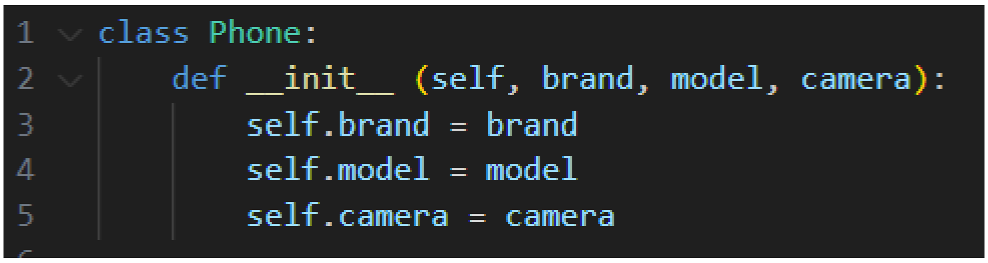
CHECKPOINT 6

1. ¿Para qué usamos Clases en Python?

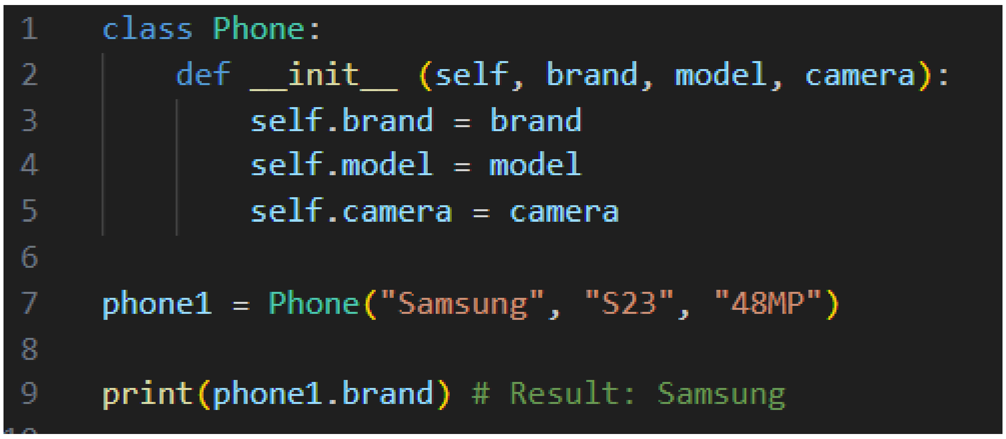
In Python, we do use classes in order to define the properties that the objects we are creating are going to have.

The syntax we use in order to create a **Class** is as follows in the next example, where we are creating the phone class, that contains the **attributes** brand, model and camera:



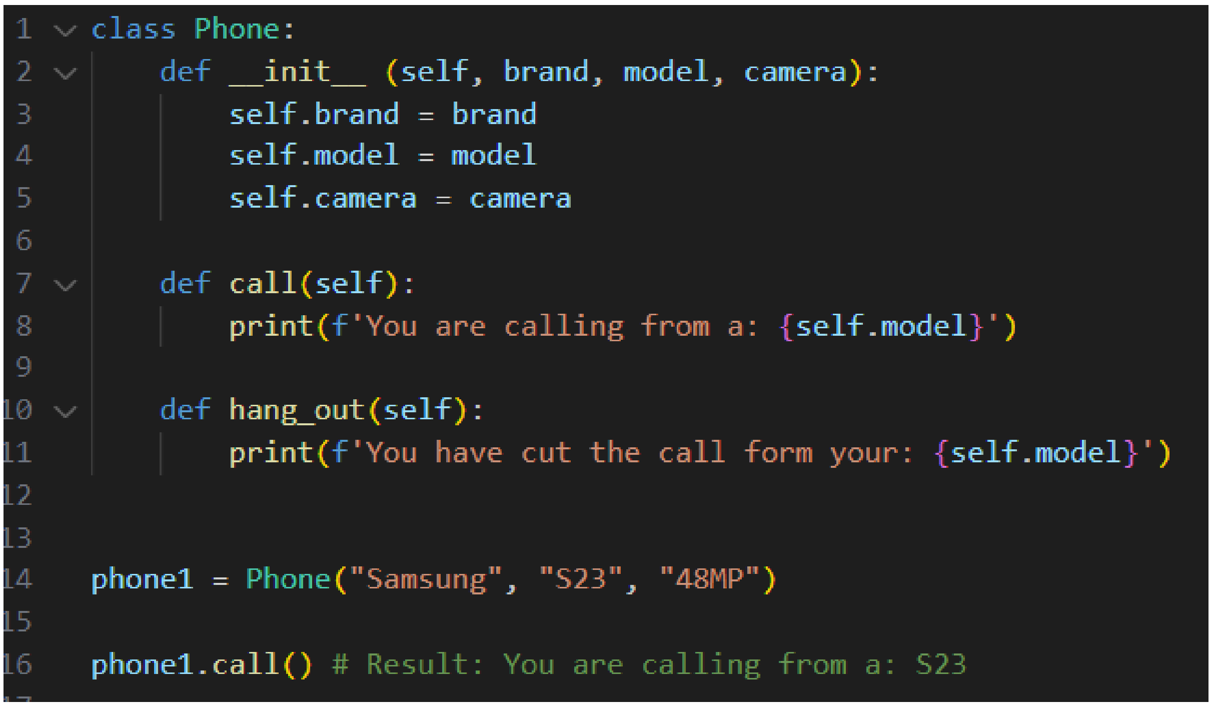
```python
class Phone:
    def __init__ (self, brand, model, camera):
        self.brand = brand
        self.model = model
        self.camera = camera
```

Once we have created the class, we can create objects by **instantiating the class**, which is the same as saying that we are creating the **object** as the result of passing it through the class.



```python
class Phone:
    def __init__ (self, brand, model, camera):
        self.brand = brand
        self.model = model
        self.camera = camera

phone1 = Phone("Samsung", "S23", "48MP")

print(phone1.brand) # Result: Samsung
```

As we can see, an object can have attributes, as we showed the attributes brand, model and camera in our example, but also can perform actions. These actions we are talking about are called **methods**, and are basically functions we create inside a class. Here we can see how we can create the methods call and hang_out in our example:



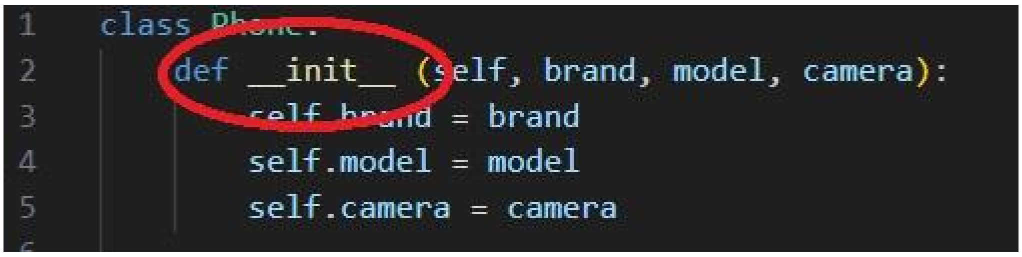
```python
class Phone:
    def __init__ (self, brand, model, camera):
        self.brand = brand
        self.model = model
        self.camera = camera

    def call(self):
        print(f'You are calling from a: {self.model}')

    def hang_out(self):
        print(f'You have cut the call form your: {self.model}')


phone1 = Phone("Samsung", "S23", "48MP")

phone1.call() # Result: You are calling from a: S23
```

2. ¿Qué método se ejecuta automáticamente cuando se crea una instancia de una clase?

In Python, a **constructor** is a **special method** that is called when an object is created. The purpose of a python constructor is to assign values to the attributes of the class.

The syntax used for creating a constructor is to create a function with **__init__** name.

We have already used the constructor in the example we walked through previously, as we can see in the next picture:



```python
class Phone:
    def __init__ (self, brand, model, camera):
        self.brand = brand
        self.model = model
        self.camera = camera
```

3. ¿Cuáles son los tres verbos de API?

There are a few verbs, specific to the HTTP protocol, that are used to define very specific operations on the API resources.

The most used ones are : GET, PUT and DELETE.

These methods allow you to create endpoints in order to perform the specific operation each one of them can do. Here you have what each of then can perform:

- **GET:**

GET means what is called the show route or the show endpoint where you are querying for a single guide.

- **PUT:**

With PUT we can update or edit a specific guide.

- **DELETE:**

With the DELETE method we can create a function that is going to take in a specific guide, and then it's going to remove it from the database.

4. ¿Es MongoDB una base de datos SQL o NoSQL?

MongoDB is a document oriented open source NoSQL database system. Opposite to SQL databases that store data in a structured way, NoSQL databases store data in its original format, making them more scalable.

Instead of storing data in tables, as in SQL databases, MongoDB stores BSON data structures (similar to JSON) in a dynamic scheme, so data integration in some parts of the applications are easier and faster.

You can create a MongoDB collection and then insert documents to the collection and query specific documents in a MongoDB collection.

Thanks to the document model used in MongoDB, information can be embedded inside a single document rather than relying on expensive join operations from traditional relational databases. This makes queries much faster, and returns all the necessary information in a single call to the database.

5. ¿Qué es una API?

API goes for Application Programming Interface. It is a set of well-defined rules used to formally specify communication between two software components so they can share data.

One type of API is the **API REST.** They use the HTTP protocol to send and receive data. REST goes for REpresentational State Transfer. This is an architectural style for providing standards between computer systems on the web, making it easier for systems to communicate with each other. These systems are characterized by how they are stateless and separate the concerns of client and server.

Systems that follow the REST paradigm are stateless, meaning that the server does not need to know anything about what state the client is in and vice versa. In this way, both the server and the client can understand any message received, even without seeing previous messages.

In the REST architectural style, the implementation of the client and the implementation of the server can be done independently without each knowing about the other. This means that the code on the client side can be changed at any time without affecting the operation of the server, and the code on the server side can be changed without affecting the operation of the client.

The most used format for sending information is JSON (JavaScript Object Notation).

APIs can be public or private. A private API requires authentication. When you authenticate for the first time, the server returns you a token. A token is an object that contains the authentication data. When you request additional information, the server will check if the token is still available and, if so, will not ask you for a new authentication.

APIs can be local or remote. The local APIs are the ones that are executed in the same environment. For example, if you are developing an Android app and you need to allow notifications to make the phone vibrate, in that case you connect with the phone’s vibration API, so everything happens in the phone. With remote APIs you use data from an application that is outside, in another place. Remote APIs can use web services and, in that case, they can use REST architecture.

6. ¿Qué es Postman?

Postman is an application that allows you to test APIs, allowing you to send requests to web services and see results.

Postman offers a user-friendly interface that makes the system of API development easy. The application lets you create, take a look at, and manage APIs without the need for complex coding, making it accessible to beginners and skilled programmers.

Postman allows you to store your API requests, so you can check and reuse them in the future. With collections, you can link related API elements together for easy editing, sharing, testing, and reuse. It helps arrange and group related requests, so you can reference and reuse particular API calls faster.

It's so easy to use Postman. Let's imagine that you want to test the next API: https://pokeapi.co/api/v2/pokemon/ditto.

You have to open a new request in Postman, select the GET method and paste the API in the available space for it. Once you have it, you have to click send and, if there's no error, the code 200 is going to appear in your screen and this way Postman is going to communicate with that outside API and then, whenever it gets data back, it's going to return it and you are going to be able to see the returned data on the screen.

Depending on the type of code you get back, the meaning is going to be different. HTTP status codes are grouped into five classes, each of them beginning with a number that represents the type of response. The classes are:

- **1XX:** These codes indicate that the server has received the request and is processing it. They are primarily used to manage communication between the client and server during the early stages of a request-response cycle.

- **2XX:** These codes indicate that the client's request was successfully received, understood, and processed by the server.

- **3XX:** These codes indicate that the client needs to take additional actions to fulfill the request. They are often used when the requested resource has moved to a different location.

- **4XX:** These codes indicate that there was an issue with the client's request, such as a mistyped URL or invalid credentials.

- **5XX:** These codes indicate that the server encountered an error while trying to fulfill the client's request, such as, 500 Internal Server Error, 502 Bad Gateway or 503 Service Unavailable.

7. ¿Qué es el polimorfismo?

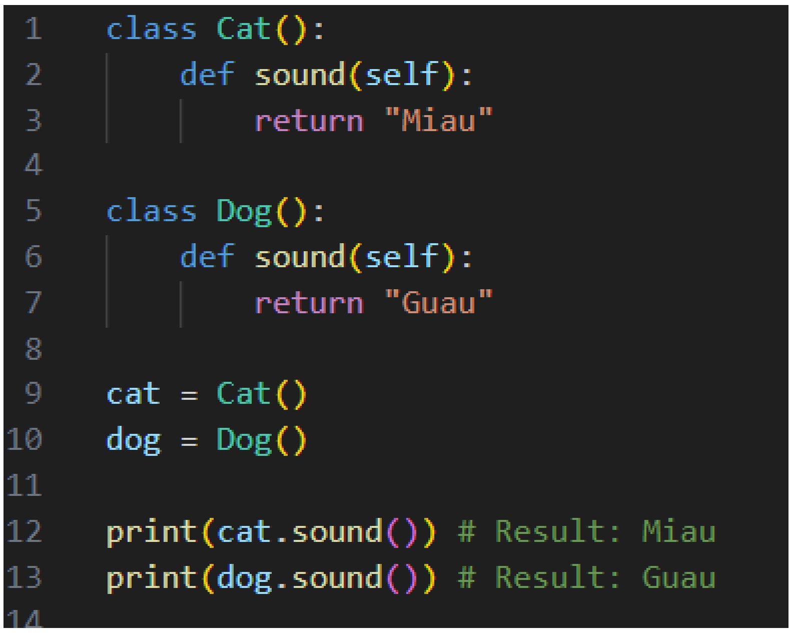
In Python, **polymorphism** is often used in Class methods, where we can have multiple classes with the same method name.

The concept itself, refers to the fact of being able to send the same kind of message to different objects.

Even if we are giving the same method to different objects, their behavior it's different since their properties are different as well.

For example, let's imagine we have the classes Cat and Dog. We are going to walk through three ways of how we can have polymorphism, linked to the sound each animal performs:

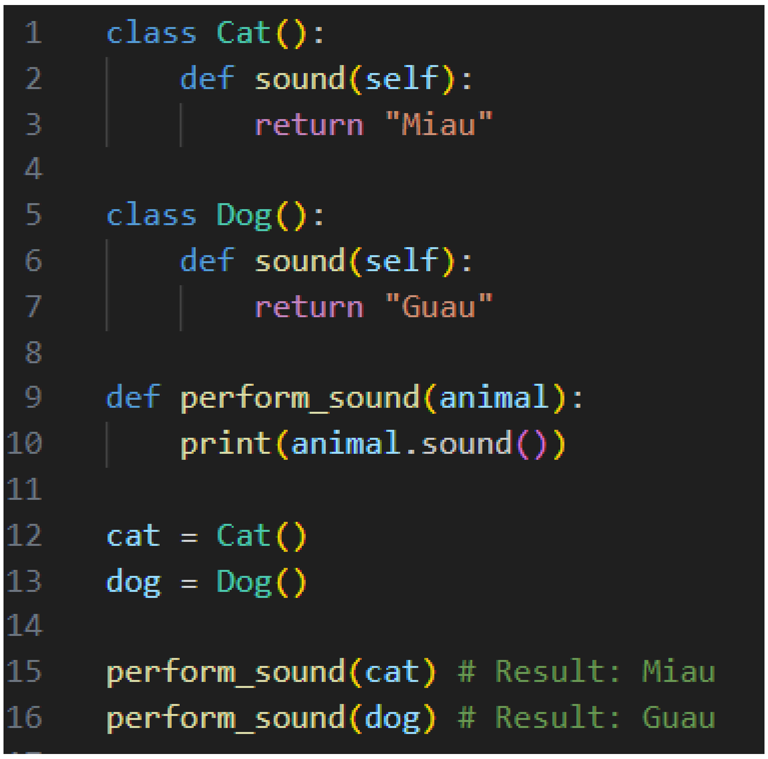
1. **By executing the same method for different objects:**



```python
class Cat():
    def sound(self):
        return "Miau"

class Dog():
    def sound(self):
        return "Guau"

cat = Cat()
dog = Dog()

print(cat.sound()) # Result: Miau
print(dog.sound()) # Result: Guau
```

Here we have polymorphism since we are sending the same message but we are changing the object where the method is being implemented.

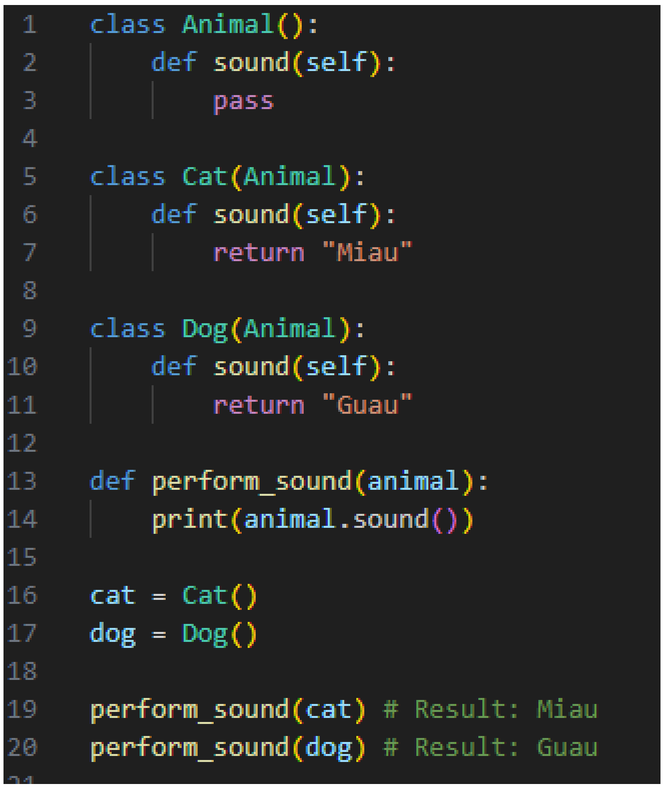
2. **Same function but different arguments:**



```python
class Cat():
    def sound(self):
        return "Miau"

class Dog():
    def sound(self):
        return "Guau"

def perform_sound(animal):
    print(animal.sound())

cat = Cat()
dog = Dog()

perform_sound(cat) # Result: Miau
perform_sound(dog) # Result: Guau
```

Here we are passing different arguments to the same function.

3. **Inheritance polymorphism:**



```python
class Animal():
    def sound(self):
        pass

class Cat(Animal):
    def sound(self):
        return "Miau"

class Dog(Animal):
    def sound(self):
        return "Guau"

def perform_sound(animal):
    print(animal.sound())

cat = Cat()
dog = Dog()

perform_sound(cat) # Result: Miau
perform_sound(dog) # Result: Guau
```

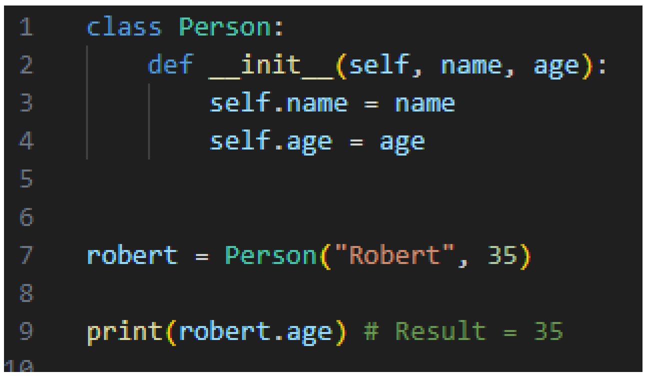
Here we are inheriting from the **Parent class**, Animal.

8. ¿Qué es un método dunder?

In Python, dunder methods are functions with a special name reserved for them. These dunder methods are created with the aim of creating special functionalities that with normal methods we couldn't create.

We have already seen the **constructor** method, that it's a dunder method used for creating the object.

- **__init__:**

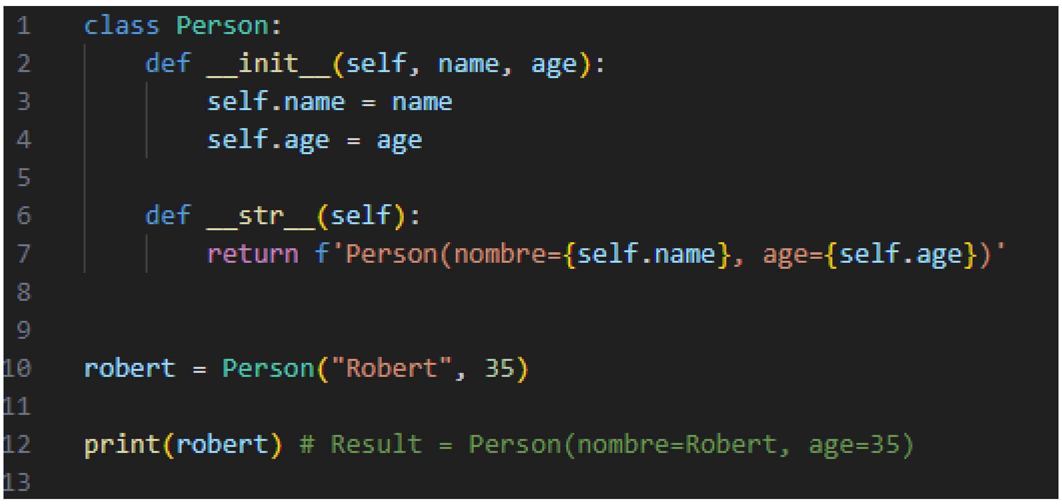


```python
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age


robert = Person("Robert", 35)

print(robert.age) # Result = 35
```

Apart from the constructor, we have some other dunder methods, as we are going to see below:

- **__str__:**

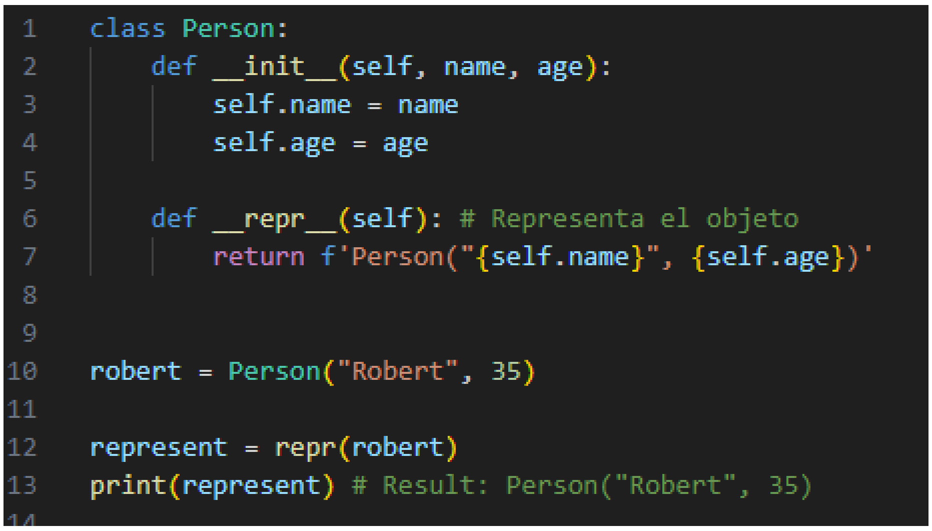
The __str__() dunder method returns a reader-friendly string representation of a class object. Returns a representation of the object in a string.



```python
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def __str__(self):
        return f'Person(nombre={self.name}, age={self.age})'


robert = Person("Robert", 35)

print(robert) # Result = Person(nombre=Robert, age=35)
```

- **__repr__:**

The __repr__() dunder method returns the string representation of the object or class.



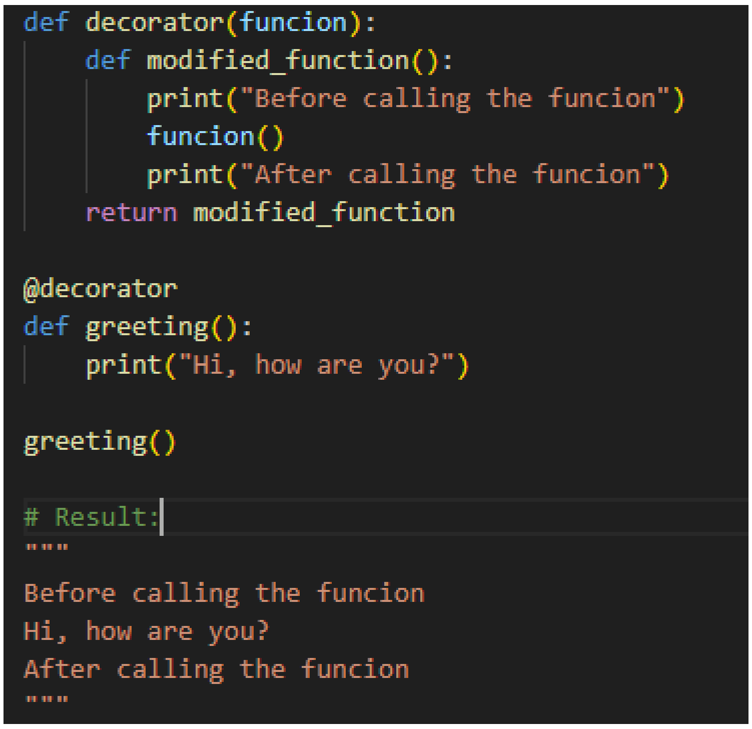
```python
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def __repr__(self): # Representa el objeto
        return f'Person("{self.name}", {self.age})'


robert = Person("Robert", 35)

represent = repr(robert)
print(represent) # Result: Person("Robert", 35)
```

9. ¿Qué es un decorador de python?

Decorators in Python are a special function that decorates another function. Adds extra code to the function we already had, it could be before or after.
Decorators don't change the function's original code but add an extra functionality to the already existing one.

The syntax for using decorator is by using by adding **@**.

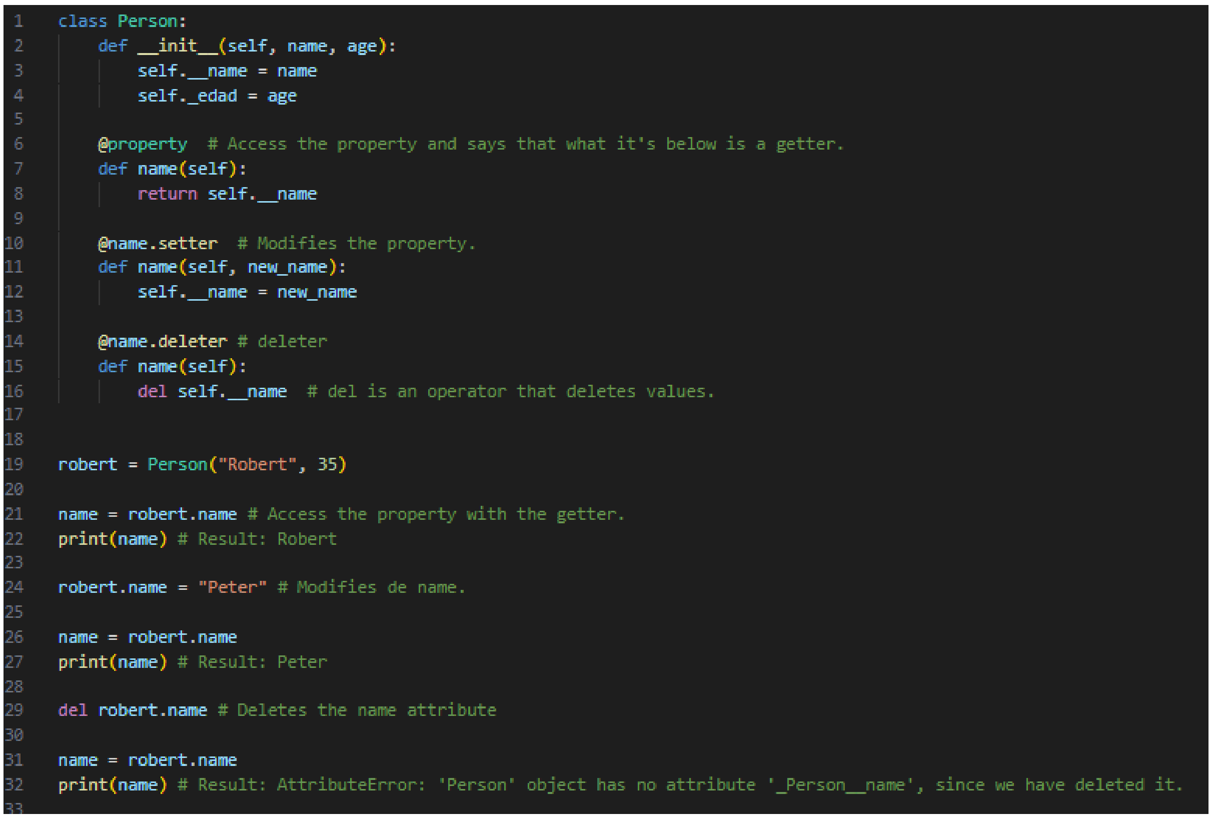
With the next example we can see how we can use decorators to add extra functionalities to our original code:



```python
def decorator(funcion):
    def modified_function():
        print("Before calling the funcion")
        funcion()
        print("After calling the funcion")
    return modified_function

@decorator
def greeting():
    print("Hi, how are you?")

greeting()

# Result:
"""
Before calling the funcion
Hi, how are you?
After calling the funcion
"""
```

Inside decorators, we have **@property**, which is used to assign a "special" functionality to certain methods so that they act as **getters**, **setters** or **deleters** when defining and using properties in a class. This way we can access private attributes of a class that are defined by using an **_** (one underscore) or **__** (two underscores), depending on how private they are, in the __init__ method.

With the next example qe are going to see how we can define getters, setters, and deleters with @property:



```python
class Person:
    def __init__(self, name, age):
        self.__name = name
        self._edad = age

    @property  # Access the property and says that what it's below is a getter.
    def name(self):
        return self.__name

    @name.setter  # Modifies the property.
    def name(self, new_name):
        self.__name = new_name

    @name.deleter # deleter
    def name(self):
        del self.__name  # del is an operator that deletes values.


robert = Person("Robert", 35)

name = robert.name # Access the property with the getter.
print(name) # Result: Robert

robert.name = "Peter" # Modifies de name.

name = robert.name
print(name) # Result: Peter

del robert.name # Deletes the name attribute

name = robert.name
print(name) # Result: AttributeError: 'Person' object has no attribute '_Person__name', since we have deleted it.
```