se han atendido 0 clientes.
se han atendido 23 clientes.
se han atendido 50 clientes.
You cannot do that
se han atendido 50 clientes.
```

- Del ejercicio relacionado al nombre de usuario de la práctica 1, agregar método llamado increment_login_attempts() que incremente el número de intentos de ingresar a la cuenta. Escribir un método reset_loging_attempts que resetee el contador de intentos.

  Generar un método que se llame secure_account en el cual si el número de intentos es mayor a 10, imprima alguna clase de error. Tarea_2_2

```python
class user():
    def __init__(self,first_name,last_name,pasword,mail):
        self.first_name=first_name
        self.last_name=last_name
        self.pasword=password
        self.mail=mail
        self.user_attempt = 0
        self.reset=0
    def describe_user(self):
        print('\nName of user: '+self.first_name)
        print('\nlast name of user: '+self.last_name)
        print('\npasword: '+self.password)
        print('\nmail: '+self.mail)
    def greet_user(self):
        print('Hola, '+self.first_name+', welcome to the program.')
    def increment_login_attempts(self,us):
        if us <= 10:
            self.user_attempt = us
        else: print("you tried too many times to log in")
    def reset_loging_attempts(self):
        self.reset=user_attempt
    def read_number(sefl):
        """Imprime los clientes qeu se han atendido"""
        print("se han intentado iniciar sesion "+str(sefl.user_attempt)+ " veces.")
my_user=user('Edwin','Gonzalez','Soporte321','edwingonzales@gmail.es')
print(my_user.describe_user())
my_user.increment_login_attempts(1)
my_user.read_number()
my_user.increment_login_attempts(2)
my_user.read_number()
my_user.increment_login_attempts(3)
my_user.read_number()
my_user.increment_login_attempts(4)
my_user.read_number()
my_user.increment_login_attempts(6)
my_user.read_number()
my_user.increment_login_attempts(7)
my_user.read_number()
my_user.increment_login_attempts(8)
my_user.read_number()
my_user.increment_login_attempts(9)
my_user.read_number()
my_user.increment_login_attempts(10)
my_user.read_number()
my_user.increment_login_attempts(11)
my_user.read_number()
my_user.increment_login_attempts(12)
my_user.read_number()
```

```
edwin@edwin-virtual-machine:~/Desktop$ python Tarea_2_2.py

Name of user: Edwin

last name of user: Gonzalez

pasword: Soporte321

mail: edwingonzales@gmail.es
None
se han intentado iniciar sesion 1 veces.
se han intentado iniciar sesion 2 veces.
se han intentado iniciar sesion 3 veces.
se han intentado iniciar sesion 4 veces.
se han intentado iniciar sesion 6 veces.
se han intentado iniciar sesion 7 veces.
se han intentado iniciar sesion 8 veces.
se han intentado iniciar sesion 9 veces.
se han intentado iniciar sesion 10 veces.
you tried too many times to log in
se han intentado iniciar sesion 10 veces.
you tried too many times to log in
se han intentado iniciar sesion 10 veces.
```

3 Herencia

Usualmente no siempre se tiene que comenzar desde el comienzo al momento de escribir una clase. Si la clase que se desea implementar es una versio´n especializada de otra clase, entonces se puede utilizar la *herencia*. Cuando una clase hereda de otra, entonces, automa´ticamente copia los atibutos y m´etodos de la primera clase. La clase original se le conoce como Padre mientras que la nueva clase se le conoce como hij@. La clase hija hereda todos los atributos de la clase padre, pero tambi´en es libre de definir nuevos atributos y m´etodos para s´ı misma.

3.1 El constructor de la clase hija

La primero que hace Python cuando crea una nueva instancia de la clase hija es asignar valores a los atributos de la clase Padre. Para esto el método __init__() de la clase hija necesita ayuda de la clase Padre.

Como ejemplo se puede retomar la clase Car de los apartados anteriores, supongamos que ahora se quiere crear un auto eléctrico, entonces, el auto eléctrico hereda de Car. Las clases querían como se muestra a continuación:

```
class Car():
    """Un intento de representar un auto"""
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year
        self.odometer_reading = 0
    def get_descriptive_name(self):
        long_name = str(self.year) + ' ' + self.make + ' ' + self.model

        return long_name.title()

    def read_odometer(self):

     print("This car has " + str(self.odometer_reading) + " miles on it.")

    def update_odometer(self, mileage):
        if mileage >= self.odometer_reading:
            self.odometer_reading = mileage
        else: print("You can't roll back an odometer!")

    def increment_odometer(self, miles):
        self.odometer_reading += miles


class ElectricCar(Car):
    """Un intento de representar un auto electrico""" def
    __init__(self,make,model,year):
        """ Inicializa los atributos de la clase padre """

        super().__init__(make,model,year)

my_tesla = ElectricCar('tesla','model s',2020) print(my_tesla.get_descriptive_name())
```

Ejercicio 6:

- Capturar el código anterior y ejecutarlo, analizar la salida en la terminal, (Ejercicio_6_1.py)

```python
class Car():
    """Un intento de representar un auto"""
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year
        self.odometer_reading = 0
    def get_descriptive_name(self):
        long_name = str(self.year) + ',' + self.make + ',' + self.model
        return long_name.title()
    def read_odometer(self):
        print("This car has " + str(self.odometer_reading) + " miles on it.")
    def update_odometer(self, mileage):
        if mileage >= self.odometer_reading:
            self.odometer_reading = mileage
        else:
            print("You can't roll back an odometer!")
    def increment_odometer(self, miles):
        self.odometer_reading += miles

class ElectricCar(Car):
    """Un intento de representar un auto electrico"""
    def __init__(self,make,model,year):
        """ Inicializa los atributos de la clase padre """
        Car.__init__(self,make,model,year)
my_tesla = ElectricCar('tesla','model s',2020)
print(my_tesla.get_descriptive_name())
```

```
edwin@edwin-virtual-machine:~/Desktop$ python Ejercicio_6_1.py
2020,Tesla,Model S
```

- Hacer una clase HybridCar que herede de la clase Car, definir la clase FuelCar que herede de la clase Car, generar un objeto de cada clase y ejecutar el método get_descriptive_name() de cada objeto. (Ejercicio_6_2.py)

```
class Car():
    """Un intento de representar un auto"""
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year
        self.odometer_reading = 0
    def get_descriptive_name(self):
        long_name = str(self.year) + ',' + self.make + ',' + self.model
        return long_name.title()
    def read_odometer(self):
        print("This car has " + str(self.odometer_reading) + " miles on it.")
    def update_odometer(self, mileage):
        if mileage >= self.odometer_reading:
            self.odometer_reading = mileage
        else:
            print("You can't roll back an odometer!")
    def increment_odometer(self, miles):
        self.odometer_reading += miles

class ElectricCar(Car):
    """Un intento de representar un auto electrico"""
    def __init__(self,make,model,year):
        """ Inicializa los atributos de la clase padre """
        Car.__init__(self,make,model,year)
my_tesla = ElectricCar('tesla','model s',2020)
print(my_tesla.get_descriptive_name())

class HybridCar(Car):
    """Un intento de representar un auto electrico"""
    def __init__(self,make,model,year):
        """ Inicializa los atributos de la clase padre """
        Car.__init__(self,make,model,year)
my_Hybridcar = HybridCar('BMW','i3',2020)
print(my_Hybridcar.get_descriptive_name())
class FuelCar(Car):
    """Un intento de representar un auto electrico"""
    def __init__(self,make,model,year):
        """ Inicializa los atributos de la clase padre """
        Car.__init__(self,make,model,year)
my_FuelCar= FuelCar('mustang','shelby',2020)
print(my_FuelCar.get_descriptive_name())
```

```
edwin@edwin-virtual-machine:~/Desktop$ python Ejercicio_6_2.py
2020,Tesla,Model S
2020,Bmw,I3
2020,Mustang,Shelby
```

3.2 Atributos y métodos de la clase hija

Como se mencionó con anterioridad, las clases hijas además de los atributos y métodos que han heredado, pueden definir sus propios métodos y atributos como se puede verificar en el siguiente ejemplo.

```
class Car():
    """Un intento de representar un auto""" def
    __init__(self, make, model, year):
        self.make = make self.model =
        model self.year = year
        self.odometer_reading = 0
    def get_descriptive_name(self):
        long_name = str(self.year) + ' ' + self.make + ' ' + self.model return long_name.title()

    def read_odometer(self): print("This car has " + str(self.odometer_reading) + " miles on it.")

    def update_odometer(self, mileage):
        if mileage >= self.odometer_reading:
            self.odometer_reading = mileage
        else: print("You can't roll back an odometer!")

    def increment_odometer(self, miles):
        self.odometer_reading += miles


class ElectricCar(Car):
    """Un intento de representar un auto electrico""" def
    __init__(self,make,model,year):
        """ Inicializa los atributos de la clase padre """
        super().__init__(make,model,year)
        self.battery_size = 70

    def describe_battery(self):
        """ Imprime la el tamanio de la bateria """ print("This car has a " +
        str(self.battery_size) + "-kWh battery.")

my_tesla = ElectricCar('tesla','model s',2020)
print(my_tesla.get_descriptive_name()) my_tesla.describe_battery()
```

En la clase hija se pueden agregar los métodos y atributos que el desarrollador desee, no hay límite.

Ejercicio 7:

- Capturar el co´digo anterior y ejecutarlo, analizar la salida en la terminal,(Ejercicio_7_1.py)

```python
class Car():
    """Un intento de representar un auto"""
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year
        self.odometer_reading = 0
    def get_descriptive_name(self):
        long_name = str(self.year) + ' ' + self.make + ' ' + self.model
        return long_name.title()
    def read_odometer(self):
        print("This car has " + str(self.odometer_reading) + " miles on it.")
    def update_odometer(self, mileage):
        if mileage >= self.odometer_reading:
            self.odometer_reading = mileage
        else:
            print("You can't roll back an odometer!")
    def increment_odometer(self, miles):
        self.odometer_reading += miles
class ElectricCar(Car):
    """Un intento de representar un auto electrico"""
    def __init__(self,make,model,year):
        """ Inicializa los atributos de la clase padre """
        Car.__init__(self,make,model,year)
        self.battery_size = 70
    def describe_battery(self):
        """ Imprime la el tamanio de la bateria """
        print("This car has a " + str(self.battery_size) + "-kWh battery.")
my_tesla = ElectricCar('tesla','model s',2020)
print(my_tesla.get_descriptive_name())
my_tesla.describe_battery()
```

```
edwin@edwin-virtual-machine:~/Desktop$ python Ejercicio_7_1.py
2020 Tesla Model S
This car has a 70-kWh battery.
```

- Del Ejercicio_6_2.py agregar un método y un atributo a las clases HybridCar y FuelCar, generar métodos para imprimir los atributos propios de las clases hijas. (Ejercicio_7_2.py)

```python
class Car():
    """Un intento de representar un auto"""
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year
        self.odometer_reading = 0
    def get_descriptive_name(self):
        long_name = str(self.year) + ',' + self.make + ',' + self.model
        return long_name.title()
    def read_odometer(self):
        print("This car has " + str(self.odometer_reading) + " miles on it.")
    def update_odometer(self, mileage):
        if mileage >= self.odometer_reading:
            self.odometer_reading = mileage
        else:
            print("You can't roll back an odometer!")
    def increment_odometer(self, miles):
        self.odometer_reading += miles

class ElectricCar(Car):
    """Un intento de representar un auto electrico"""
    def __init__(self,make,model,year):
        """ Inicializa los atributos de la clase padre """
        Car.__init__(self,make,model,year)
        self.battery_size = 70
    def describe_battery(self):
        """ Imprime la el tamanio de la bateria """
        print("This car has a " + str(self.battery_size) + "-kWh battery.")

class HybridCar(Car):
    """Un intento de representar un auto electrico"""
    def __init__(self,make,model,year):
        """ Inicializa los atributos de la clase padre """
        Car.__init__(self,make,model,year)
        self.battery_size = 70
        self.tank_gas= 40
    def describe_meditions(self):
        """ Imprime la el tamanio de la bateria """
        print("This car has a " + str(self.battery_size) + "-kWh battery."+'and'+str(self.tank_gas)+'liters of gasoline')

class FuelCar(Car):
    """Un intento de representar un auto electrico"""
    def __init__(self,make,model,year):
        """ Inicializa los atributos de la clase padre """
        Car.__init__(self,make,model,year)
        self.tank_gas = 40
    def describe_Tank(self):
        """ Imprime la el tamanio de la bateria """
        print("This car has a " + str(self.tank_gas) + "liters of gasoline")

my_tesla = ElectricCar('tesla','model s',2020)
print(my_tesla.get_descriptive_name())
my_tesla.describe_battery()
my_hybrid_car = HybridCar('BMW','i9',2020)
print(my_hybrid_car.get_descriptive_name())
my_hybrid_car.describe_meditions()
my_FuelCar = FuelCar('Mustang','shelby',2020)
print(my_FuelCar.get_descriptive_name())
my_FuelCar.describe_Tank()
```

```
edwin@edwin-virtual-machine:~/Desktop$ python Ejercicio_7_2.py
2020,Tesla,Model S
This car has a 70-kWh battery.
2020,Bmw,I9
This car has a 70-kWh battery.and40liters of gasoline
2020,Mustang,Shelby
This car has a 40liters of gasoline
```

3.3 Sobrecarga de métodos de la clase padre

Se puede hacer sobrecarga de métodos si algún elemento de la clase padre no se ajusta a la clase hija, para esto es necesario que en la clase hija se declare la función con el mismo nombre de la clase padre, Python utilizará la función sobrecargada de la clase hija.

```
class Car():
    """Un intento de representar un auto""" def
    __init__(self, make, model, year):
        self.make = make self.model =
        model self.year = year
        self.odometer_reading = 0


    def get_descriptive_name(self):
        long_name = str(self.year) + ' ' + self.make + ' ' + self.model return long_name.title()

    def read_odometer(self): print("This car has " + str(self.odometer_reading) + " miles on it.")

    def update_odometer(self, mileage):
        if mileage >= self.odometer_reading:
            self.odometer_reading = mileage
        else:
            print("You can't roll back an odometer!")

    def increment_odometer(self, miles):
        self.odometer_reading += miles

    def charge(self): print("The car is charging
        ....")


class ElectricCar(Car):
    """Un intento de representar un auto electrico""" def
    __init__(self,make,model,year):
        """ Inicializa los atributos de la clase padre """
        super().__init__(make,model,year) self.battery_size = 70

    def describe_battery(self):
        """ Imprime la el tamanio de la bateria """ print("This car has a " +
        str(self.battery_size) + "-kWh battery.")

    def charge(self): print("The battery is almost empty, the car is charging")
```

```
my_tesla = ElectricCar('tesla','model s',2020)
print(my_tesla.get_descriptive_name())
my_tesla.describe_battery()
my_tesla.charge()
```

Ejercicio 8

- Capturar el co´digo anterior y ejecutarlo, analizar la salida en la terminal,(Ejercicio_8_1.py)

```python
class Car():
    """Un intento de representar un auto"""
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year
        self.odometer_reading = 0
    def get_descriptive_name(self):
        long_name = str(self.year) + ' ' + self.make + ' ' + self.model
        return long_name.title()
    def read_odometer(self):
        print("This car has " + str(self.odometer_reading) + " miles on it.")
    def update_odometer(self, mileage):
        if mileage >= self.odometer_reading:
            self.odometer_reading = mileage
        else:
            print("You can't roll back an odometer!")
    def increment_odometer(self, miles):
        self.odometer_reading += miles
class ElectricCar(Car):
    """Un intento de representar un auto electrico"""
    def __init__(self,make,model,year):
        """ Inicializa los atributos de la clase padre """
        Car.__init__(self,make,model,year)
        self.battery_size = 70
    def describe_battery(self):
        """ Imprime la el tamanio de la bateria """
        print("This car has a " + str(self.battery_size) + "-kWh battery.")
    def charge(self):
        print("The battery is almost empty, the car is charging")


my_tesla = ElectricCar('tesla','model s',2020)
print(my_tesla.get_descriptive_name())
my_tesla.describe_battery()
my_tesla.charge()
```

```
edwin@edwin-virtual-machine:~/Desktop$ python Ejercicio_8_1.py
2020 Tesla Model S
This car has a 70-kWh battery.
The battery is almost empty, the car is charging
```

- Del Ejercicio_7_2.py hacer la sobrecarga de la función charging de las clases HybridCar y FuelCar.(Ejercicio_8_2.py)

```
    def __init__(self,make,model,year):
        """ Inicializa los atributos de la clase padre """
        Car.__init__(self,make,model,year)
        self.battery_size = 70
    def describe_battery(self):
        """ Imprime la el tamanio de la bateria """
        print("This car has a " + str(self.battery_size) + "-kWh battery.")
    def charge(self):
        print("The battery is almost empty, the car is charging")

class HybridCar(Car):
    """Un intento de representar un auto electrico"""
    def __init__(self,make,model,year):
        """ Inicializa los atributos de la clase padre """
        Car.__init__(self,make,model,year)
        self.battery_size = 70
        self.tank_gas= 40
    def describe_meditions(self):
        """ Imprime la el tamanio de la bateria """
        print("This car has a " + str(self.battery_size) + "-kWh battery."+'and'+str(self.tank_gas)+'liters of gasoline')
    def charge(self):
        print("The battery is almost empty, the car is charging")

class FuelCar(Car):
    """Un intento de representar un auto electrico"""
    def __init__(self,make,model,year):
        """ Inicializa los atributos de la clase padre """
        Car.__init__(self,make,model,year)
        self.tank_gas = 40
    def describe_Tank(self):
        """ Imprime la el tamanio de la bateria """
        print("This car has a " + str(self.tank_gas) + "liters of gasoline")
    def charge(self):
        print("the gasoline tanke is empty, refill gasoline as soon as possible")


my_tesla = ElectricCar('tesla','model s',2020)
print(my_tesla.get_descriptive_name())
my_tesla.describe_battery()
my_tesla.charge()

my_hybrid_car = HybridCar('BMW','i9',2020)
print(my_hybrid_car.get_descriptive_name())
my_hybrid_car.describe_meditions()
my_hybrid_car.charge()

my_FuelCar = FuelCar('Mustang','shelby',2020)
print(my_FuelCar.get_descriptive_name())
my_FuelCar.describe_Tank()
my_FuelCar.charge()
```

```
edwin@edwin-virtual-machine:~/Desktop$ python Ejercicio_8_2.py
2020 Tesla Model S
This car has a 70-kWh battery.
The battery is almost empty, the car is charging
2020 Bmw I9
This car has a 70-kWh battery.and40liters of gasoline
The battery is almost empty, the car is charging
2020 Mustang Shelby
This car has a 40liters of gasoline
the gasoline tanke is empty, refill gasoline as soon as possible
```

4 Instancias (Objetos) como atributos

Dentro de la programación orientada a objetos, se puede hacer que las clases tengan objetos de otras clases como atributos a este hecho se le conoce como *composición de clases*.

```
class Car():
    """Un intento de representar un auto""" def
    __init__(self, make, model, year):
        self.make = make self.model =
        model self.year = year
        self.odometer_reading = 0


    def get_descriptive_name(self):
        long_name = str(self.year) + ' ' + self.make + ' ' + self.model return long_name.title()

    def read_odometer(self): print("This car has " + str(self.odometer_reading) + " miles on it.")

    def update_odometer(self, mileage):
        if mileage >= self.odometer_reading:
            self.odometer_reading = mileage
        else: print("You can't roll back an odometer!")

    def increment_odometer(self, miles):
        self.odometer_reading += miles

    def charge(self): print("The car is charging
        ....")


class Battery():
    """ Descripcion de una bateria""" def __init__
    (self,battery_size = 70):
        """Incializacion de los atributos de una bateria""" self.battery_size = battery_size

    def describe_battery(self):
        """Imprime las caracteristicas de la bateria""" print("This car has a " +
        str(self.battery_size)+"-KWh battery.")


class ElectricCar(Car):
    """Un intento de representar un auto electrico""" def
    __init__(self,make,model,year):
```

```
        """ Inicializa los atributos de la clase padre """
        super().__init__(make,model,year) self.battery = Battery()

    def describe_battery(self):
        """ Imprime la el tamanio de la bateria """ print("This car has a " +
        str(self.battery_size) + "-kWh battery.")

    def charge(self): print("The battery is almost empty, the car is charging")

my_tesla = ElectricCar('tesla','model s',2020) print(my_tesla.get_descriptive_name())
my_tesla.battery.describe_battery() my_tesla.charge()
```

---

5 Tarea