CONQUER BLOCKS

PYTHON

PROGRAMACION ORIENTEADA A OBJETOS (POO)

MIEMBROS PRIVADOS (private members)

EJEMPLO: Tenemos un coche que vendemos a 60000 pero cuyo coste de producción es de 10000. Esto último no querríamos que el usuario lo sepa.

```python
class Coche:
    def __init__(self, marca, modelo, anio):
        """Inicializamos atributos del coche"""
        self.marca = marca
        self.modelo = modelo
        self.anio = anio
        self.coste_produccion = 10000

mi_coche = Coche("audi", "a4", 2018)
print(mi_coche.coste_produccion)
```

✓ 0.0s

```
10000
```

MIEMBROS PRIVADOS (private members)

EJEMPLO: Tenemos un coche que v
es de 10000. Esto último no querríam

```
class Coche:
    def __init__(self, marca, model
        """Inicializamos atributos
        self.marca = marca
        self.modelo = modelo
        self.anio = anio
        self.coste_produccion = 100

mi_coche = Coche("audi", "a4", 2018
print(mi_coche.coste_produccion)
```

✓ 0.0s

```
10000
```

```
class Coche:
    def __init__(self, marca, modelo, anio):
        """Inicializamos atributos del coche"""
        self.marca = marca
        self.modelo = modelo
        self.anio = anio
        self.__coste_produccion = 10000

mi_coche = Coche("audi", "a4", 2018)
print(mi_coche.__coste_produccion)
```

⊗ 0.1s

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[2], line 10
      7         self.__coste_produccion = 10000
      9 mi_coche = Coche("audi", "a4", 2018)
---> 10 print(mi_coche.__coste_produccion)

AttributeError: 'Coche' object has no attribute '__coste_produccion'
```

MIEMBROS PRIVADOS (private members)

EJEMPLO: Tenemos un coche que v
es de 10000. Esto último no querríam

```
class Coche:
    def __init__(self, marca, model
        """Inicializamos atributos
        self.marca = marca
        self.modelo = modelo
        self.anio = anio
        self.coste_produccion = 100

mi_coche = Coche("audi", "a4", 2018
print(mi_coche.coste_produccion)
```

✓ 0.0s

```
10000
```

```
class Coche:
    def __init__(self, marca, modelo, anio):
        """Inicializamos atributos del coche"""
        self.marca = marca
        self.modelo = modelo
        self.anio = anio
        self.__coste_produccion = 10000

mi_coche = Coche("audi", "a4", 2018)
print(mi_coche.__coste_produccion)
```

Miembro privado

⊗ 0.1s

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[2], line 10
      7         self.__coste_produccion = 10000
      9 mi_coche = Coche("audi", "a4", 2018)
---> 10 print(mi_coche.__coste_produccion)

AttributeError: 'Coche' object has no attribute '__coste_produccion'
```

MIEMBROS PRIVADOS (private members)

En realidad esto no hace el atributo del todo privado…

```python
class Coche:
    def __init__(self, marca, modelo, anio):
        """Inicializamos atributos del coche"""
        self.marca = marca
        self.modelo = modelo
        self.anio = anio
        self.__coste_produccion = 10000

mi_coche = Coche("audi", "a4", 2018)
print(mi_coche.__coste_produccion)
```

Miembro privado

⊗ 0.1s

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[2], line 10
      7         self.__coste_produccion = 10000
      9 mi_coche = Coche("audi", "a4", 2018)
---> 10 print(mi_coche.__coste_produccion)

AttributeError: 'Coche' object has no attribute '__coste_produccion'
```

MIEMBROS PRIVADOS (private members)

En realidad esto no hace el atributo del todo privado…

```
class Coche:
    def __init__(self, marca, modelo, anio):
        """Inicializamos atributos del coche"""
        self.marca = marca
        self.modelo = modelo
        self.anio = anio
        self.__coste_produccion = 10000

mi_coche = Coche("audi", "a4", 2018)
print(mi_coche._Coche__coste_produccion)
```

✓ 0.0s

```
10000
```

MIEMBROS PRIVADOS (private members)

```
class Coche:
    def __init__(self, marca, modelo, anio):
        """Inicializamos atributos del coche"""
        self.marca = marca
        self.modelo = modelo

mi_coche = Coche("audi", "a4", 2018)
print(mi_coche._Coche__coste_produccion)
```

✓ 0.0s

```
10000
```

Lo que llamamos atributos privados en python no tiene que ver con privacidad real

MIEMBROS PRIVADOS (private members)

¿Para qué nos interesan?

1. **Encapsulación:**

   Al marcar un atributo como privado, estás indicando que ese atributo no debe ser accedido directamente desde fuera de la clase.

   Esto permite un mejor control sobre cómo los datos son manipulados y protege la integridad de los datos internos de la clase.

```python
class Coche:
    def __init__(self, marca, modelo, anio):
        """Inicializamos atributos del coche"""
        self.marca = marca
        self.modelo = modelo
        self.anio = anio
        self.__coste_produccion = 10000


class CocheElectrico(Coche):
    """Representa aspectos de un coche,
    especifico para coches electricos"""
    def __init__(self, marca, modelo, anio):
        super().__init__(marca, modelo, anio)
        self.tamanio_bateria = 70
```

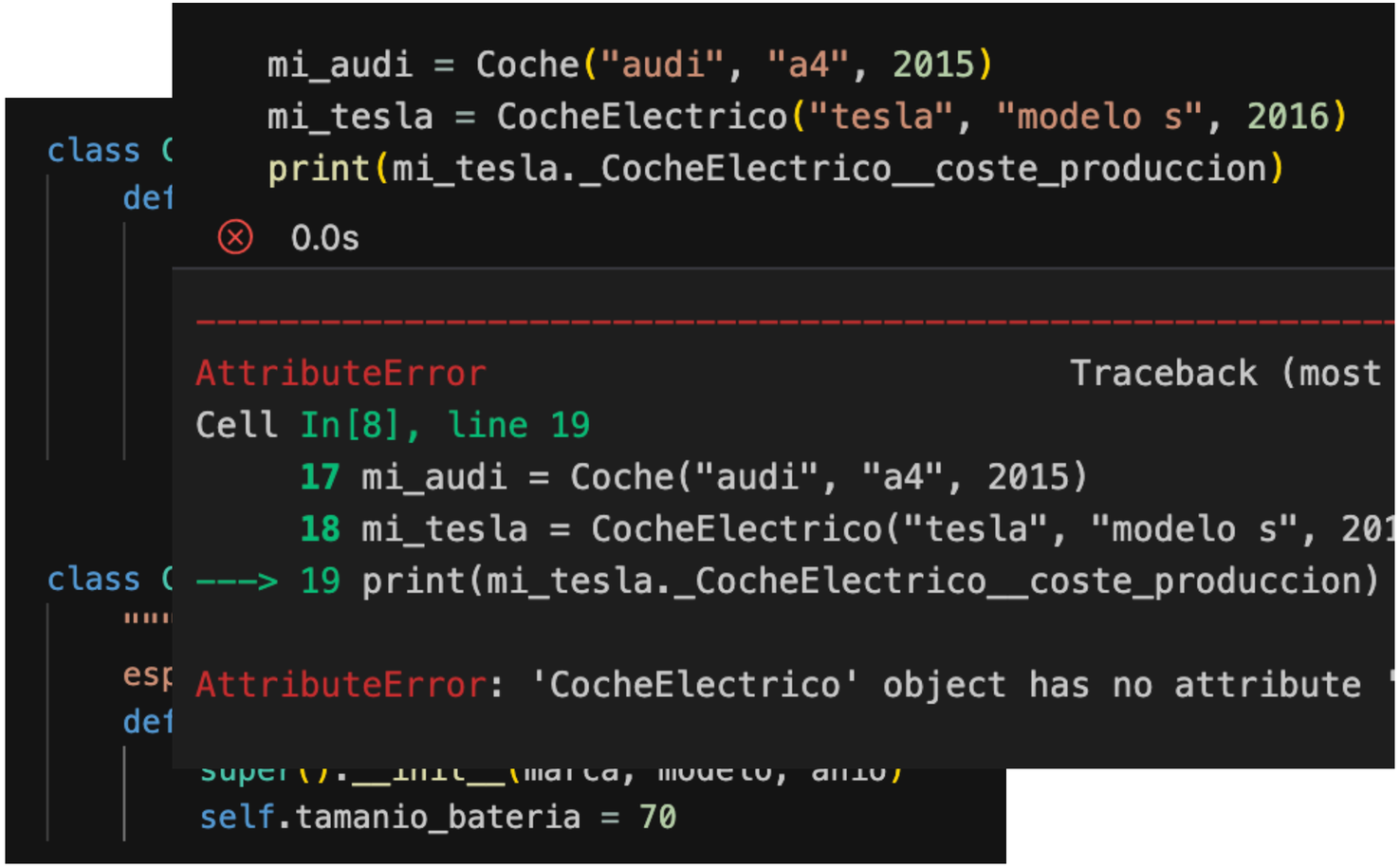
MIEMBROS PRIVADOS (private members)

¿Para qué nos interesan?

1. **Encapsulación:**

   Al marcar un atributo como privado, estás indicando que ese atributo no debe ser accedido directamente desde fuera de la clase.

   Esto permite un mejor control sobre cómo los datos son manipulados y protege la integridad de los datos internos de la clase.



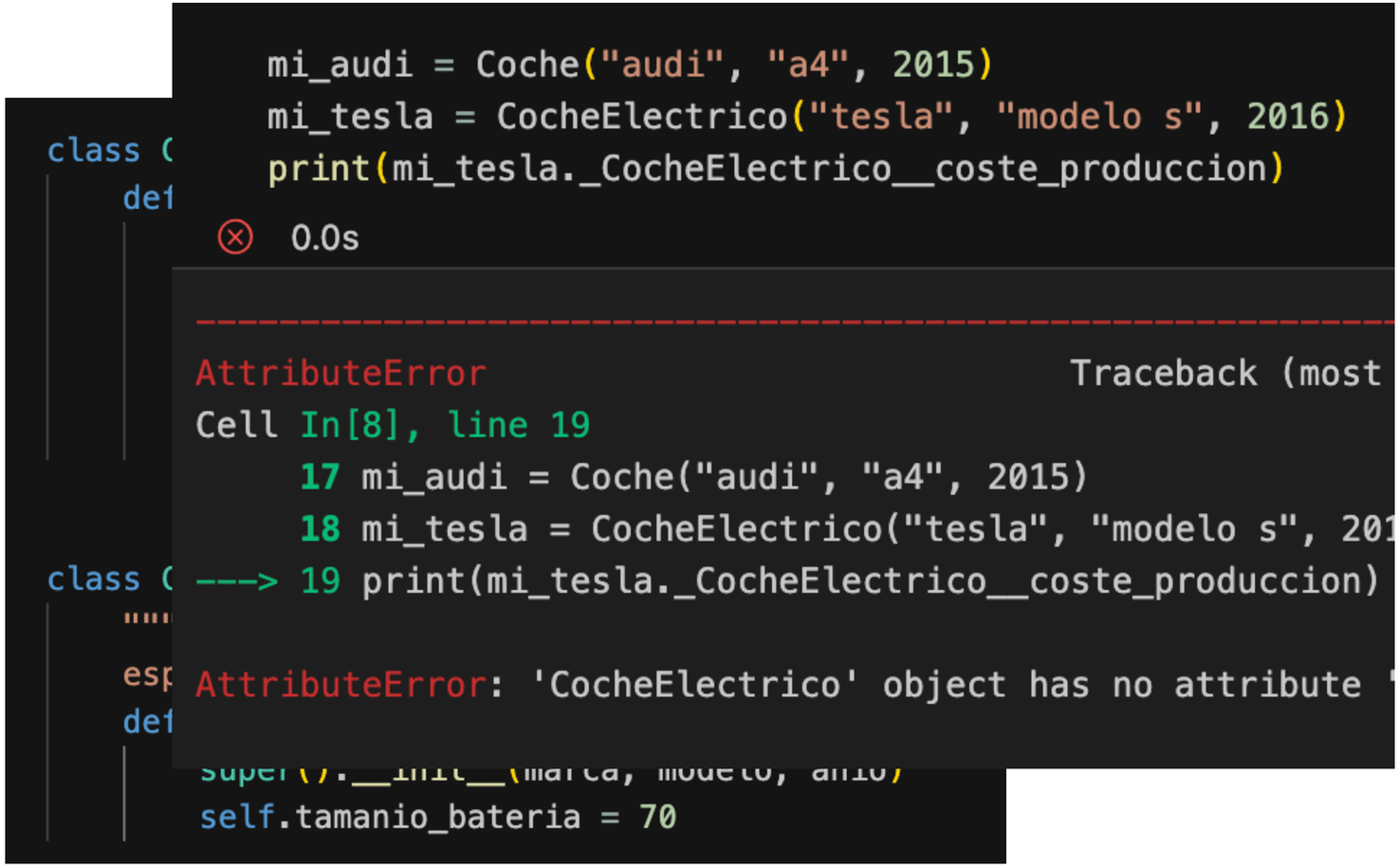
MIEMBROS PRIVADOS (private members)

¿Para qué nos interesan?

2. Control de Acceso:

Los atributos privados restringen el acceso directo a los datos desde fuera de la clase.

Solo los métodos dentro de la misma clase pueden acceder y modificar estos atributos. Esto evita modificaciones accidentales o inapropiadas de los datos.

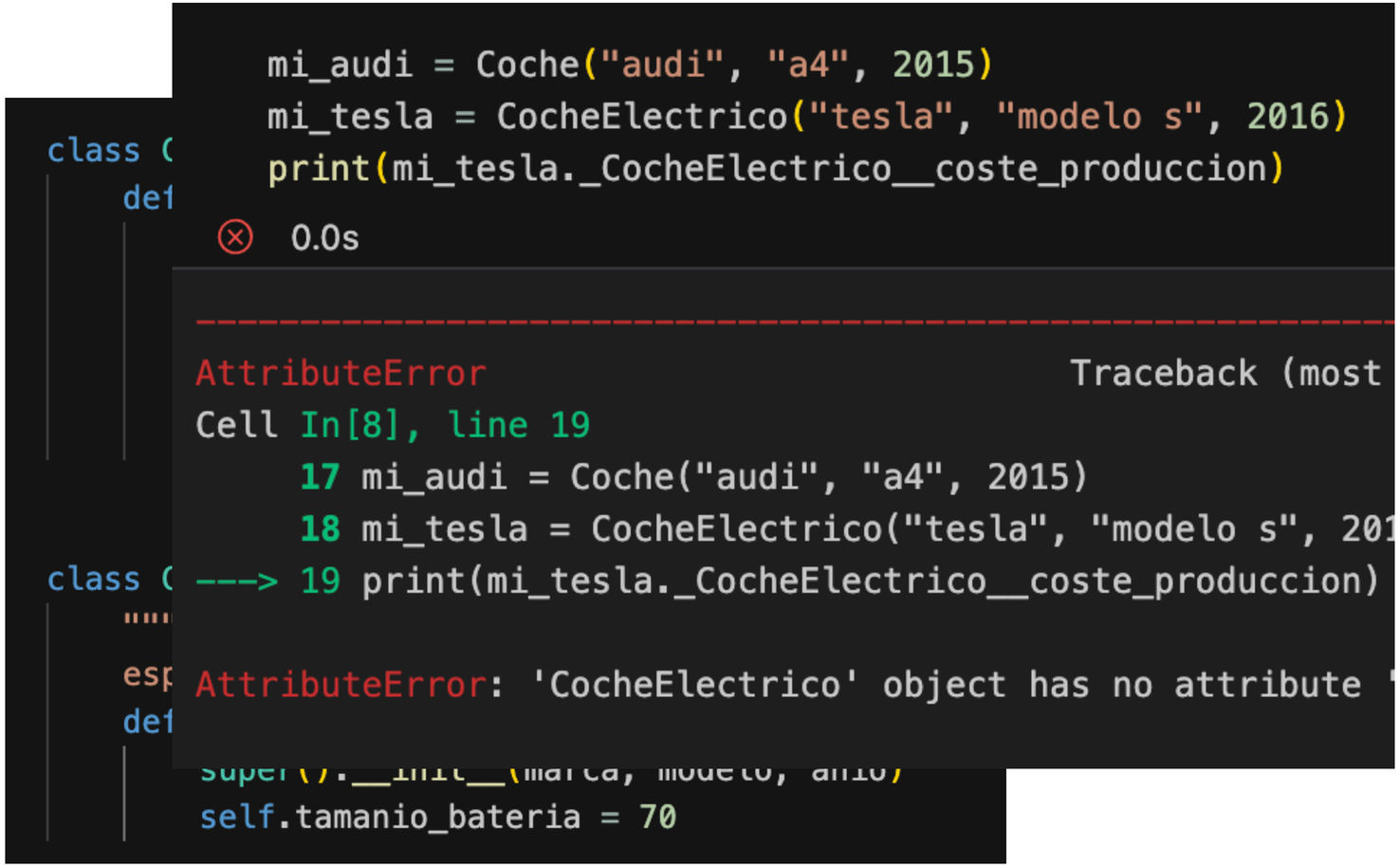


MIEMBROS PRIVADOS (private members)

¿Para qué nos interesan?

3. Evitar Colisiones

Al usar atributos privados, reduces el riesgo de colisiones de nombres con atributos de otras clases o del código externo

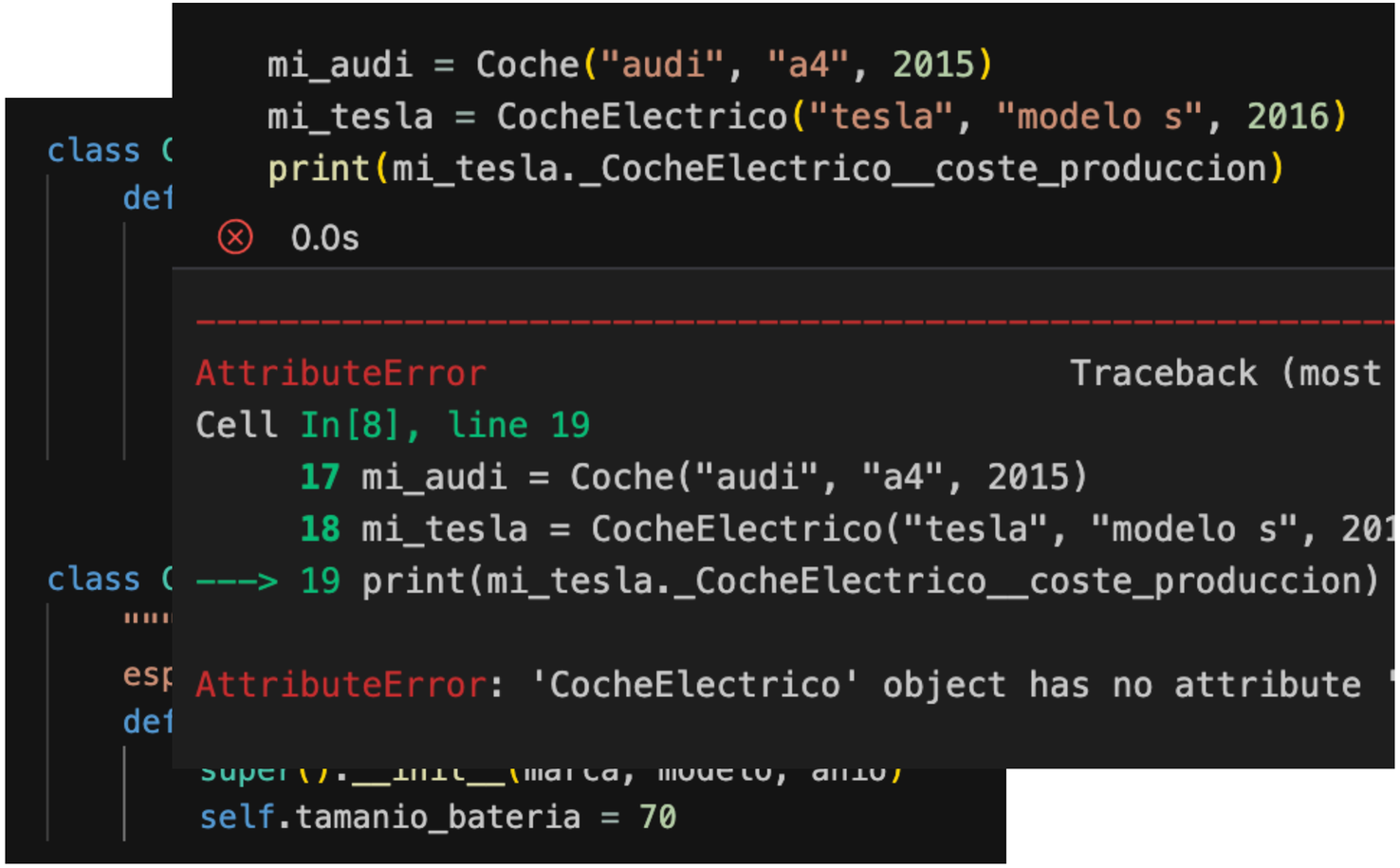


MIEMBROS PRIVADOS (private members)

¿Para qué nos interesan?

4. **Documentación y Abstracción**

   Al marcar los atributos como privados, estás señalando a otros programadores que estos atributos no están destinados a ser accedidos directamente y que deben consultar la documentación de la clase para comprender cómo interactuar adecuadamente con ella.



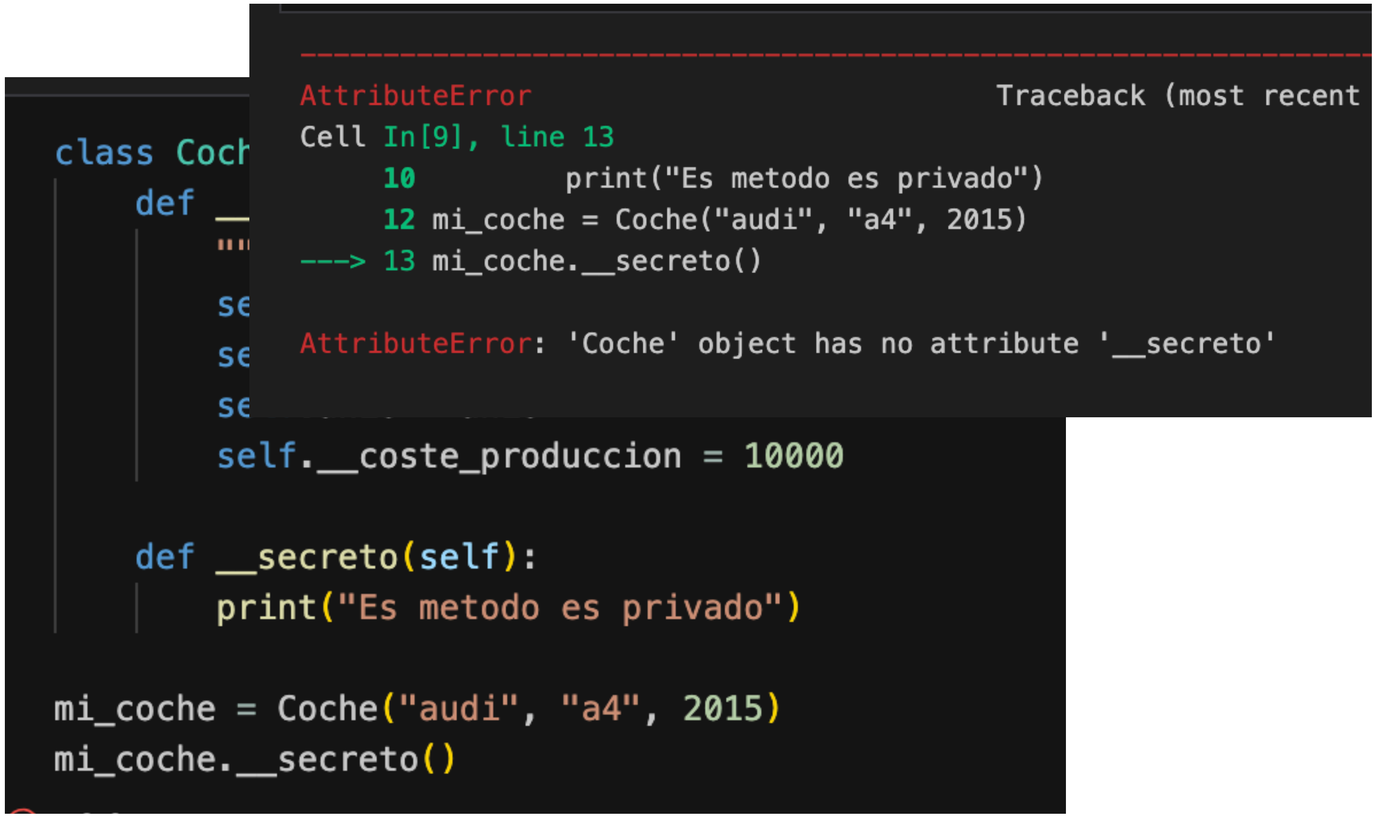
MIEMBROS PRIVADOS (private members)

Podemos aplicar esto mismo a los métodos…

```
class Coche:
    def __init__(self, marca, modelo, anio):
        """Inicializamos atributos del coche"""
        self.marca = marca
        self.modelo = modelo
        self.anio = anio
        self.__coste_produccion = 10000

    def __secreto(self):
        print("Es metodo es privado")

mi_coche = Coche("audi", "a4", 2015)
mi_coche.__secreto()
```

MIEMBROS PRIVADOS (private members)

Podemos aplicar esto mismo a los métodos...



```
class Coch
    def __
        """
        se
        se
        se
        self.__coste_produccion = 10000

    def __secreto(self):
        print("Es metodo es privado")

mi_coche = Coche("audi", "a4", 2015)
mi_coche.__secreto()
```

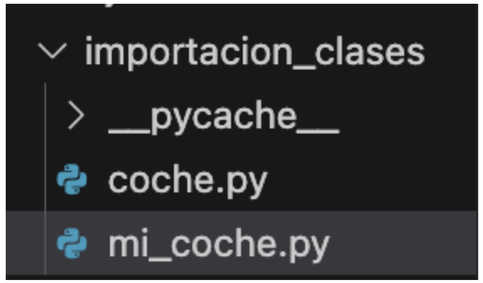
```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent
Cell In[9], line 13
     10         print("Es metodo es privado")
     12 mi_coche = Coche("audi", "a4", 2015)
---> 13 mi_coche.__secreto()

AttributeError: 'Coche' object has no attribute '__secreto'
```

MIEMBROS PRIVADOS (private members)

Podemos aplicar esto mismo a los métodos...

IMPORTACION DE CLASES

IMPORTACION DE UNA CLASE:



coche.py

importacion_clases > coche.py > ...

```python
class Coche:
    def __init__(self, marca, modelo, anio):
        """Inicializamos atributos del coche"""
        self.marca = marca
        self.modelo = modelo
        self.anio = anio
        self.__coste_produccion = 10000

    def llenar_deposito(self):
        """Simula el llenado del deposito del coche"""
        self.deposito = 100
        print("El deposito esta a", self.deposito)

```

mi_coche.py | coche.py | clase9_c.ipynb

importacion_clases > mi_coche.py > anio

```python
from coche import Coche

mi_nuevo_coche = Coche("audi", "a4", 2016)
mi_nuevo_coche.llenar_deposito()

mi_nuevo_coche.anio = 2014
print(mi_nuevo_coche.anio)
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
(cblocks) MacBook-Pro-5:Clase 9 Elena$ /Users/Elena/minic
ython/Avanzado/Clase 9/importacion_clases/mi_coche.py"
El deposito esta a 100
2014
(cblocks) MacBook-Pro-5:Clase 9 Elena$
```

IMPORTACION DE CLASES

GUARDAR MÚLTIPLES CLASES EN UN MÓDULO:

mi_coche_elec.py

importacion_clases > mi_coche_elec.py > ...

```
1  from coche import CocheElectrico
2
3  mi_tesla = CocheElectrico("tesla", "modelo s", 2016)
4
5  print(mi_tesla.bateria.tamanio_bateria)
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL JUPY

```
(cblocks) MacBook-Pro-5:Clase 9 Elena$ /Users/Elena/miniconda3/
ython/Avanzado/Clase 9/importacion_clases/mi_coche_elec.py"
70
(cblocks) MacBook-Pro-5:Clase 9 Elena$
```

```
class Coche:
    def __init__(self, marca, modelo, anio):
        """Inicializamos atributos del coche"""
        self.marca = marca
        self.modelo = modelo
        self.anio = anio
        self.deposito = 0

    def llenar_deposito(self):
        """Simula el llenado del deposito del coche"""
        self.deposito = 100
        print("El deposito esta a", self.deposito)

class Bateria:
    """Un intento simple de modelizar
    una bateria"""

    def __init__(self, tamanio_bateria = 70):
        """Inicializamos los atributos de la bateria"""
        self.tamanio_bateria = tamanio_bateria

    def describir_bateria(self):
        """Imprimir el tamaño de la bateria"""
        print("Este coche tiene una bateria de tamaño",
        self.tamanio_bateria,"-kWh.")

class CocheElectrico(Coche):
    """Representa aspectos de un coche,
    especifico para coches electricos"""
    def __init__(self, marca, modelo, anio):
        """ Inicializar atributos de superclase.
        Despues inicializar atributos especificos
        del coche electrico """
        super().__init__(marca, modelo, anio)
        self.bateria = Bateria()

mi_tesla = CocheElectrico("tesla", "modelo s", 2016)
mi_tesla.llenar_deposito()
mi_tesla.bateria.describir_bateria()
```

IMPORTACION DE CLASES

IMPORTAR UNA PARENT CLASS:

coche.py

importacion_clases > coche.py > ...

```python
class Coche:
    def __init__(self, marca, modelo, anio):
        """Inicializamos atributos del coche"""
        self.marca = marca
        self.modelo = modelo
        self.anio = anio
        self.__coste_produccion = 10000

    def llenar_deposito(self):
        """Simula el llenado del deposito del coche"""
        self.deposito = 100
        print("El deposito esta a", self.deposito)

```

coche_electrico.py

importacion_clases > coche_electrico.py > ...

```python
from coche import Coche

class Bateria:
    """Un intento simple de modelizar
    una bateria"""

    def __init__(self, tamanio_bateria = 70):
        """Inicializamos los atributos de la bateria"""
        self.tamanio_bateria = tamanio_bateria

    def describir_bateria(self):
        """Imprimir el tamaño de la bateria"""
        print("Este coche tiene una bateria de tamaño",
        self.tamanio_bateria,"-kWh.")

class CocheElectrico(Coche):
    """Representa aspectos de un coche,
    especifico para coches electricos"""
    def __init__(self, marca, modelo, anio):
        """ Inicializar atributos de superclase.
        Despues inicializar atributos especificos
        del coche electrico """
        super().__init__(marca, modelo, anio)
        self.bateria = Bateria()
```

GUIA DE ESTILO…

- **Variables:**
  - Letras minúsculas
  - Separamos las palabras con guiones bajos (snake_case)
  - Nombres descriptivos que transmitan el propósito de la variable
- **Funciones:**
  - Letras minúsculas
  - Separamos las palabras con guiones bajos (snake_case)
  - Usa verbos para describir las acciones
- **Constantes:**
  - Letras mayúsculas
  - Separamos las palabras con guiones bajos (SNAKE_CASE)
  - Nombres claros y concisos que transmitan el propósito de la contante
- **Clases:**
  - Usa CamelCase (capitaliza la primera letra de cada palabra sin espacios)
  - Nombres descriptivos que reflejen la naturaleza de la clase

GUIA DE ESTILO...

- Los nombres de instancias y módulos se escriben en snake_case

```
mi_instancia = MiClaseEjemplo()
otra_instancia = OtraClaseEjemplo()

import modulo_personalizado
import modulo_estandar
```

GUIA DE ESTILO...

- Cada clase debe tener una cadena de documentación (docstring) inmediatamente después de la definición de la clase. La cadena de documentación debe ser una breve descripción de lo que hace la clase, y debes seguir las mismas convenciones de formato que usaste para escribir cadenas de documentación en funciones.

```
"""
Módulo personalizado para realizar tareas específicas.
Contiene varias clases que pueden ser útiles en diversos escenarios.
"""

class MiClase:
    """Esta es una clase de ejemplo que muestra cómo usar docstrings."""
    def __init__(self):
        pass
```

GUIA DE ESTILO...

- Cada módulo también debe tener una cadena de documentación que describa para qué se pueden usar las clases en un módulo.

```
"""
Módulo personalizado para realizar tareas específicas.
Contiene varias clases que pueden ser útiles en diversos escenarios.
"""

class MiClase:
    """Esta es una clase de ejemplo que muestra cómo usar docstrings."""
    def __init__(self):
        pass
```

GUIA DE ESTILO...

- Puedes usar líneas en blanco para organizar el código, pero no las uses en exceso. Dentro de una clase, puedes usar una línea en blanco entre métodos, y dentro de un módulo, puedes usar dos líneas en blanco para separar clases.

```
class MiClase:
    def __init__(self):
        pass

    def metodo_uno(self):
        pass

class OtraClase:
    def __init__(self):
        pass

    def metodo_dos(self):
        pass
```

GUIA DE ESTILO...

- Si necesitas importar un módulo de la biblioteca estándar y un módulo que escribiste, coloca la declaración de importación para el módulo de la biblioteca estándar primero. Luego agrega una línea en blanco y la declaración de importación para el módulo que escribiste. En programas con múltiples declaraciones de importación, esta convención facilita ver de dónde provienen los diferentes módulos utilizados en el programa.

```
import os

import mi_modulo_personalizado

# Resto del código del programa
```

```
# standard modules
import os

# other modules
import mi_modulo_personalizado

# Resto del código del programa
```

CONQUER BLOCKS