Python

Sumário

[Estrutura de repetição ‘for’ 1](#_Toc185001732)

[Programação orientada a objeto (POO) 1](#_Toc185001733)

[Herança 2](#_Toc185001734)

[Encapsulamento 2](#_Toc185001735)

[1. Atributos e métodos públicos (padrão) 2](#_Toc185001736)

[2. Atributos e métodos privados 3](#_Toc185001737)

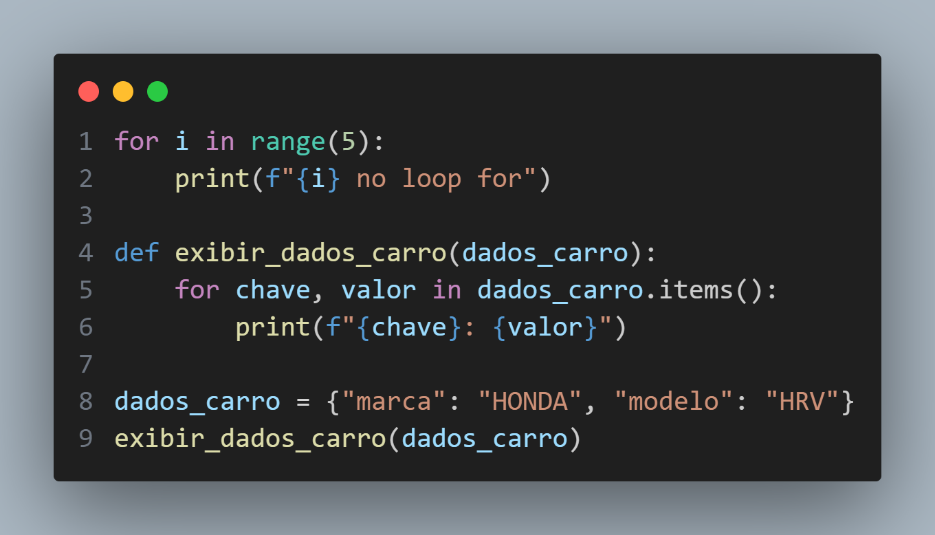
[3. Métodos de acesso 3](#_Toc185001738)

[Polimorfismo 4](#_Toc185001739)

[1. Polimorfismo em método: 4](#_Toc185001740)

[2. Polimorfismo por herança 4](#_Toc185001741)

# Estrutura de repetição ‘for’

É utilizada para iterar sobre uma sequência

Aqui usei dicionário (dados\_carro), ‘*for chave, valor in dados\_carro.items()’* esse trecho significa que o loop irá iterar sobre os pares de chave e valor do dicionário.

Items(), o método do dicionário retorna uma sequencia de tuplas, cada tupla possui um par de chave e valor. Seria algo como: [(“marca”, “HONDA”), (“modelo”, “HRV”)].

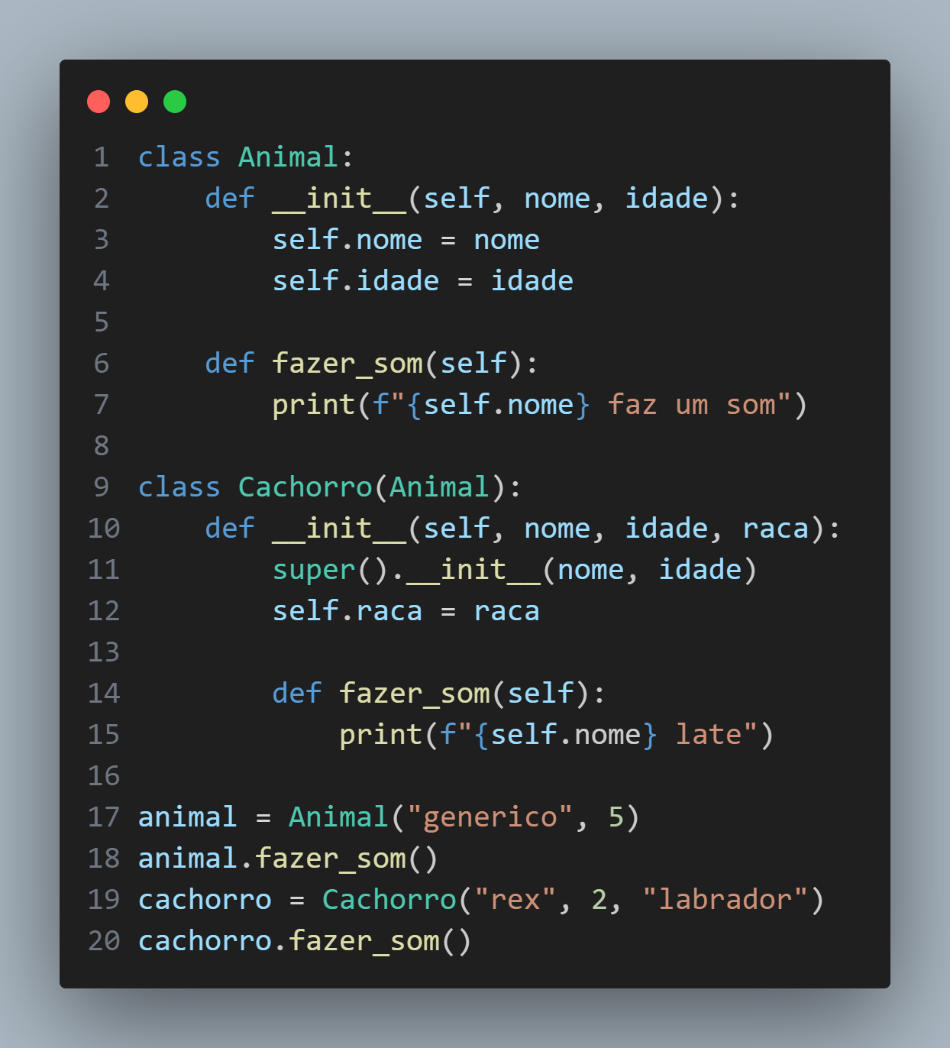
# Programação orientada a objeto (POO)

Porque usar POO?

Podemos usar objetos o que torna o programa mais “fácil” e traz diversas possibilidades que não seriam sem a utilização de objetos.

## Herança

Permite que uma classe herde atributos e métodos de outra classe. Sendo assim pode reutilizar, modificar ou estender o comportamento da classe pai.

Classe Animal: \_\_init\_\_ inicializa o nome e a idade do animal.

fazer\_som é uma função genérica para animais.

Classe Cachorro:

Herda os atributos de animal e métodos.

Porem a subclasse modifica o método fazer\_som.

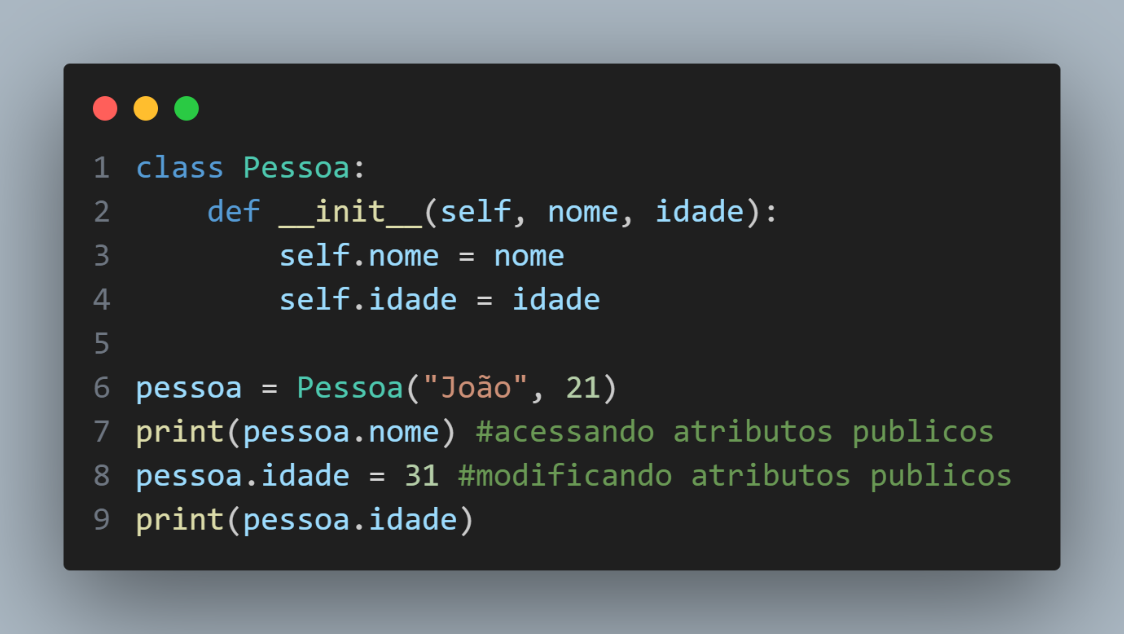
Super() para chamar o construtor de Animal.

## Encapsulamento

Permite que os dados dentro de um objeto sejam acessados e manipulados de maneira controlada.

### Atributos e métodos públicos (padrão)

Por padrão em Python todos os atributos e métodos de uma classe são públicos, ou seja, podem ser acessados diretamente de fora da classe.



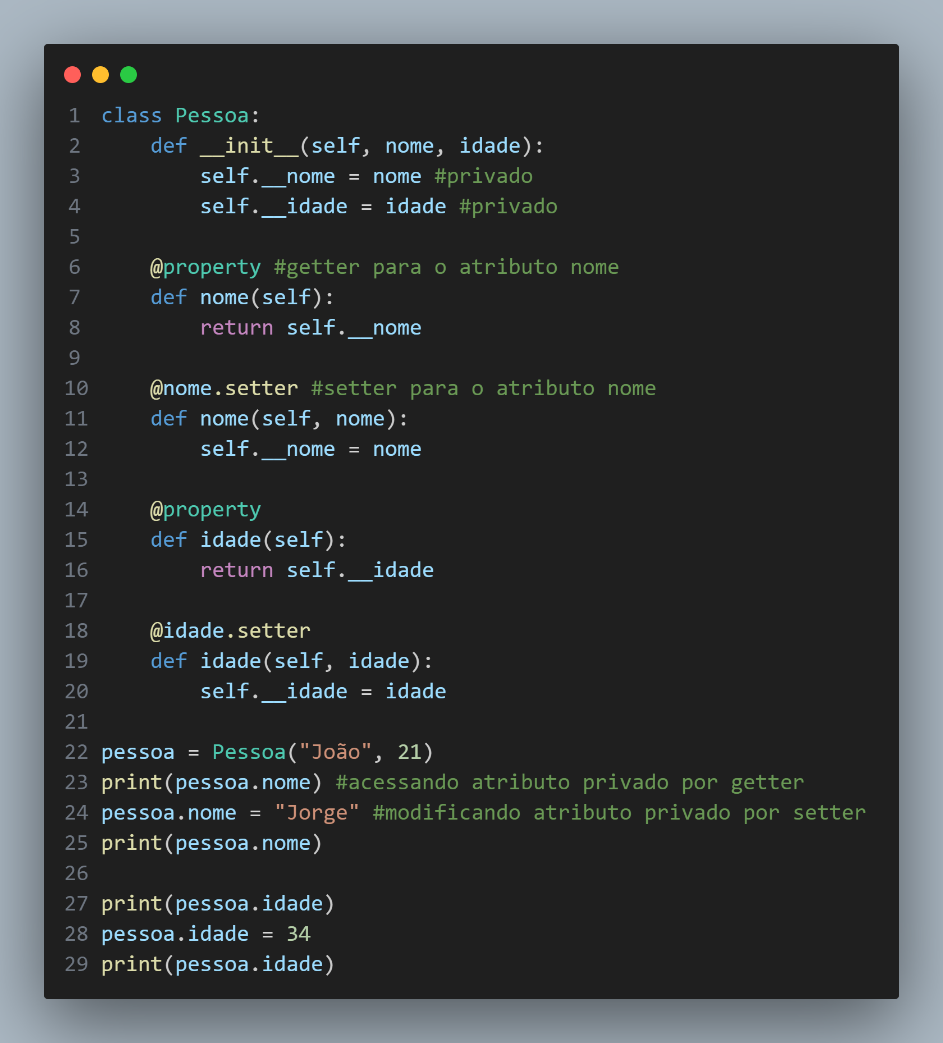
### Atributos e métodos privados



### Métodos de acesso

Getter: acessar o valor do atributo *@property*

Setter: modifica o valor do atributo *@(função).setter*



## Polimorfismo

Método de mesmo nome terem comportamentos diferentes, dependendo do objeto que são chamados.

### Polimorfismo em método

Um método com mesmo nome pode funcionar de forma distinta em diferentes classes.

### Polimorfismo por herança

Quando uma classe filha subscreve um método da classe pai para oferecer uma implementação específica.



### Duck Typing

O foco está no comportamento do objeto e não na sua classe

