

# 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 métodos. A continuación se propone una clase que servirá de base para los ejemplos que se mostrarán 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 características 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`.

### 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 características en la pantalla"""
        long_name = str(self.year)+' '+self.make + ' '+self.model
        return long_name.title()

    def read_odometer(self):
        """Imprime los kilometros recorrido por el auto"""
        print("This car has " + str(self.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.
- 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. 
- Agregue dos atributos predefinidos a la clase `Car`
- Agrugue 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 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 características en la pantalla"""
        long_name = str(self.year)+' '+self.make + ' '+self.model
        return long_name.title()


    def read_odometer(self):
        """Imprime los kilómetros recorrido por el auto"""
        print("This car has " + str(self.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 
- 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

## 1.4 Modificando los valores através de los métodos

De manera general el usuario no debería de tener acceso a los atributos de una clase, ya que los puede modificar accidentalmente y con ello modificaría el flujo del programa y consecuentemente la ejecución del mismo obteniendo resultados erróneos. Para evitar que el usuario tenga de acceso de manera directa a los atributos, se generan métodos para modificar el atributo.

---

```

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

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

```

```

        return long_name.title()

def read_odometer(self):
    """Imprime los kilometros recorrido por el auto"""
    print("This car has " + str(self.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 
- 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

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 características en la pantalla"""
        long_name = str(self.year)+' '+self.make + ' '+self.model
        return long_name.title()

    def read_odometer(self):
        """Imprime los kilometros recorrido por el auto"""
        print("This car has " + str(self.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** 

## 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 **number\_served** y vuelva a imprimir el número de cliente que se han atendido. (Tarea\_1\_1)

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)

- 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\_login\_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

## 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 versión especializada de otra clase, entonces se puede utilizar la *herencia*. Cuando una clase hereda de otra, entonces, automáticamente copia los atributos y métodos de la primera clase. La clase original se le conoce como **Padre** mientras que la nueva clase se le conoce como **hija**. La clase hija hereda todos los atributos de la clase padre, pero también es libre de definir nuevos atributos y métodos para sí 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 quedarían 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)
- 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)

## 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 tamaño de la batería """
        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 código anterior y ejecutarlo, analizar la salida en la terminal, (Ejercicio\_7\_1.py)
- 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)

## 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 tamaño de la batería """
        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 código anterior y ejecutarlo, analizar la salida en la terminal, (Ejercicio\_8\_1.py)
- Del Ejercicio\_8\_2.py hacer la sobrecarga de la función `charging` de las clases `HybridCar` y `FuelCar`. (Ejercicio\_8\_2.py)

## 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 tamaño 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