Clase 14 – POO I

IIC1103 Sección 9 – 2019-2

Profesor: Felipe López

05-11-2019

- ¿Cómo almacenamos información en Python? Mediante **variables**.

- ¿Cómo almacenamos información en Python? Mediante **variables**.
- El resultado de cualquier tipo de operación entre distintos tipos de variables se puede almacenar en otra variable.

- ¿Cómo almacenamos información en Python? Mediante **variables**.
- El resultado de cualquier tipo de operación entre distintos tipos de variables se puede almacenar en otra variable.
- **¿Y las funciones?** Permiten evitar repetir un código varias veces.

- ¿Cómo almacenamos información en Python? Mediante **variables**.
- El resultado de cualquier tipo de operación entre distintos tipos de variables se puede almacenar en otra variable.
- **¿Y las funciones?** Permiten evitar repetir un código varias veces.
- … pero también la interacción entre variables.

- Por ejemplo:

```
import math

def distancia_entre_dos_puntos(x1,x2,y1,y2):
        return math.sqrt((y2-y1)**2+(x2-x1)**2)

x1 = 3
x2 = 6

y1 = -1
y2 = 3

print(distancia_entre_dos_puntos(x1,x2,y1,y2))
```

- No obstante, surgen algunas dudas:
  - ¿Y si tenemos una variable de nombre `distancia_maxima` que queremos ir actualizando con la distancia máxima entre dos pares de puntos en un plano?

```python
import math

def distancia_entre_dos_puntos2(x1,x2,y1,y2):
  distancia_maxima = math.sqrt((y2-y1)**2+(x2-x1)**2)
  return distancia_maxima

distancia_maxima = 0
x1 = 3
x2 = 6

y1 = -1
y2 = 3

print(distancia_entre_dos_puntos2(x1,x2,y1,y2))
print(distancia_maxima)
```

```
5.0
5.0
0
```

- No obstante, surgen algunas dudas:
  - ¿Y si tenemos una variable de nombre `distancia_maxima` que queremos ir actualizando con la distancia máxima entre dos pares de puntos en un plano?

- No obstante, surgen algunas dudas:
  - ¿Y si tenemos una variable de nombre `distancia_maxima` que queremos ir actualizando con la distancia máxima entre dos pares de puntos en un plano?
  - ¿Y si tenemos 1 millón de puntos en el plano?
    - Podríamos almacenarlos en una lista, pero no es tan práctico.

- No obstante, surgen algunas dudas:
  - ¿Y si tenemos una variable de nombre `distancia_maxima` que queremos ir actualizando con la distancia máxima entre dos pares de puntos en un plano?
  - ¿Y si tenemos 1 millón de puntos en el plano?
    - Podríamos almacenarlos en una lista, pero no es tan práctico.
- Nos gustaría crear una variable de tipo “punto”, que tuviera la coordenada x e y.
- También una variable de tipo “plano”, que tuviera una variable `distancia_máxima`.

Estas necesidades, y muchas otras, dieron origen a la **programación orientada a objetos.**

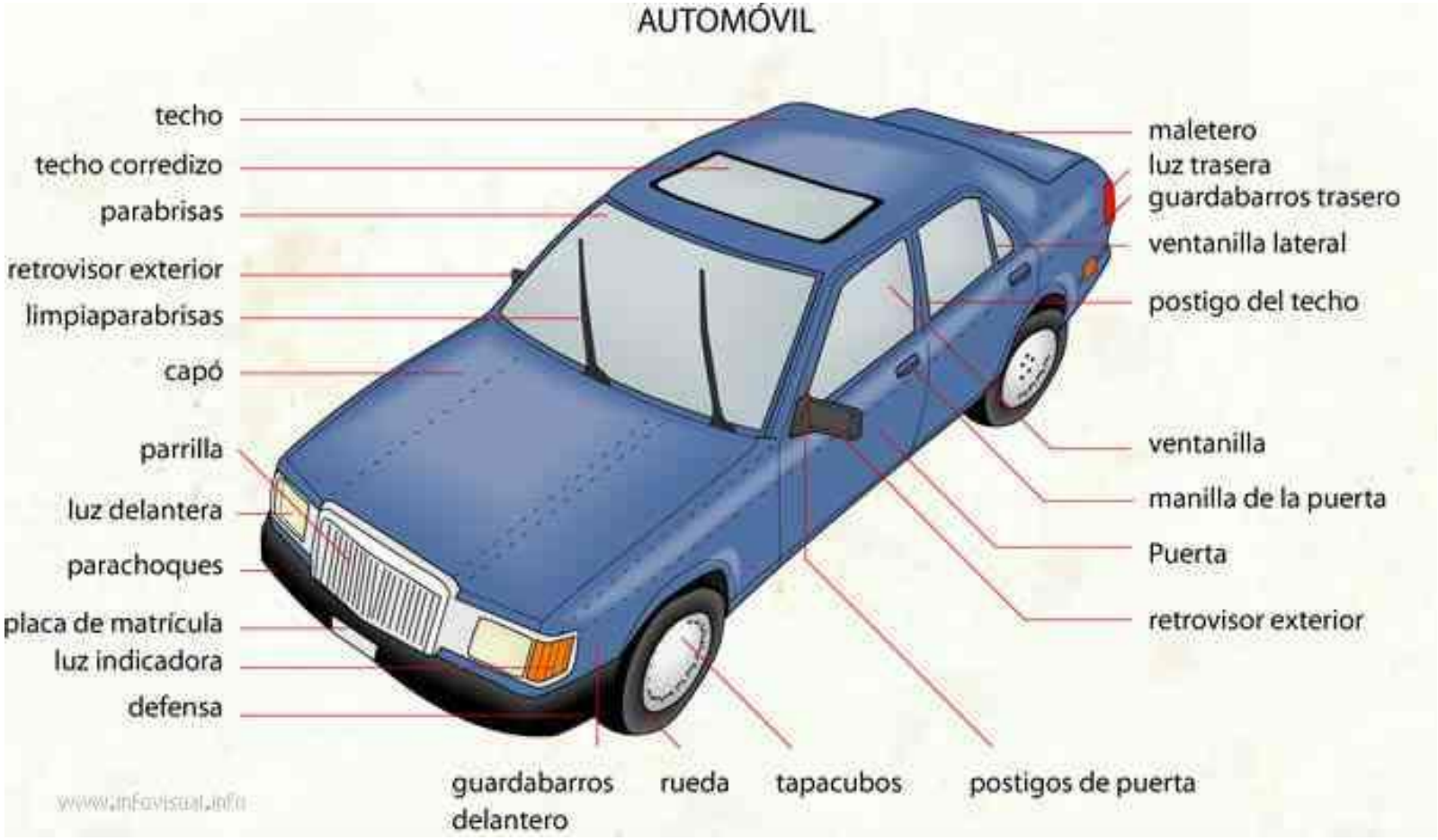
Estas necesidades, y muchas otras, dieron origen a la **programación orientada a objetos.**

¿?

¿Qué es un objeto?

Para eso debemos definir una **clase.**


PSBank


AUTOMÓVIL
techo
techo corredizo
parabrisas
retrovisor exterior
limpiaparabrisas
capó
parrilla
luz delantera
parachoques
placa de matrícula
luz indicadora
defensa
maletero
luz trasera
guardabarros trasero
ventanilla lateral
postigo del techo
ventanilla
manilla de la puerta
Puerta
retrovisor exterior
guardabarros delantero
rueda
tapacubos
postigos de puerta
www.infovisual.info


Marca
Cantidad de puertas
Color
Cantidad de ruedas
Motor


Ciaz


DB11

¿Marca?

¿Motor?

¿Cantidad de puertas?

¿Color?

¿Cantidad de ruedas?

Clases y objetos

- ¿Cómo podemos abstraer nuestra representación de un Auto?
  - En base a los atributos anteriormente definidos.
- Del ejemplo anterior, podemos definir la **clase** “Auto” con los siguiente atributos:
  - Marca
  - Motor
  - Cantidad de puertas
  - Color
  - Cantidad de ruedas

Clases y objetos

- Cada instancia de la clase “Auto” va a representar uno de estos autos:

Clases y objetos

- Estas instancias las denominamos **objetos**. Cada objeto tendrá atributos específicos según su definición.

Clase “Auto”:
- Atributos:
  - Marca
  - Motor
  - Cant. puertas
  - Cant. ruedas
  - Color

Clase “Auto”:
- Atributos:
  - Marca: Mahindra
  - Motor: 1
  - Cant. puertas: 0
  - Cant. ruedas: 3
  - Color: Amarillo

Clase “Auto”:
- Atributos:
  - Marca: Suzuki
  - Motor: 1
  - Cant. puertas: 4
  - Cant. ruedas: 4
  - Color: Negro

Clase “Auto”:
- Atributos:
  - Marca: Aston Martin
  - Motor: 1
  - Cant. puertas: 2
  - Cant. ruedas: 4
  - Color: Naranjo

Clase “Auto”:
- Atributos:
  - Marca
  - Motor
  - Cant. puertas
  - Cant. ruedas
  - Color

Clase “Auto”:
- Atributos:
  - Marca: Mahindra
  - Motor: 1
  - Cant. puertas: 0
  - Cant. ruedas: 3
  - Color: Amarillo

Objeto

Clase “Auto”:
- Atributos:
  - Marca: Suzuki
  - Motor: 1
  - Cant. puertas: 4
  - Cant. ruedas: 4
  - Color: Negro

Objeto

Clase “Auto”:
- Atributos:
  - Marca: Aston Martin
  - Motor: 1
  - Cant. puertas: 2
  - Cant. ruedas: 4
  - Color: Naranjo

Objeto

Desafío

Busquen una cosa que tienen actualmente encima suyo (ropa, *gadgets*, lápices, cuadernos, etc.) y definan una clase en base a ciertos atributos.

Clases y objetos en Python

¿Cómo podemos representar la clase Auto en Python?

```
class Auto:
        def __init__(self,marca,motor,cant_puertas,cant_ruedas,color):
                self.marca = marca
                self.motor = motor
                self.cant_puertas = cant_puertas
                self.cant_ruedas = cant_ruedas
                self.color = color
```

Clases y objetos en Python

```
class Auto:
        def __init__(self,marca,motor,cant_puertas,cant_r
                self.marca = marca
                self.motor = motor
                self.cant_puertas = cant_puertas
                self.cant_ruedas = cant_ruedas
                self.color = color
```

Esta función dentro de la clase “Auto” se denomina **constructor**. Este sirve para poder definir los atributos de una clase.

Clases y objetos en Python

Para crear una clase de forma genérica:

```
class (Nombre clase):
    def __init__(self,…):
        atributo1
        atributo2
        …
```

Clases y objetos en Python

```
class Auto:
        def __init__(self,marca,motor,cant_puertas,cant_ruedas,color):
                self.marca = marca
                self.motor = motor
                self.cant_puertas = cant_puertas
                self.cant_ruedas = cant_ruedas
                self.color = color
```

Esta función dentro de la clase “Auto” se denomina **constructor**. Este sirve para poder definir los atributos de una clase.

¿Y si queremos crear un objeto de la clase Auto?

Clases y objetos en Python

```
class Auto:
        def __init__(self,marca,motor,cant_puertas,cant_ruedas,color):
                self.marca = marca
                self.motor = motor
                self.cant_puertas = cant_puertas
                self.cant_ruedas = cant_ruedas
                self.color = color

rickshaw = Auto("Mahindra",1,0,3,"Amarillo")
```

Clases y objetos en Python

```
class Auto:
        def __init__(self,marca,motor,cant_puertas,cant_ruedas,color):
                self.marca = marca
                self.motor = motor
                self.cant_puertas = cant_puertas
                self.cant_ruedas = cant_ruedas
                self.color = color

rickshaw = Auto("Mahindra",1,0,3,"Amarillo")
```

En este caso la variable `rickshaw` representa un objeto de la clase Auto con los siguientes atributos:

- Marca: Mahindra
- Motor: 1
- Cant. puertas: 0
- Cant. ruedas: 3
- Color: Amarillo

Clases y objetos en Python

```
class Auto:
        def __init__(self,marca,motor,cant_puertas,cant_ruedas,color):
                self.marca = marca
                self.motor = motor
                self.cant_puertas = cant_puertas
                self.cant_ruedas = cant_ruedas
                self.color = color

rickshaw = Auto("Mahindra",1,0,3,"Amarillo")
citycar = Auto("Suzuki",1,4,4,"Negro")
deportivo = Auto("Alfa Romeo",1,3,4,"Naranjo")

print(rickshaw.marca)
print(citycar.cant_puertas)
print(deportivo.color)
```

```
Mahindra
4
Naranjo
```

Clases y objetos en Python

```
class Auto:
        def __init__(self,marca,motor,cant_puertas,cant_ruedas,color):
                self.marca = marca
                self.motor = motor
                self.cant_puertas = cant_puertas
                self.cant_ruedas = cant_ruedas
                self.color = color

rickshaw = Auto("Mahindra",1,0,3,"Amarillo")
citycar = Auto("Suzuki",1,4,4,"Negro")
deportivo = Auto("Alfa Romeo",1,3,4,"Naranjo")

print(rickshaw.marca)
print(citycar.cant_puertas)
print(deportivo.color)
```

```
Mahindra
4
Naranjo
```

Podemos acceder a los atributos de un objeto mediante la notación: `variable_objeto.atributo`

Clases y objetos en Python

```
class Auto:
        def __init__(self,marca,motor,cant_puertas,cant_ruedas,color):
                self.marca = marca
                self.motor = motor
                self.cant_puertas = cant_puertas
                self.cant_ruedas = cant_ruedas
                self.color = color

rickshaw = Auto("Mahindra",1,0,3,"Amarillo")
citycar = Auto("Suzuki",1,4,4,"Negro")
deportivo = Auto("Alfa Romeo",1,3,4,"Naranjo")

print(rickshaw.marca)
print(citycar.cant_puertas)
print(deportivo.color)

deportivo.color = "Blanco"
print(deportivo.color)
```

```
Mahindra
4
Naranjo
Blanco
```

Clases y objetos en Python

```
class Auto:
        def __init__(self,marca,motor,cant_puertas,cant_ruedas,color):
                self.marca = marca
                self.motor = motor
                self.cant_puertas = cant_puertas
                self.cant_ruedas = cant_ruedas
                self.color = color

rickshaw = Auto("Mahindra",1,0,3,"Amarillo")
citycar = Auto("Suzuki",1,4,4,"Negro")
deportivo = Auto("Alfa Romeo",1,3,4,"Naranjo")

print(rickshaw.marca)
print(citycar.cant_puertas)
print(deportivo.color)

deportivo.color = "Blanco"
print(deportivo.color)
```

```
Mahindra
4
Naranjo
Blanco
```

También podemos modificar atributos ya definidos en un objeto.

Clases y objetos en Python

- Modifiquemos un poco nuestro ejemplo. Podemos notar que todos los autos que creamos tienen 1 motor, por lo tanto no es necesario definir este atributo al crear cada objeto.
- Además, ahora queremos añadir el atributo `encendido`, que indica si el motor está encendido o no (con `True` o `False`).

Clases y objetos en Python

```
class Auto:
  def __init__(self,marca,cant_puertas,cant_ruedas,color):
    self.marca = marca
    self.motor = 1
    self.encendido = True
    self.cant_puertas = cant_puertas
    self.cant_ruedas = cant_ruedas
    self.color = color


rickshaw = Auto("Mahindra",0,3,"Amarillo")

print(rickshaw.marca)
print(rickshaw.motor)
```

Clases y objetos en Python

```
class Auto:
  def __init__(self,marca,cant_puertas,cant_ruedas,color):
    self.marca = marca
    self.motor = 1
    self.encendido = True
    self.cant_puertas = cant_puertas
    self.cant_ruedas = cant_ruedas
    self.color = color


rickshaw = Auto("Mahindra",0,3,"Amarillo")

print(rickshaw.marca)
print(rickshaw.motor)
```

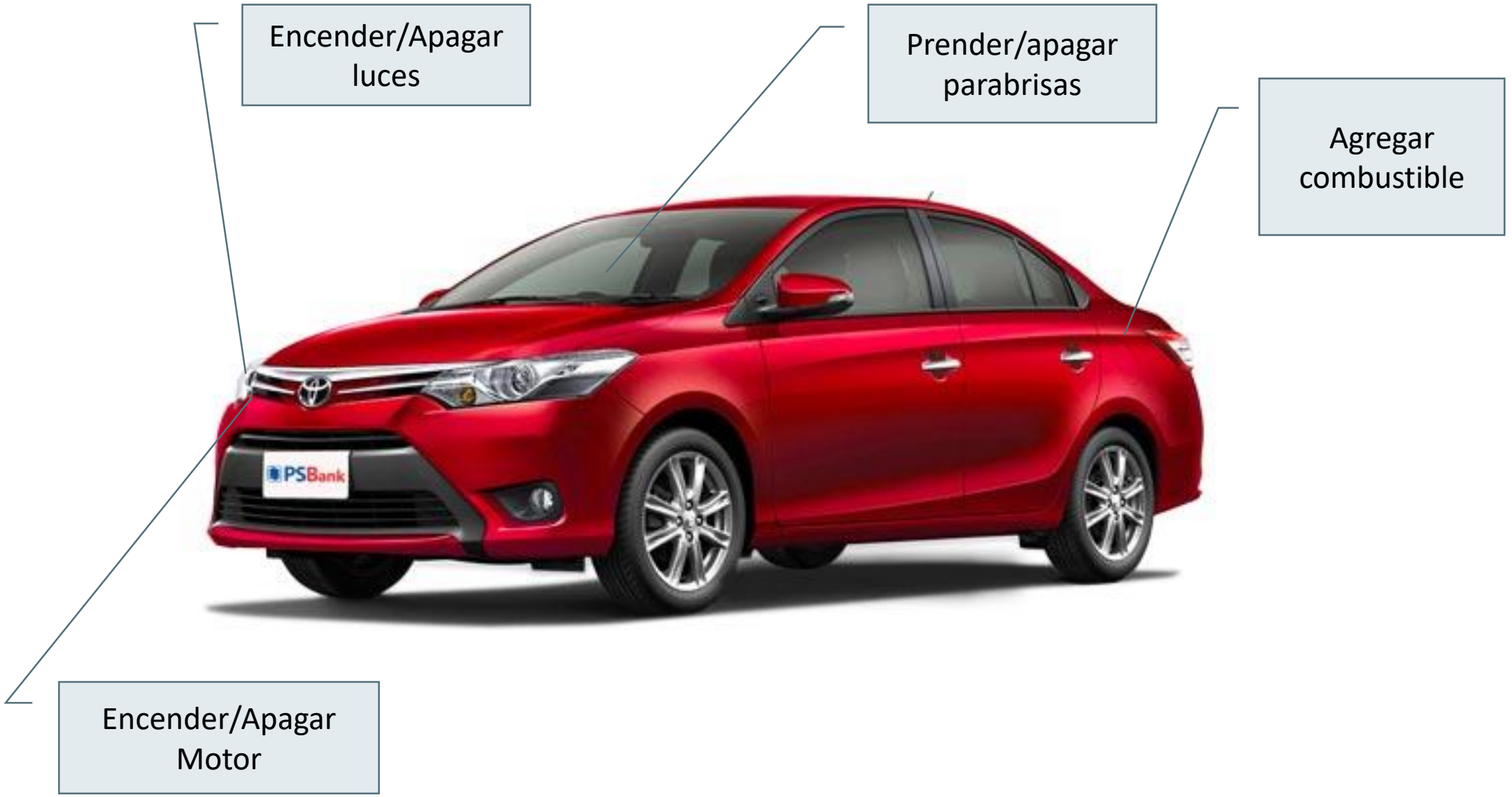
Podemos notar que hay atributos de casi todos los tipos de datos de Python (int, string, bool). Asimismo, podríamos también crear un atributo “estanque” como float, para almacenar la cantidad de litros de bencina que tiene el auto.

break

5 minutos

Métodos

- En el ejemplo anterior, si quisiéramos “encender” el auto, podríamos modificar directamente el atributo `encendido`.
- No obstante, esto supone ciertos riesgos ¿Qué pasa si al asignar True o False a este atributo nos equivocamos en nuestro algoritmo y asignamos otro valor?
- Para esto existen los **métodos.** Estas son funcionalidades que tienen las clases las que permiten interactuar con los atributos bajo ciertas condiciones, y también interactuar con otros objetos.


Encender/Apagar luces
Prender/apagar parabrisas
Agregar combustible
Encender/Apagar Motor
PSBank

Métodos

- La clase Auto puede tener muchos métodos.
- Crearemos dos, que serán `encender_motor()` y `apagar_motor()`.
- Además, crearemos un tercer método que nos indique el estado del motor, de acuerdo al valor del atributo `encendido`.

```python
class Auto:
  def __init__(self,marca,cant_puertas,cant_ruedas,color):
    self.marca = marca
    self.motor = 1
    self.encendido = False
    self.cant_puertas = cant_puertas
    self.cant_ruedas = cant_ruedas
    self.color = color

  def encender_motor(self):
    self.encendido = True

  def apagar_motor(self):
    self.encendido = False

  def estado_motor(self):
    if self.encendido:
      return "El motor está encendido"
    else:
      return "El motor está apagado"
```

```python
rickshaw = Auto("Mahindra",0,3,"Amarillo")

print(rickshaw.estado_motor())
rickshaw.encender_motor()
print(rickshaw.estado_motor())
rickshaw.apagar_motor()
print(rickshaw.estado_motor())
```

```
El motor está apagado
El motor está encendido
El motor está apagado
```

Imprimir

- Digamos que ya creamos el objeto “rickshaw” de la clase Auto.
- ¿Qué ocurre si lo imprimimos?

```
#ejemplo imprimir
rickshaw = Auto("Mahindra",0,3,"Amarillo")

print(rickshaw)
```

```
<__main__.Auto object at 0x7fc9341a1fd0>
```

Imprimir

- Al imprimir el objeto “rickshaw” de la clase Auto, nos gustaría saber información más útil que el espacio de memoria donde está guardado.
- Por ejemplo: la marca, cantidad de puertas, cantidad de puertas, y color.
- Podemos crear un método imprimir que lo haga:

```
#ejemplo 2 clase Auto con método imprimir.
class Auto:
  def __init__(self,marca,cant_puertas,cant_ruedas,color):

 (...)

  def imprimir(self):
    print("Marca:",self.marca,"Cant. Puertas:",self.cant_puer-
tas,"Cant. Ruedas:",self.cant_ruedas,"Color:",self.color)

rickshaw = Auto("Mahindra",0,3,"Amarillo")

rickshaw.imprimir()
```

```
Marca: Mahindra Cant. Puertas: 0 Cant. Ruedas: 3 Color: Ama-
rillo
```

Imprimir

- O de forma alternativa:

```
#ejemplo 3 clase Auto con método imprimir.
class Auto:
  def __init__(self,marca,cant_puertas,cant_ruedas,color):

 (...)

  def imprimir(self):
    return "Marca: "+self.marca+" Cant. Puertas:"+str(self.cant_puertas)+"Cant. Rue-
das: "+str(self.cant_ruedas)+" Color: "+self.color

rickshaw = Auto("Mahindra",0,3,"Amarillo")

print(rickshaw.imprimir())
```

```
Marca: Mahindra Cant. Puertas: 0 Cant. Ruedas: 3 Color: Amarillo
```

Método __str(self)__

- No obstante, aún no resolvemos el problema de poder imprimir directamente la variable que contiene al objeto “rickshaw”, sin necesidad de llamar a un método.
- Podemos hacerlo mediante el método __str(self)__

```
#ejemplo 1 str
class Auto:
  def __init__(self,marca,cant_puertas,cant_ruedas,color):

 (...)

  def __str__(self):
    return "Marca: "+self.marca+" Cant. Puertas: "+str(self.cant_puertas)+" Cant.
Ruedas: "+str(self.cant_ruedas)+" Color: "+self.color

rickshaw = Auto("Mahindra",0,3,"Amarillo")

print(rickshaw)
```

```
Marca: Mahindra Cant. Puertas: 0 Cant. Ruedas: 3 Color: Amarillo
```

`__str(self)__`

- Es posible "sobrecargar"[1] el método (str) de un objeto para que retorne lo que nosotros queramos.
- En este caso, definimos el siguiente método de una clase:

```
def __str__(self):
    …
    return (string)
```

- Esto permitirá poder ocupar `print()` sobre un objeto de esa clase y obtener lo que retorne este método. También permitirá obtener información cada vez que hagamos `str()` a este objeto.

[1] dar una nueva funcionalidad a un método ya existente

Resumen

- Las clase en son representaciones abstractas en Python.
- Una instancia de una clase con atributos específicos es un objeto.
- Se puede interactuar con una clase mediante sus métodos. Estos también permiten la interacción entre distintas clases.

- En Python se definen de la siguiente manera:

```
class Nombre_clase:
  def __init__(self,…):
        self.atributo1
        self.atributo2
                …
  def metodo1(self,…):

  def metodo2(self,…):
```

Bibliografía

- http://runest.ing.puc.cl/class.html
- A. B. Downey. Think Python: How to think like a computer scientist. Green Tea Press, 2013 -> Capítulo 15

Links

- https://repl.it/@FelipeLopez/IIC1103POOI que contiene todos los ejemplos de la clase.

Clase 14 – POO I

IIC1103 Sección 9 – 2019-2

Profesor: Felipe López

05-11-2019