# OO Exercises

Geovanna de Almeida García

* **What is OO?**

Object Orientation (OO):

It is a programming philosophy that organizes code around objects, which are instances of classes. Classes are models that define the attributes and methods that an object will have.

Objects:

They are instances of classes. They are like real-world entities, with state (attributes) and behavior (methods).

Attributes:

It is data that an object possesses. They can be of different types (integers, strings, objects, etc.).

Methods:

These are functions that are associated with an object. They can access and modify the attributes of this object, as well as perform other operations.

Functions:

These are blocks of code that perform a specific task. They can take arguments and return values. They can be defined anywhere in the code and are called by their name.

Difference Between Methods and Functions:

The main difference is that methods are associated with objects and can access their attributes, while functions are not associated with objects and can be called anywhere in the code.

* **Advantages of OO?**

reuse of code across classes and objects, modular design organization, and ease of maintenance and scalability

* **Definition of Class:**

Creating a project involves defining classes and objects to represent the problem domain and its functionalities.

* **Examples:**

**Inheritance:**

Inheritance is the mechanism that allows a class (child class) to inherit attributes and methods from another class (parent class). This promotes code reuse and hierarchical organization of objects, where child classes are specializations of a parent class.

Example: An "Animal" class can have attributes such as "name" and "color". A "Dog" class, which inherits from "Animal", will also have these attributes, in addition to being able to have specific attributes such as "breed".

**Encapsulation:**

Encapsulation is the practice of hiding the implementation details of a class and exposing only one controlled interface to interact with it. This protects the internal data of the class and makes it easier to maintain the code.

Example: A class "Bank" can have a "balance" attribute, which must be protected. Access to the balance can be controlled by public methods such as "deposit" and "withdraw", which update the balance internally, but do not expose the internal logic of the class.

**Polymorphism:**

Polymorphism means that different objects of different classes can respond to the same message in different ways. This allows the code to be written generically, with the specific implementation being determined by the object type.

Example: An "Animal" class can have an "emitSound" method. A "Dog" class can implement this method to emit a bark, while a "Cat" class can implement it to emit a meow, both responding to the same message in different ways.

# Activities

1. class Carro:
2. Def \_\_init\_\_(*Self*, *Make*, *Model*, *Color*, *Year*):
3. *self.brand* = *brand*
4. *self.model* = *model*
5. *self.cor* = *cor*
6. *self.yes* = *yes*
7. def exibir\_info(*self*):
8. print(f"Brand: {self.brand}, Model: {self.model}, Color: *{*self.color}, Year: *{*self.year}")
9. c1 = Carro("volvo", "wx", "prata", "2025")
10. c1.exibir\_info()

2. class Carro:

Def \_\_init\_\_(*Self*, *Make*, *Model*, *Color*, *Year*):

*self.brand* = *brand*

*self.model* = *model*

*self.cor* = *cor*

*self*.\_\_ano = *year*

def exibir\_info(*self*):

print(f"Brand: {self.brand}, Model: {self.model}, Color: {self.color}")

def get\_ano(*self*):

return *self*.\_\_ano

c1 = Carro("volvo", "wx", "prata", "2025")

c1.exibir\_info()

print(c1.get\_ano())

**3-** class Carro:

Def \_\_init\_\_(*Self*, *Make*, *Model*, *Color*, *Year*):

*self.brand* = *brand*

*self.model* = *model*

*self.cor* = *cor*

*self.yes* = *yes*

Def Details(*Self*):

print(f"Brand: {self.brand}, Model: {self.model}, Color: *{*self.color}, Year:*{*self.year}")

class VeiculoEletrico(Carro):

Def \_\_init\_\_(*self*, *make*, *model*, *color*, *year*, *autonomia\_bateria*):

super().\_\_init\_\_(*make*, *model*, *color*, *year*)

*self*.autonomia\_bateria = *autonomia\_bateria*

Def Details(*Self*):

print(f"Brand: {self.brand}, Model: {self.model}, Color: *{*self.color}*, Year:*{self.year}, Battery Life:*{*self.autonomia\_bateria}")

c1 = VeiculoEletrico("volvo", "wx", "prata", "2025", "2400")

c1.Details()

**Books**

class Book:  
  
 def \_\_init\_\_(self, title, author, isbn):  
 self.title = title  
  
 self.author = author  
  
 self.isbn = isbn  
  
 def exibir\_info(self):  
 return f"Title: {self.title}, Author: {self.author}, ISBN: {self.isbn}  
  
  
"class   
  
 Library(Book):def \_\_init\_\_(self  
  
 ):self.books = []  
  
 def adicionar\_livro(self, book):  
  
 self.books.append(book)print  
  
 (f"Book '{book.title}' added successfully!")  
  
 def remover\_livro(self, isbn):  
  
 for book in self.books:  
  
 if book.isbn == isbn:  
  
 self.books.remove(book)  
  
 print(f"Book '{book.title}' removed successfully!")  
  
 else:  
  
 print("Book not found.")  
  
 def listar\_livros(self):  
  
 if not self.livros:  
  
 print("No books in the library.")  
  
 else:  
  
 print("List of books in the library:")  
  
 for book in self.livros:  
 print(livro.exibir\_info())  
  
  
library = Library()  
  
book1 = Book("Dom Casmurro", "Machado de Assis", "123456")  
  
book2 = Book("The Little Prince", "Antoine de Saint-Exupéry", "654321")  
  
biblioteca.adicionar\_livro(book1)  
  
biblioteca.adicionar\_livro(book2)  
  
biblioteca.listar\_livros()  
  
biblioteca.remover\_livro("123456")  
  
biblioteca.listar\_livros()