Clase 13 - Programación orientada a objetos II

¿Qué era la POO?

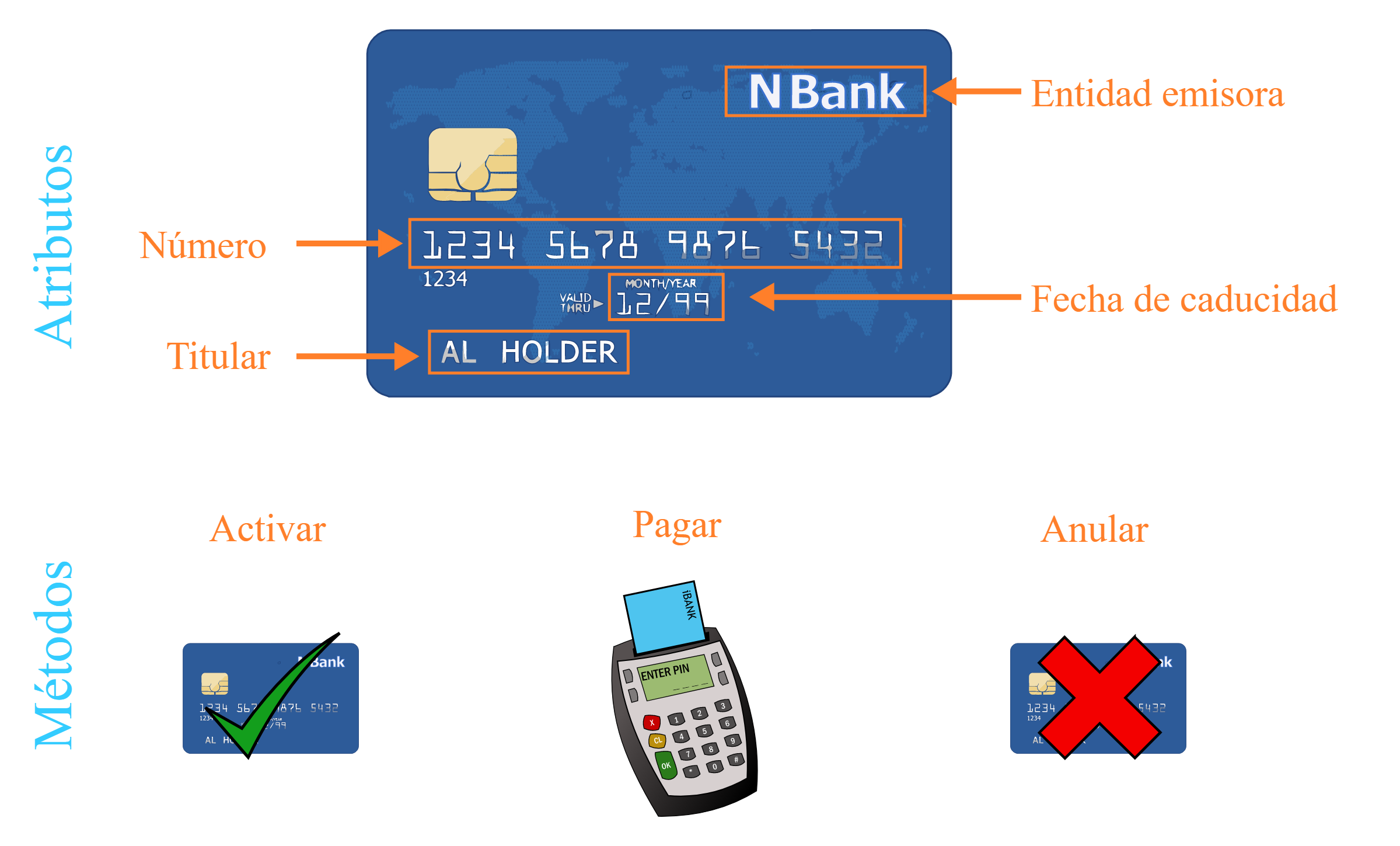
Es un modo o paradigma de programación, que nos permite organizar el código pensando el problema como una relación entre “cosas”, denominadas objetos.

Los objetos se trabajan utilizando las clases. Estas nos permiten agrupar un conjunto de variables y funciones que veremos a continuación. 👉

Usar para slides de sólo texto. Si no alcanza, no sobrecargar, usar otra con el mismo título para indicar que continúa el mismo módulo.

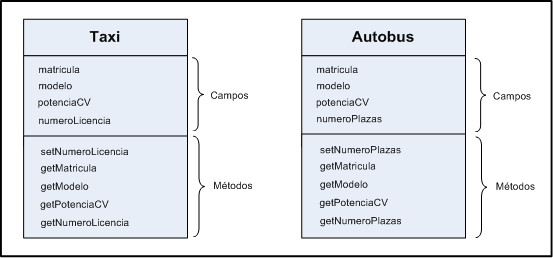
Atributos

Cosas de lo más cotidianas como una computadora o un coche pueden ser representadas con clases. Estas clases tienen diferentes características, que en el caso de la computadora podrían ser la marca o modelo. Llamaremos a estas características, atributos.



Métodos

Por otro lado, las clases tienen un conjunto de funcionalidades. En el caso de la computadora podría ser imprimir o reproducir. Llamaremos a estas funcionalidades métodos.



Objetos

Por último, pueden existir diferentes tipos de computadoras. Llamaremos a estos diferentes tipos de computadoras objetos.

Es decir, el concepto abstracto de computadora es la clase, pero cualquier tipo de computadora particular será el objeto.

¿Para qué usar la POO?

En el mundo de la programación hay gran cantidad de aplicaciones que realizan tareas muy similares y es importante identificar los patrones que nos permiten no reinventar la rueda. La programación orientada a objetos intentaba resolver esto.

Python está diseñado para soportar el paradigma de la programación orientado a objetos (POO), donde todos los módulos se tratan como entidades que tienen atributos, y comportamiento. Lo cual, en muchos casos, ayuda a crear programas de una manera sencilla, además de que los componentes son reutilizables.

Clases

Uso de POO: clases

Lo primero es crear una clase, a la cual llamaremos “perro”. Se trata de una clase vacía y sin mucha utilidad práctica, pero es la mínima clase que podemos crear. El uso del PASS realmente no tiene utilidad, pero daría un error si después de los : no tenemos contenido.

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Instanciar una clase

Ahora que tenemos la clase, podemos crear uno o más objetos de la misma, como si de una variable normal se tratase.

Nombre de la variable igual a la clase con ( ). Dentro de los paréntesis irían los parámetros de entrada si los hubiera.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Atributos

Asignando atributos

A la hora de añadir atributos a la clase, es importante distinguir dos tipos:

* Atributos de instancia: Pertenecen a la instancia de la clase o al objeto. Son atributos particulares de cada instancia, en nuestro caso de cada perro.
* Atributos de clase: Se trata de atributos que pertenecen a la clase, por lo tanto serán comunes para todos los objetos.

Asignando atributos de instancia

Empecemos creando el nombre y la raza del perro. Esto se suele hacer por medio de un método \_\_init\_\_ que será llamado automáticamente cuando se cree un objeto. No se le puede cambiar el nombre y tiene su definición propia, “Constructor”.

Texto

Descripción generada automáticamente

¿Qué es el self?

Es una variable que representa la instancia de la clase, y deberá estar siempre ahí.

El uso de \_\_init\_\_ y el doble \_\_ no es una coincidencia. Cuando veas un método con esa forma, significa que está reservado para un uso especial del lenguaje. En este caso sería lo que se conoce como constructor.

Asignando atributos de clase

Hasta ahora hemos definido atributos de instancia, ya que pertenecen a cada perro concreto. Ahora vamos a definir un atributo de clase, que será común para todos los perros.

Por ejemplo, la especie de los perros es algo común para todos los objetos Perro.

Texto

Descripción generada automáticamente

Dado que es un atributo de clase, no es necesario crear un objeto para acceder al atributo.

Se puede acceder también al atributo de clase desde el objeto.



🤩De esta manera, todos los objetos que se creen de la clase perro compartirán ese atributo de clase.

Estructura final de una clase con atributos

Texto

Descripción generada automáticamente

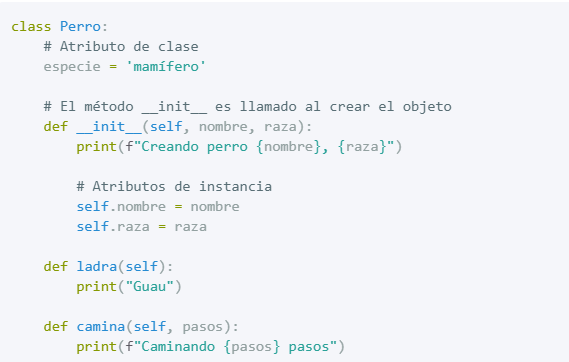
Métodos

Definiendo métodos

En realidad cuando usamos \_\_init\_\_ ya estábamos definiendo un método especial.

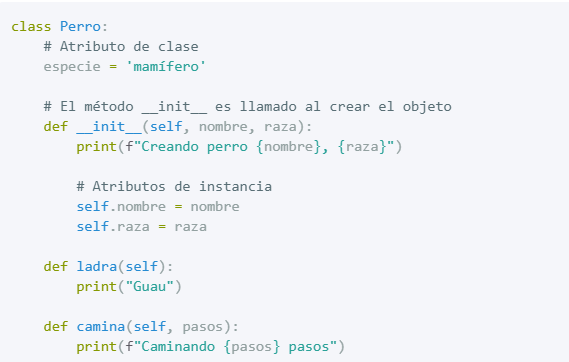
A continuación, vamos a ver cómo definir métodos que le den alguna funcionalidad interesante a nuestra clase, siguiendo con el ejemplo de perro.

Vamos a codificar dos métodos: Ladrar y caminar.

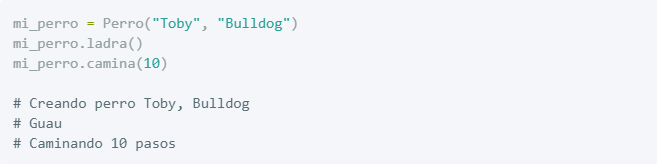


El primero no recibirá ningún parámetro y el segundo recibirá el número de pasos que queremos andar. Como hemos indicado anteriormente, self hace referencia a la instancia de la clase.

Se puede definir un método con def y el nombre, y entre ( ) los parámetros de entrada que recibe, donde siempre tendrá que estar self el primero.



Por lo tanto, si creamos un objeto mi\_perro, podremos hacer uso de sus métodos llamándolos con . y el nombre del método. Como si de una función se tratase, pueden recibir y devolver argumentos.



Nuestro código quedaría de la siguiente manera

Texto

Descripción generada automáticamente

Métodos especiales

¿Qué son?

Los métodos mágicos de las clases Python son aquellos que comienzan y terminan con dos caracteres subrayados. Son de suma importancia ya que muchas de las operaciones que se hacen con clases en Python utilizan estos métodos, como puede ser la creación del objeto (\_\_init\_\_) o generar una cadena que los represente (\_\_str\_\_).

Todos estos métodos tienen un primer parámetro que hace referencia al propio objeto, que por convenio se suele llamar self. Siendo el resto de los parámetros del método variables.

Veamos a continuación algunos de estos métodos y cuál es su finalidad.

Texto

Descripción generada automáticamente

\_init\_

El método mágico más utilizado en las clases de python es el constructor. Empleado para crear cada una de las instancias de las clases. No tiene una cantidad fija de parámetros, ya que estos dependen de las propiedades que tenga el objeto. Por ejemplo, se puede crear un objeto Vector:

class Vector():

class Vector():

def \_\_init\_\_(self, data):

self.\_data = data

v = Vector([1,2])

print(v)

\_str\_

Posiblemente el segundo método más utilizado, con el que se crea una representación del objeto con significado para las personas.

En el ejemplo anterior, al imprimir el objeto, se ha visto una cadena que indica el tipo de objeto y una dirección. Lo que generalmente no tiene sentido para las personas.

Un problema que se puede solucionar con el método \_\_str\_\_.

class Vector():

def \_\_init\_\_(self, data):

self.\_data = data

def \_\_str\_\_(self):

return f"The values are: {self.\_data}"

v = Vector([1,2])

print(v)

\_len\_

Podríamos usar la función len para obtener el número de elementos que tenga el objeto.

Pero si probamos con el objeto nos encontraremos con un error.

class Vector():

def \_\_init\_\_(self, data):

self.\_data = data

def \_\_str\_\_(self):

return f"The values are: {self.\_data}"

def \_\_len\_\_(self):

return len(self.\_data)

v = Vector([1,2])

len(v)

Esto es así porque la clase no ha implementado aún el método \_\_len\_\_. Un problema se puede solucionar de tomar fácil al incluir este método.

class Vector():

def \_\_init\_\_(self, data):

self.\_data = data

def \_\_str\_\_(self):

return f"The values are: {self.\_data}"

def \_\_len\_\_(self):

return len(self.\_data)

v = Vector([1,2])

len(v)

\_getitem\_

Si se desea que los usuarios puedan leer los elementos mediante el uso de corchetes es necesario implementar el método \_\_getitem\_\_ en la clase. Este elemento necesita un parámetro adicional, que es la posición del elemento a leer. La función debe retornar el valor leído.

class Vector():

def \_\_init\_\_(self, data):

self.\_data = data

def \_\_str\_\_(self):

return f"The values are: {self.\_data}"

def \_\_len\_\_(self):

return len(self.\_data)

def \_\_getitem\_\_(self, pos):

return self.\_data[pos]

v = Vector([1,2])

v[1]

El método \_\_getitem\_\_ solo nos sirve para leer, ya que para modificar, se tiene que crear también el método \_\_setitem\_\_, el cual veremos a continuación.

\_setitem\_

Este es el complemento del método anterior. En este caso es necesario pasar dos parámetros adicionales, la posición y el valor a reemplazar.

class Vector():

def \_\_init\_\_(self, data):

self.\_data = data

def \_\_str\_\_(self):

return f"The values are: {self.\_data}"

def \_\_len\_\_(self):

return len(self.\_data)

def \_\_getitem\_\_(self, pos):

return self.\_data[pos]

def \_\_setitem\_\_(self, pos, value):

self.\_data[pos] = value

v = Vector([1,2])

v[1] = 20

print(v)

\_iter\_

Hace que nuestra clase sea iterable, lo que permite usarla por ejemplo en bucles tipo for, es necesario implementar este método. A continuación, podemos ver una implementación de este método en el que devuelve una cadena con el índice y el valor de cada elemento.

class Vector():

def \_\_init\_\_(self, data):

self.\_data = data

def \_\_str\_\_(self):

return f"The values are: {self.\_data}"

def \_\_len\_\_(self):

return len(self.\_data)

def \_\_getitem\_\_(self, pos):

return self.\_data[pos]

def \_\_setitem\_\_(self, pos, value):

self.\_data[pos] = value

def \_\_iter\_\_(self):

for pos in range(0, len(self.\_data)):

yield f"Value[{pos}]: {self.\_data[pos]}"

v = Vector([1,2])

for vec in v:

print(vec)

Cuando necesitamos que nuestra clase sea iterable, lo que permite usarla por ejemplo en bucles tipo for, es necesario implementar este método. A continuación, podemos ver una implementación en el que devuelve una cadena con el índice y el valor de cada elemento.

class Vector():

def \_\_init\_\_(self, data):

self.\_data = data

def \_\_str\_\_(self):

return f"The values are: {self.\_data}"

def \_\_len\_\_(self):

return len(self.\_data)

def \_\_getitem\_\_(self, pos):

return self.\_data[pos]

def \_\_setitem\_\_(self, pos, value):

self.\_data[pos] = value

def \_\_iter\_\_(self):

for pos in range(0, len(self.\_data)):

yield f"Value[{pos}]: {self.\_data[pos]}"

v = Vector([1,2])

for vec in v:

print(vec)

Clases anidadas

Relación entre clases

Cuando vimos el por qué trabajar por medio de este paradigma, dijimos que era fundamental para separar a nuestro problema en problemas más pequeños (Clases), pero estos “problemas” no van a estar aislados, estarán relacionados entre sí. Por ejemplo, los Perros, suelen ser la mascota de una Persona.

Es muy simple. Pero en el próximo slide veremos otra variante de Clases Anidadas.

Texto

Descripción generada automáticamente

Esta es otra forma de realizar clases internas a otras clases. No es lo más típico y usado, pero puede ser útil para cuando hay una relación muy estricta entre dos clases.

Captura de pantalla de computadora

Descripción generada automáticamente

Encapsulamiento

¿De qué se trata?

Es un concepto relacionado con la programación, orientada a objetos, y hace referencia al ocultamiento de los estados internos de una clase al exterior. Dicho de otra manera, encapsular consiste en hacer que los atributos o métodos internos a una clase no se puedan acceder ni modificar desde fuera, sino que tan solo el propio objeto pueda acceder a ellos.

“Python por defecto no oculta los atributos y métodos de una clase al exterior.”

ellibrodepython.com

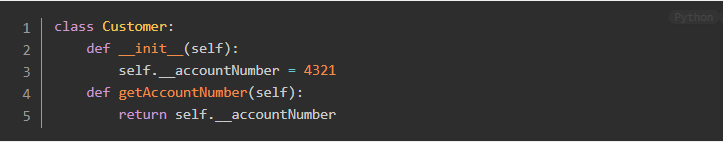
Esto es genial para “aprender”, pero en la práctica no nos gustaría que desde la clase Perro le cambien el nombre a una instancia de la clase Sueldo, por dar un ejemplo. Es por eso que es fundamental “defender” a los datos y tornarlos privados.

Visibilidad de los datos

Efectivamente, todo lo que esté privado ( con doble \_\_ ) será imposible de acceder desde el exterior de la clase

¿Cómo encapsular en Python?

Hasta el momento sabemos que la encapsulación es el ocultamiento de datos del estado interno para proteger la integridad del objeto. En el siguiente ejemplo, CLIENTE es una clase, y hemos encapsulado (al anteponer \_\_ ) la variable accountNumber.



Del mismo modo, podemos ocultar el acceso a los métodos. Para eso, necesitamos anteponer el nombre del método con \_\_ (subrayado doble).

