16

Object-Oriented Programming

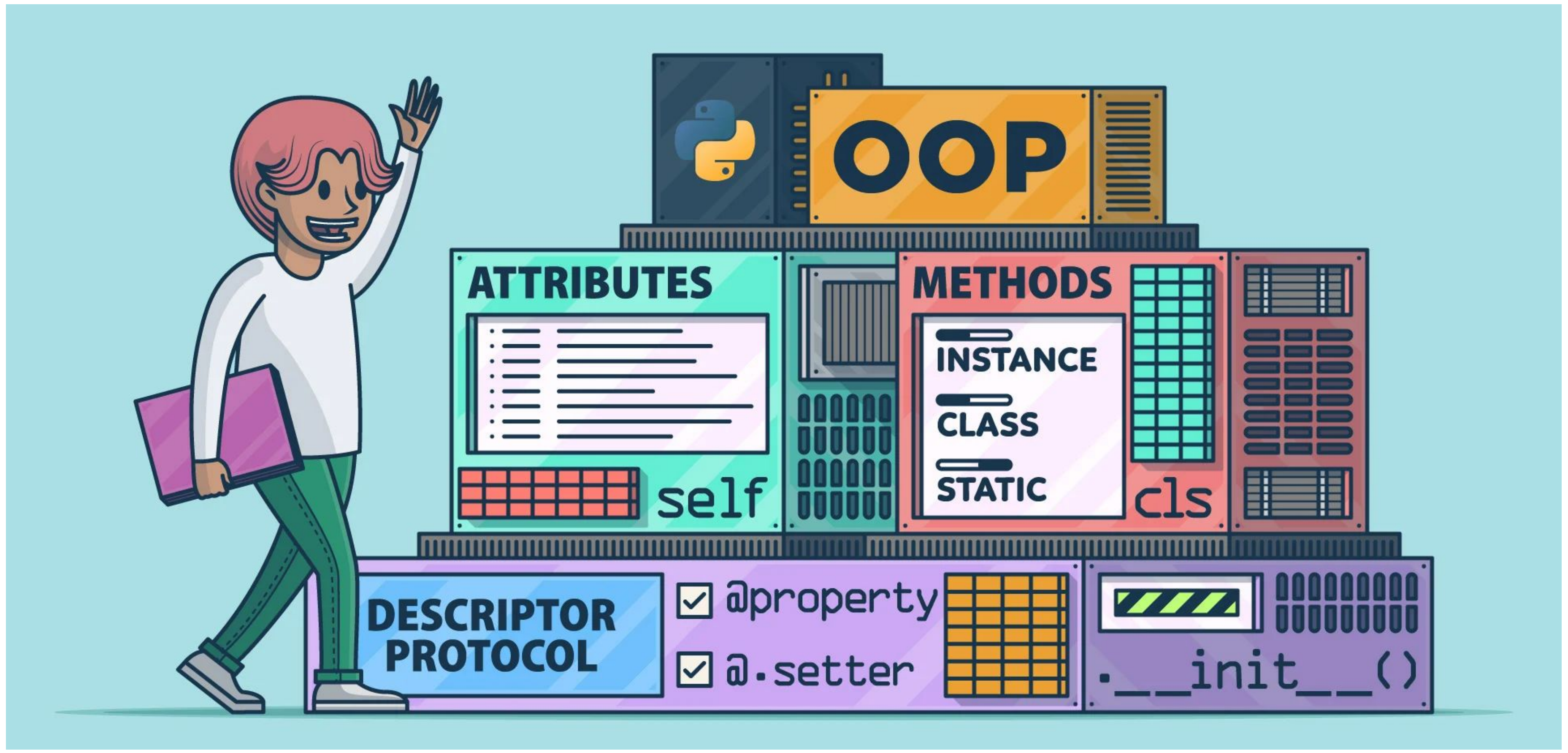
by Gladden Rumao

CSA221 : Data Structures and Algorithms

Newton School of Technology

Object Oriented Programming :

The main concept of object-oriented Programming (OOPs) or oops concepts in Python is to bind the data and the functions that work together as a single unit so that no other part of the code can access this data.



Object Oriented Programming :

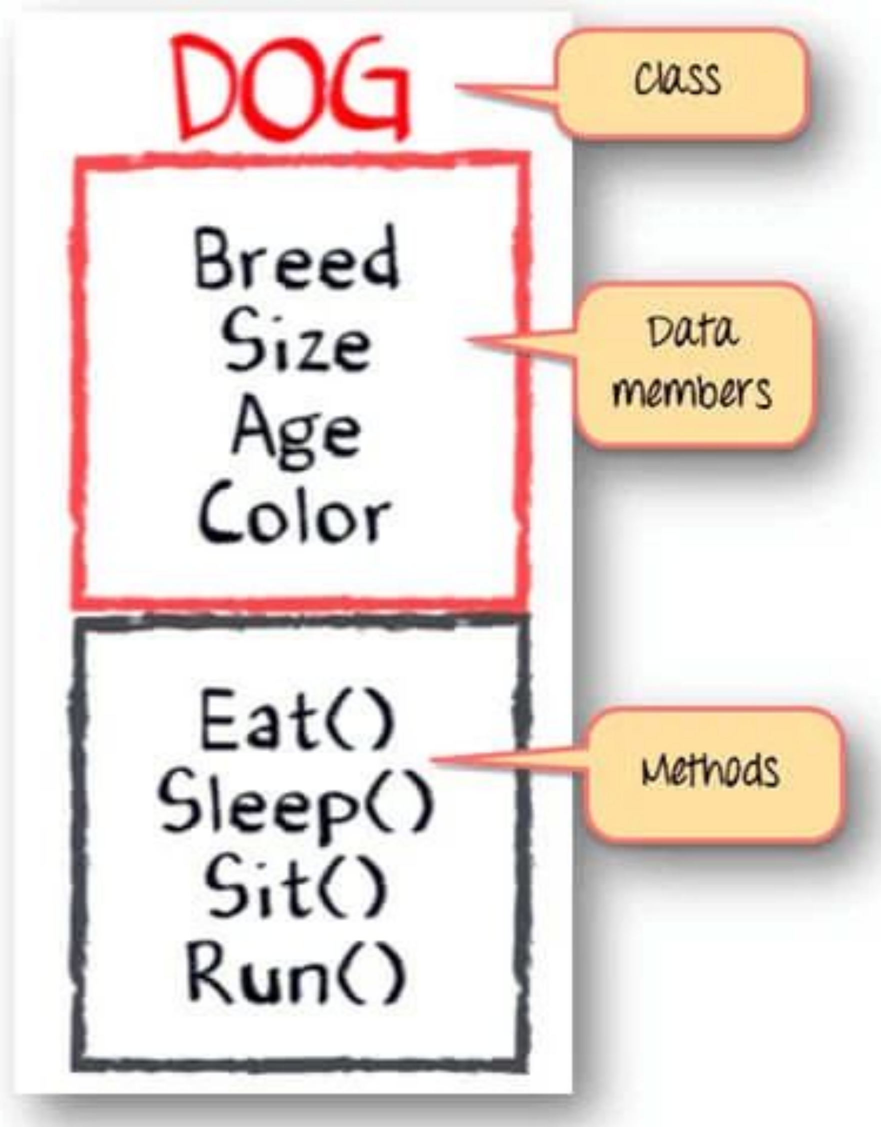
Problem: Managing Library Books
Imagine we need to manage book data in a library. The data we need to track includes:

- Title
- Author
- Price

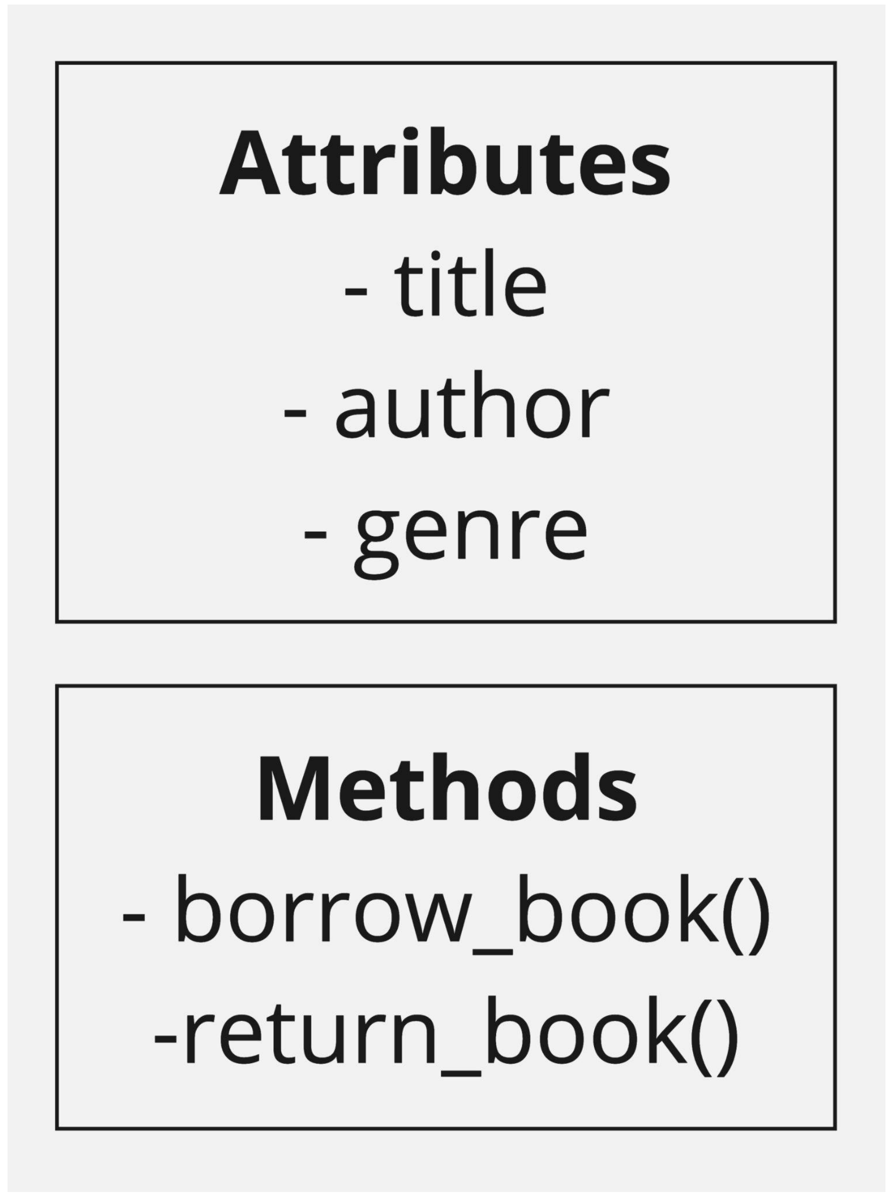
We also need actions or behaviors related to the book, like:

- Borrow
- Return

Attributes and Methods :



Attributes and Methods :

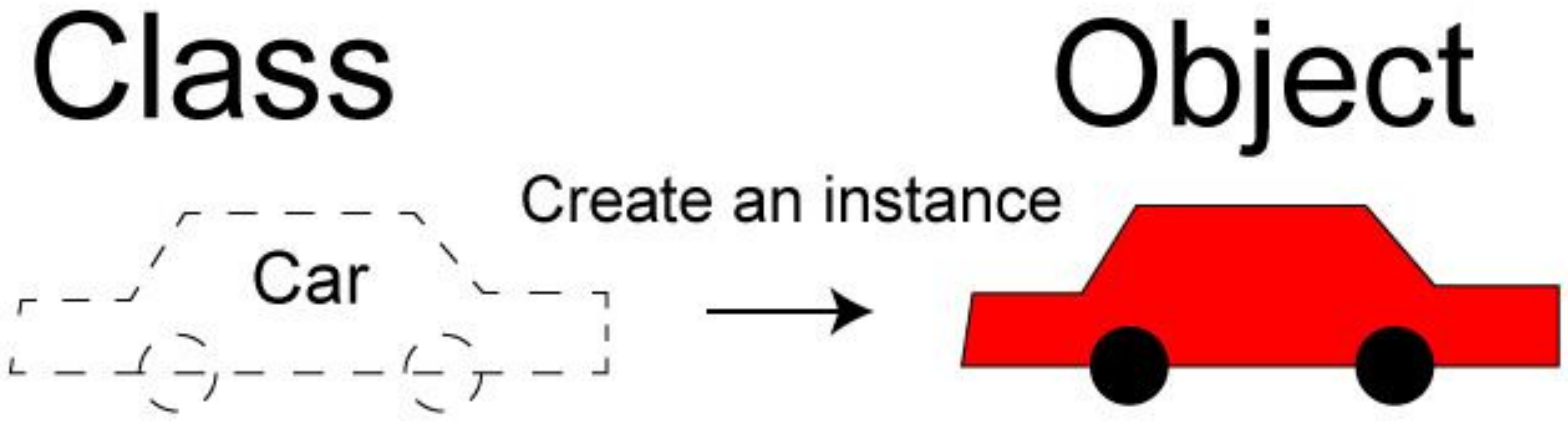


Attributes and Methods :

- **Attributes: The Data**
  - The data about the book, like Title, Author, and Price, will become attributes of a class.
  - Attributes store information related to the object.

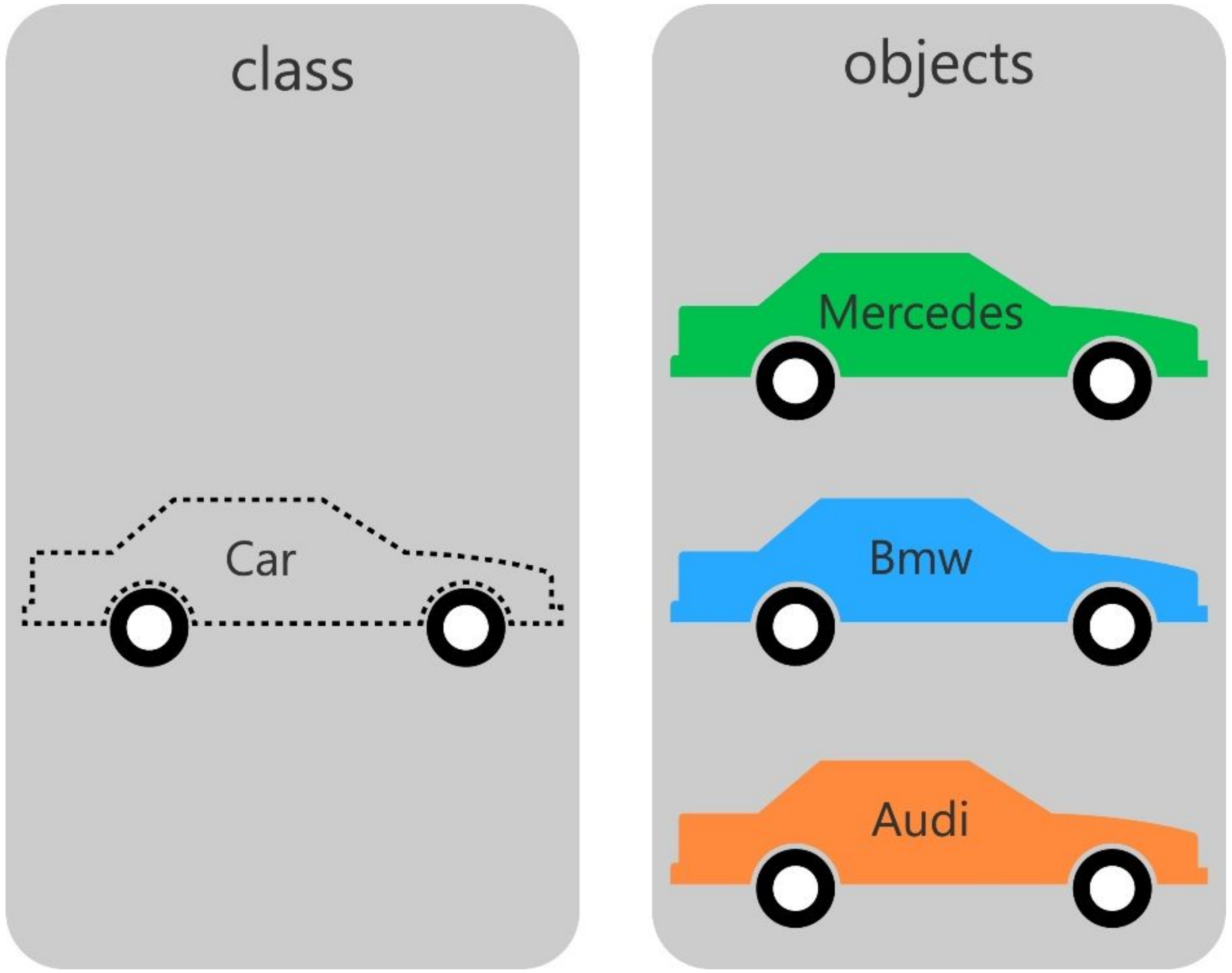
- **Methods: The Actions**
  - The actions we need to perform, like displaying book details and updating the price, will be methods of the class.
  - Methods define what actions the object can perform.

What is Class and Object?



Newton School of Technology

What is Class and Object?



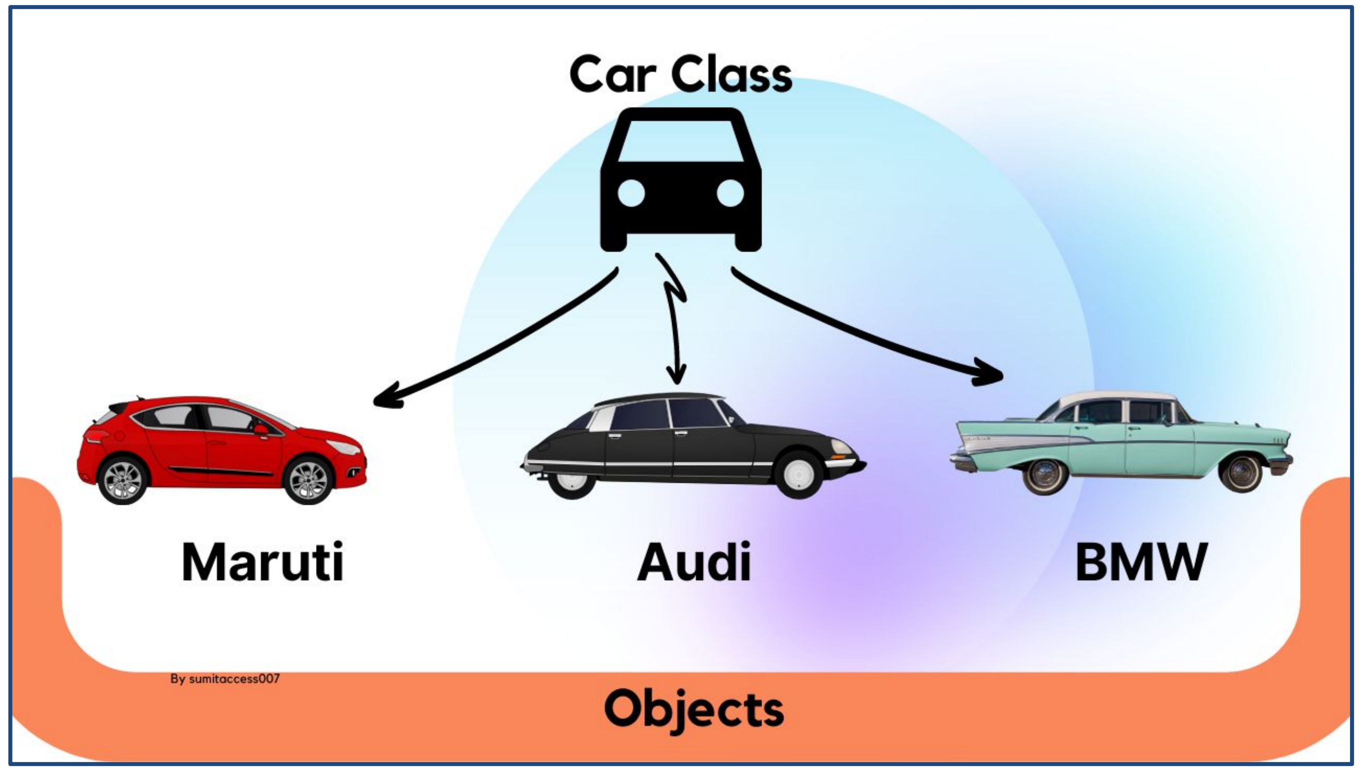
What is Class and Object?

Class: Blueprint for Objects

- A **class** is like a **blueprint** or **template**.
- It defines what attributes and methods an object will have.

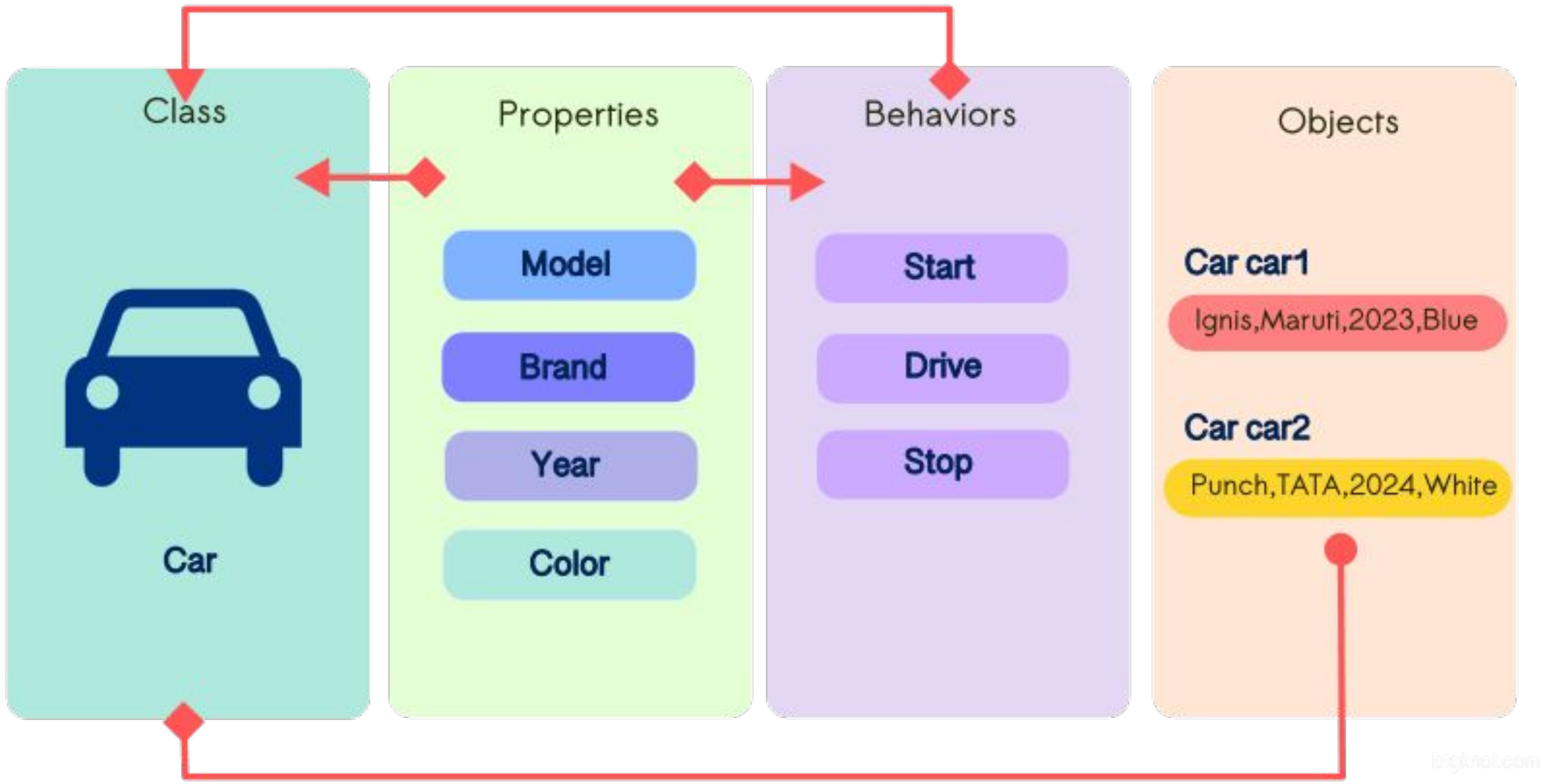
Object: An Instance of a Class

- An **object** is a **specific instance** created from a class.
- Think of an object as a **real thing** based on the blueprint.
  -



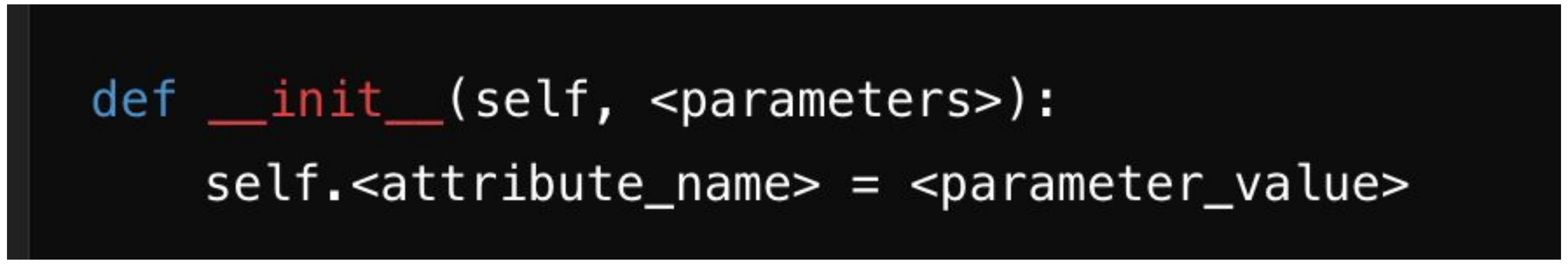
Newton School of Technology

What is Class and Object?

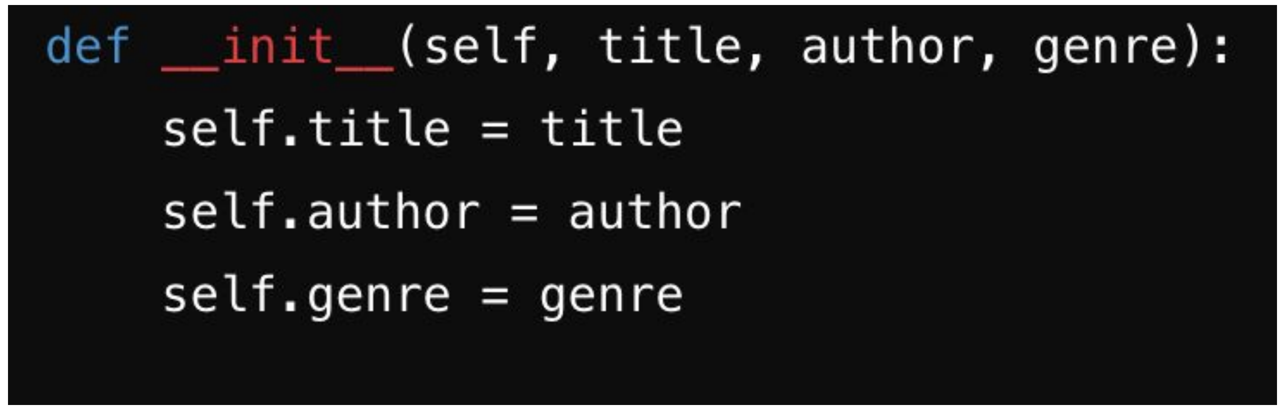


Constructor

- A **constructor** is a special method used to **initialize objects** when they are created from a class.
- The constructor method in Python is always named **`__init__()`**.
- It is automatically called when an **object** is instantiated from the class.
- **Purpose**: To set up the initial state of an object by initializing its attributes.

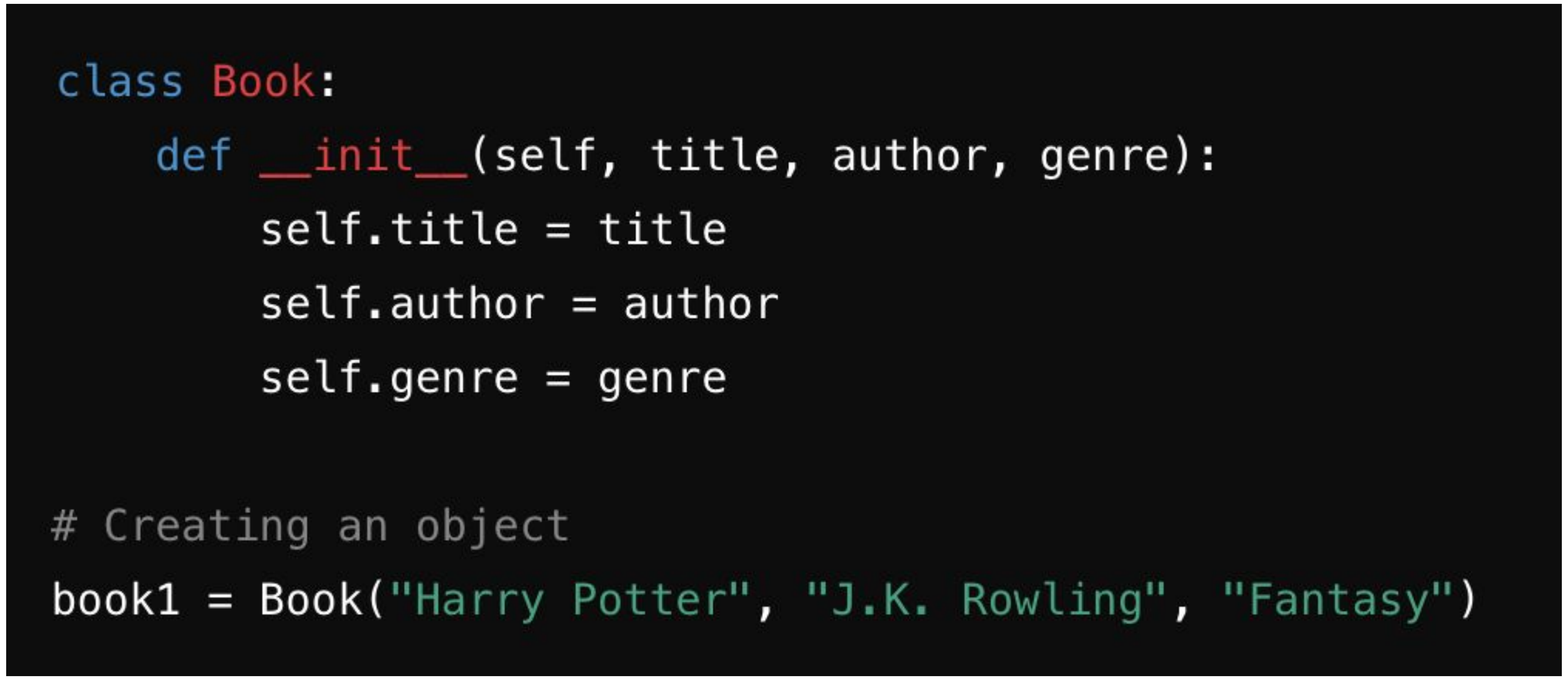


```
def __init__(self, <parameters>):
    self.<attribute_name> = <parameter_value>
```



```
def __init__(self, title, author, genre):
    self.title = title
    self.author = author
    self.genre = genre
```

What is an Object?



```
class Book:
    def __init__(self, title, author, genre):
        self.title = title
        self.author = author
        self.genre = genre

# Creating an object
book1 = Book("Harry Potter", "J.K. Rowling", "Fantasy")
```

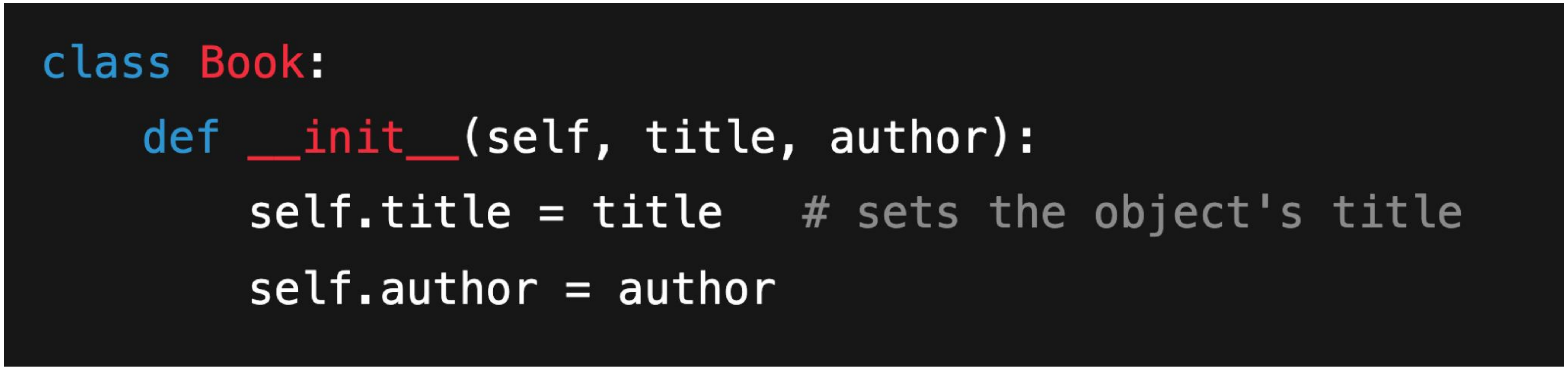
What is self in Python?

- self is a reference to the **current instance** of the class.
- It is used to **access attributes and methods** of the class in Python.
- It must be the first parameter of methods in class definitions (like __init__).

```python
class Book:
    def __init__(self, title, author):
        self.title = title    # sets the object's title
        self.author = author
```

Why self is Important?

- It tells Python which object's attribute or method we are referring to.
- Without self, all objects would share the same attributes, breaking the concept of instances.
- **self.title** refers to the attribute of the specific Book object being created.
- **title** on the right is the argument passed during object creation.



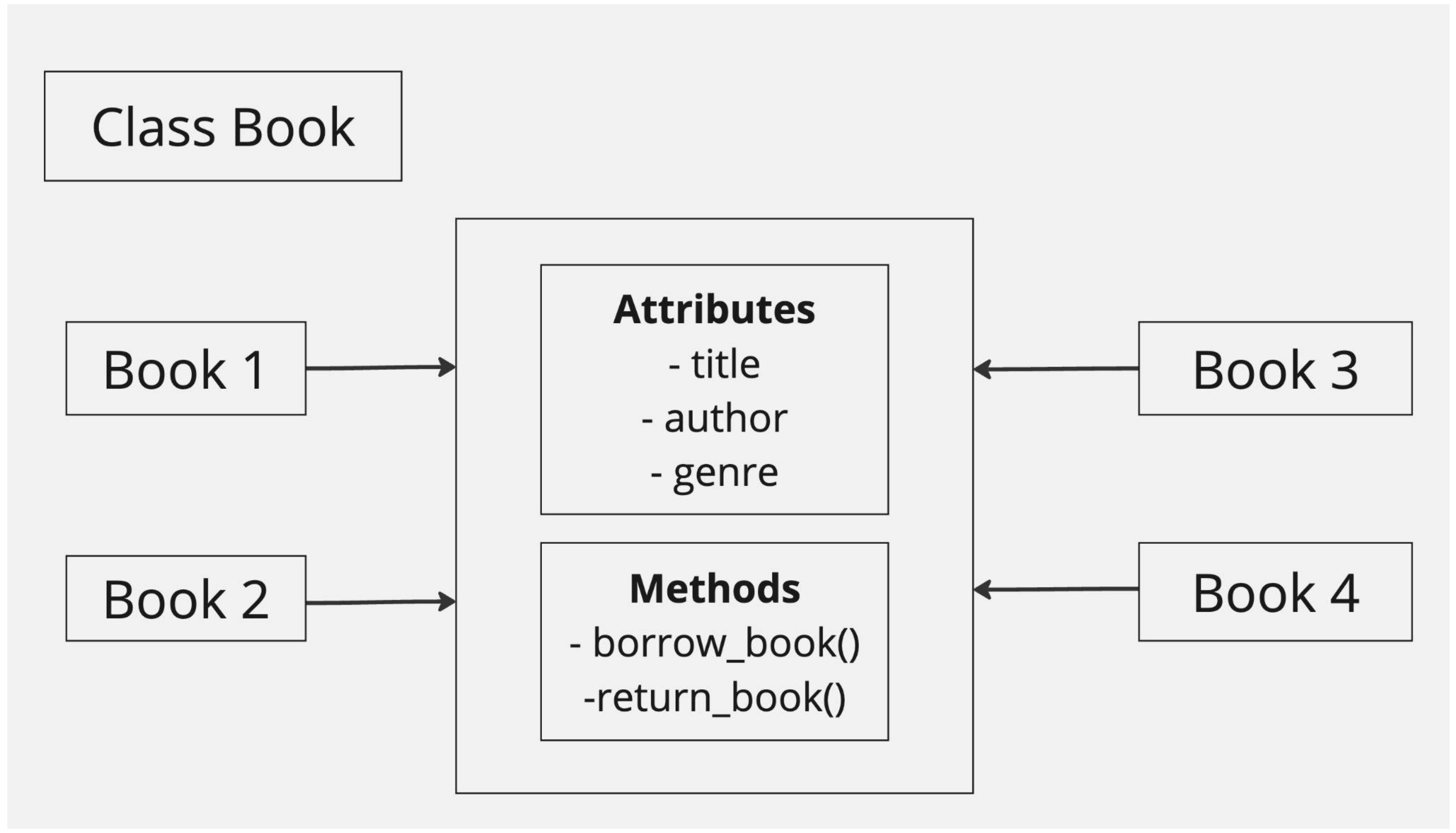
```python
class Book:
    def __init__(self, title, author):
        self.title = title   # sets the object's title
        self.author = author
```

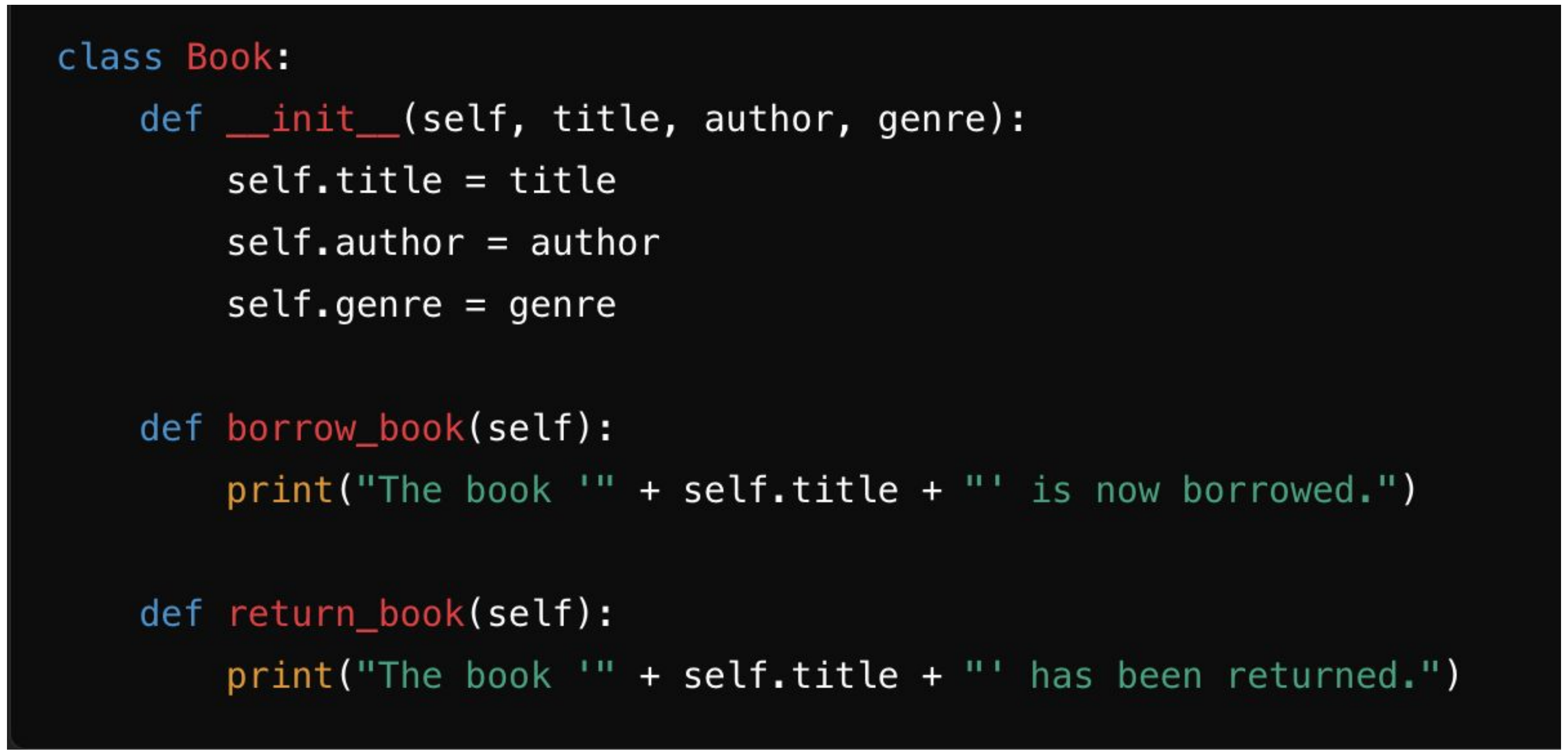
Multiple Objects from the Same Class

You can create many objects (instances) from the same class, each having its own unique data.

Example:
book1 and book2 are both instances of the Book class, but they hold different data for attributes (title, author, genre).

```
book1 = Book("Harry Potter", "J.K. Rowling", "Fantasy")
book2 = Book("1984", "George Orwell", "Dystopian")
```

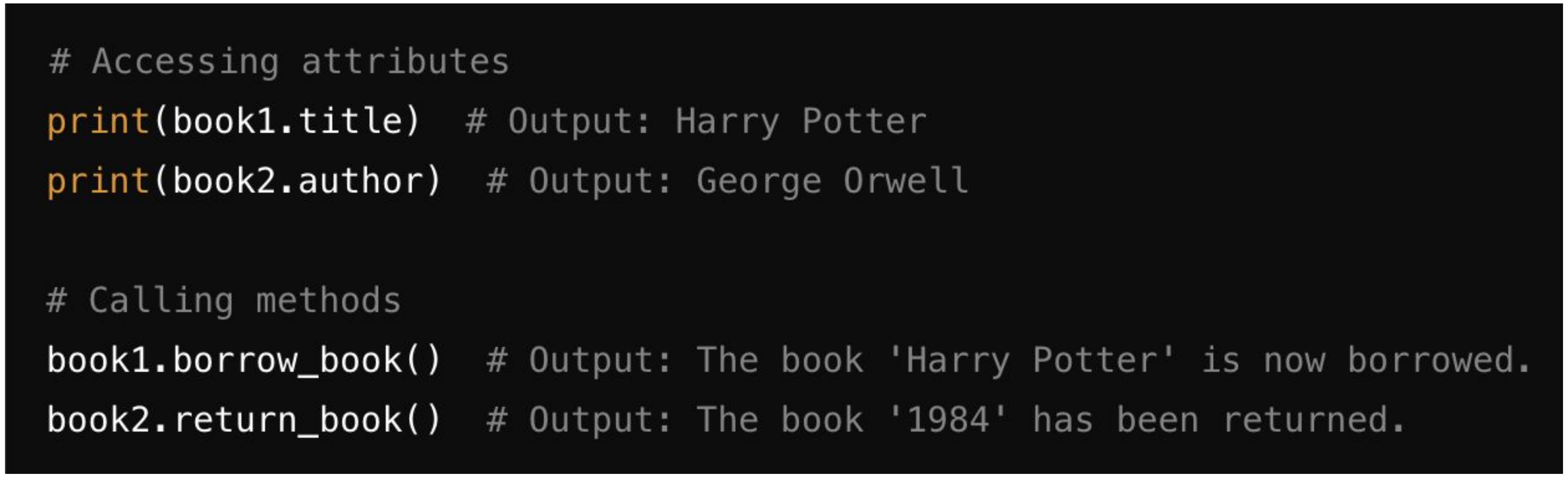




```python
class Book:
    def __init__(self, title, author, genre):
        self.title = title
        self.author = author
        self.genre = genre

    def borrow_book(self):
        print("The book '" + self.title + "' is now borrowed.")

    def return_book(self):
        print("The book '" + self.title + "' has been returned.")
```

Accessing Attributes and Methods :

Attributes: You can access attributes of an object using object.attribute.
Methods: You can call methods using object.method().



```
# Accessing attributes
print(book1.title)  # Output: Harry Potter
print(book2.author)  # Output: George Orwell

# Calling methods
book1.borrow_book()  # Output: The book 'Harry Potter' is now borrowed.
book2.return_book()  # Output: The book '1984' has been returned.
```

Instance Attribute vs Class Attribute

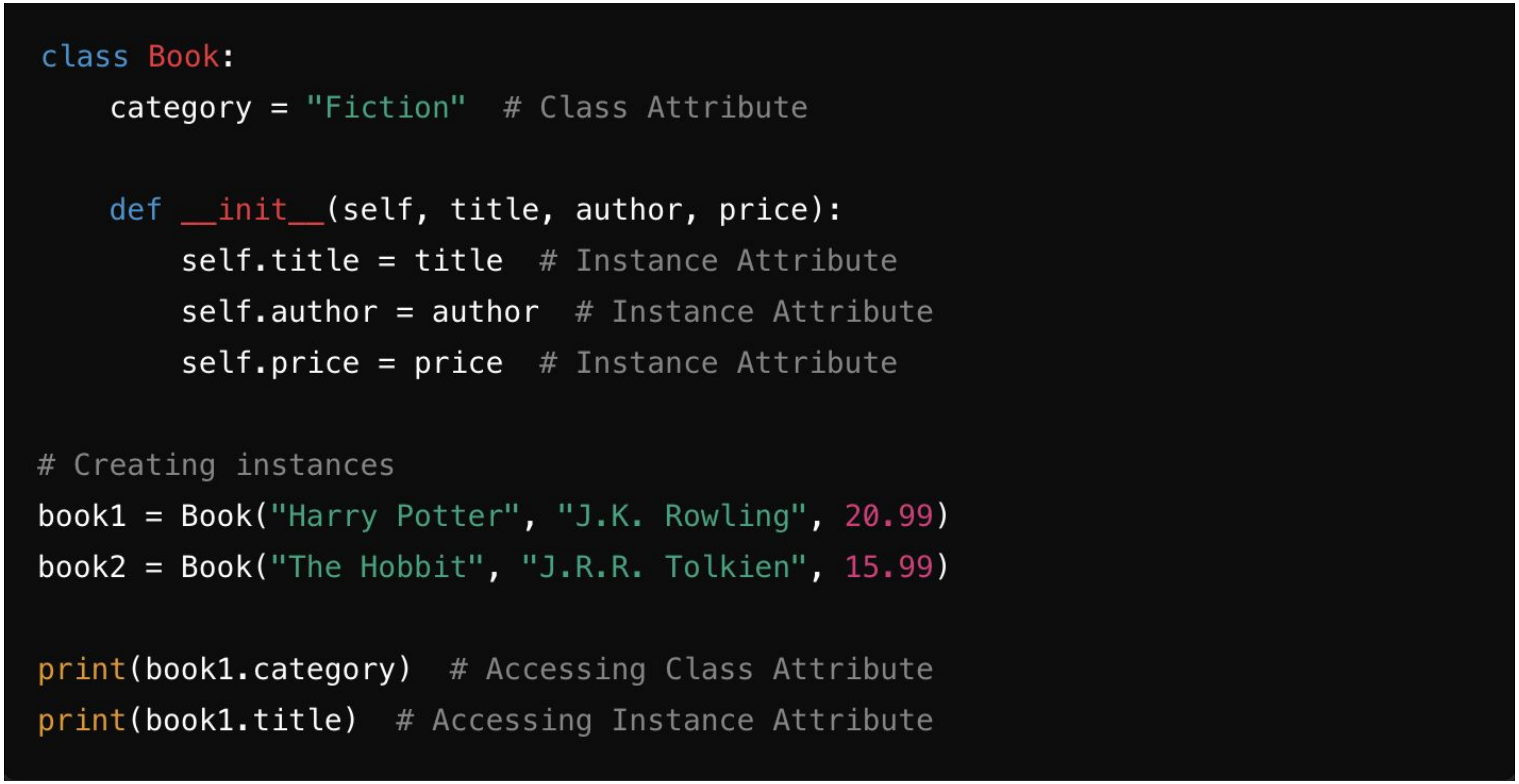
Instance Attribute

- **Defined** inside the `__init__` method.
- **Unique** for each object created from the class.
- **Accessed** using the object (e.g., `book1.title`).

Class Attribute

- **Defined** directly in the class (outside `__init__`).
- **Shared** by all instances of the class.
- **Accessed** using the class name or object (e.g., `Book.category` or `book1.category`).

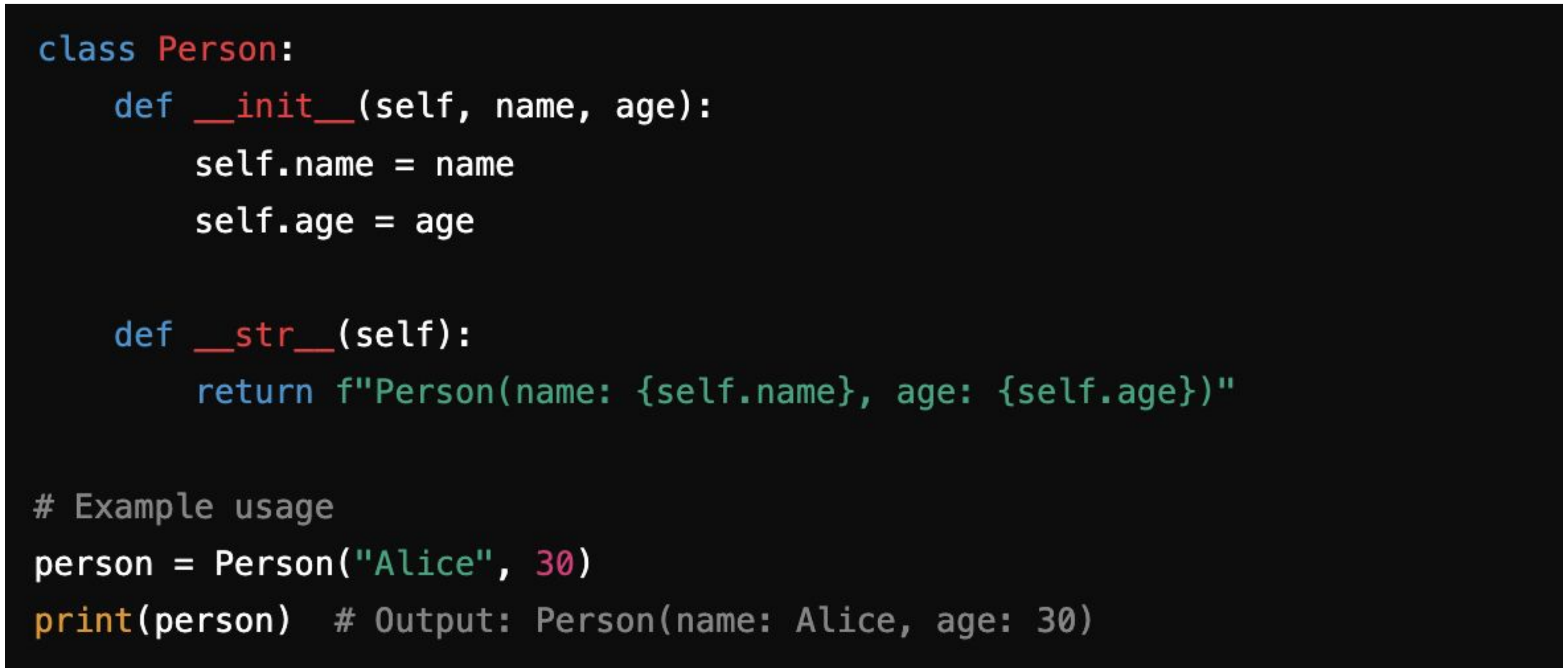
Instance Attribute vs Class Attribute



```python
class Book:
    category = "Fiction"  # Class Attribute

    def __init__(self, title, author, price):
        self.title = title  # Instance Attribute
        self.author = author  # Instance Attribute
        self.price = price  # Instance Attribute

# Creating instances
book1 = Book("Harry Potter", "J.K. Rowling", 20.99)
book2 = Book("The Hobbit", "J.R.R. Tolkien", 15.99)

print(book1.category)  # Accessing Class Attribute
print(book1.title)  # Accessing Instance Attribute
```

__str__ method

- The **__str__ method** in Python is a special method used to **provide a readable, user-friendly string representation of an object.**
- When print() is called on an object, the __str__ method is invoked if it is defined.



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

    def __str__(self):
        return f"Person(name: {self.name}, age: {self.age})"

# Example usage
person = Person("Alice", 30)
print(person)  # Output: Person(name: Alice, age: 30)
```

Thank You!