



IN

# INFINITY SCHOOL

VISUAL ART CREATIVE CENTER

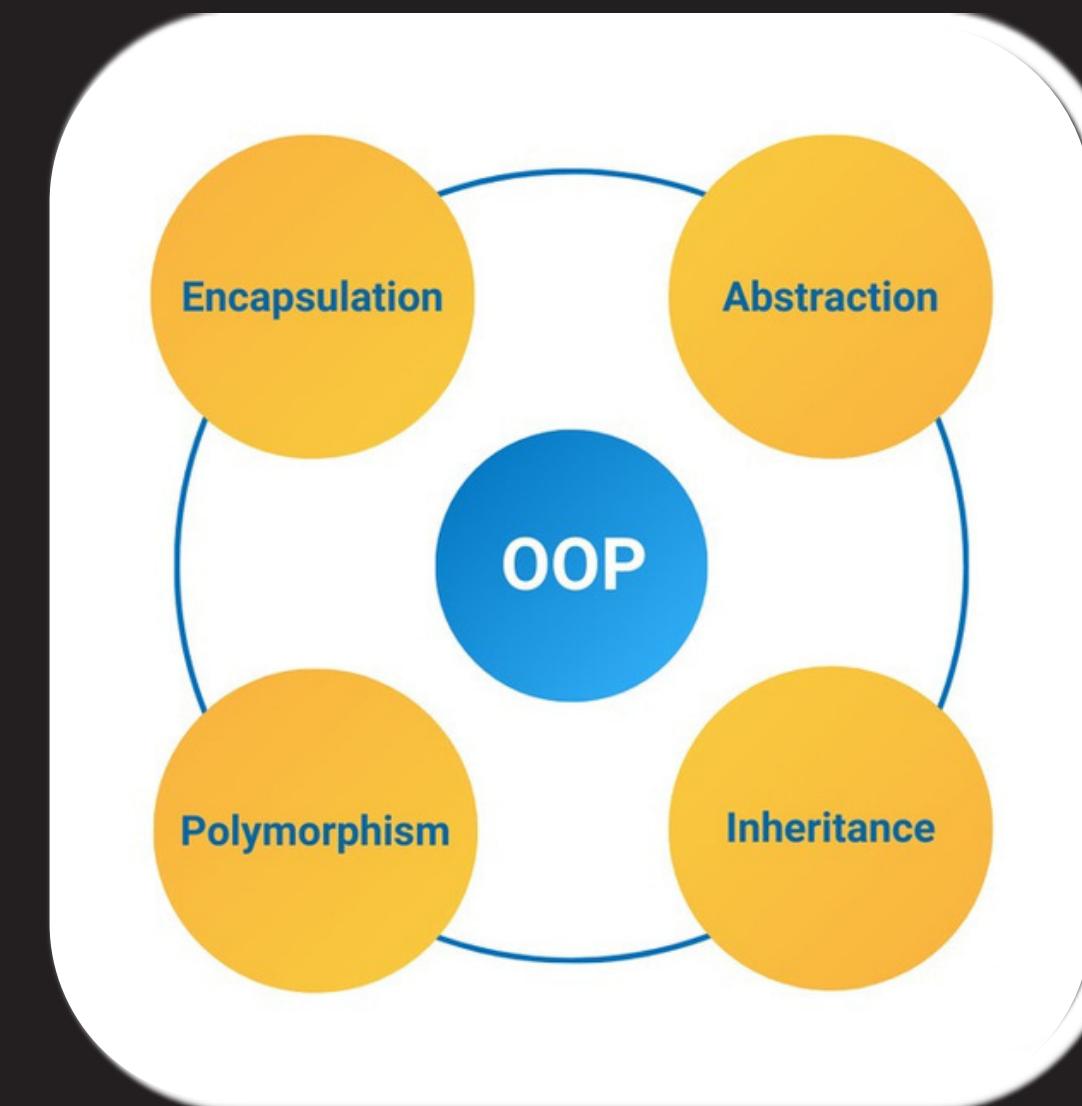
AULA 08 - POO I

# O QUE IREMOS APRENDER

- 01** PARADIGMA DE PROGRAMAÇÃO
- 02** PARADIGMA DE ORIENTAÇÃO A OBJETOS
- 03** ABSTRAÇÃO
- 04** OBJETOS
- 05** CONTEXTUALIZANDO
- 06** CLASSE
- 07** CLASSES E OBJETOS
- 08** MÃOS NO CÓDIGO

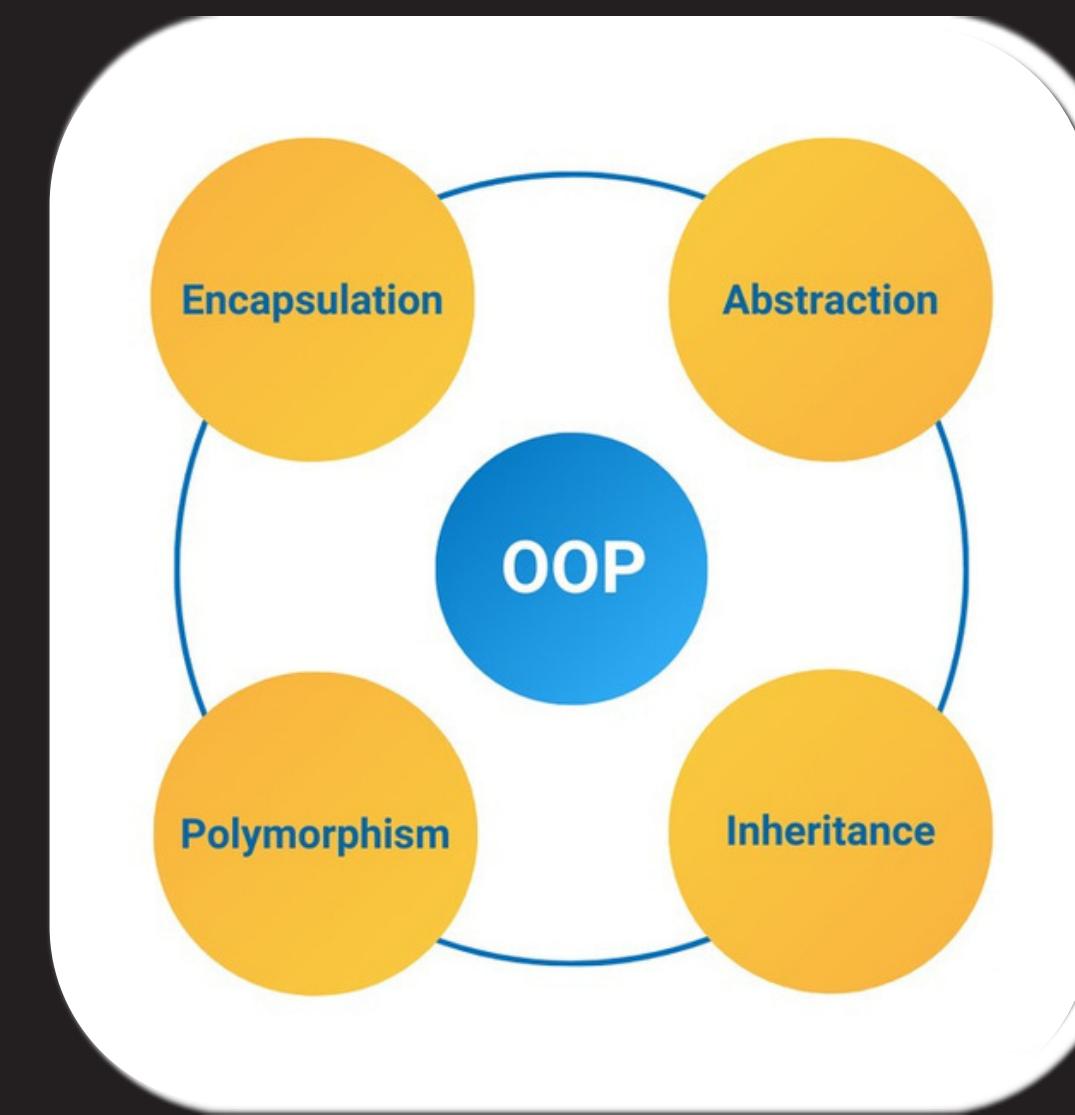
# POO (Programação Orientada a Objetos)

Quem desenvolve softwares sabe o quanto o trabalho pode ser desafiador. Afinal de contas, programar é trazer soluções para os problemas dos clientes utilizando algoritmos em uma linguagem de programação. Portanto, a programação orientada a objetos trouxe agilidade no desenvolvimento de software, reduzindo o tempo para identificar e corrigir erros de programação. Isso revitalizou o paradigma da programação como um todo. E desse assunto que iremos tratar nesta aula.



# Paradigma de Programação

Um paradigma de programação é uma "regra", uma estrutura, que uma linguagem de programação segue para podermos classificá-la. Em outras palavras, podemos entender um paradigma de programação em como determinada linguagem irá fornecer recursos para resolver determinado problema



# Paradigma de Programação

Um paradigma de programação fornece e determina a visão que o programador possui sobre a estruturação e execução do programa. Por exemplo, em **programação orientada a objetos**, os programadores podem abstrair um programa como uma coleção de **objetos** que interagem entre si.



# Paradigma de Programação

A POO foi criada para tentar aproximar o mundo real do mundo virtual.

A idéia fundamental é tentar simular o mundo real dentro do computador. Para isso, nada mais natural do que utilizar Objetos, afinal, nosso mundo é composto de objetos, certo?!



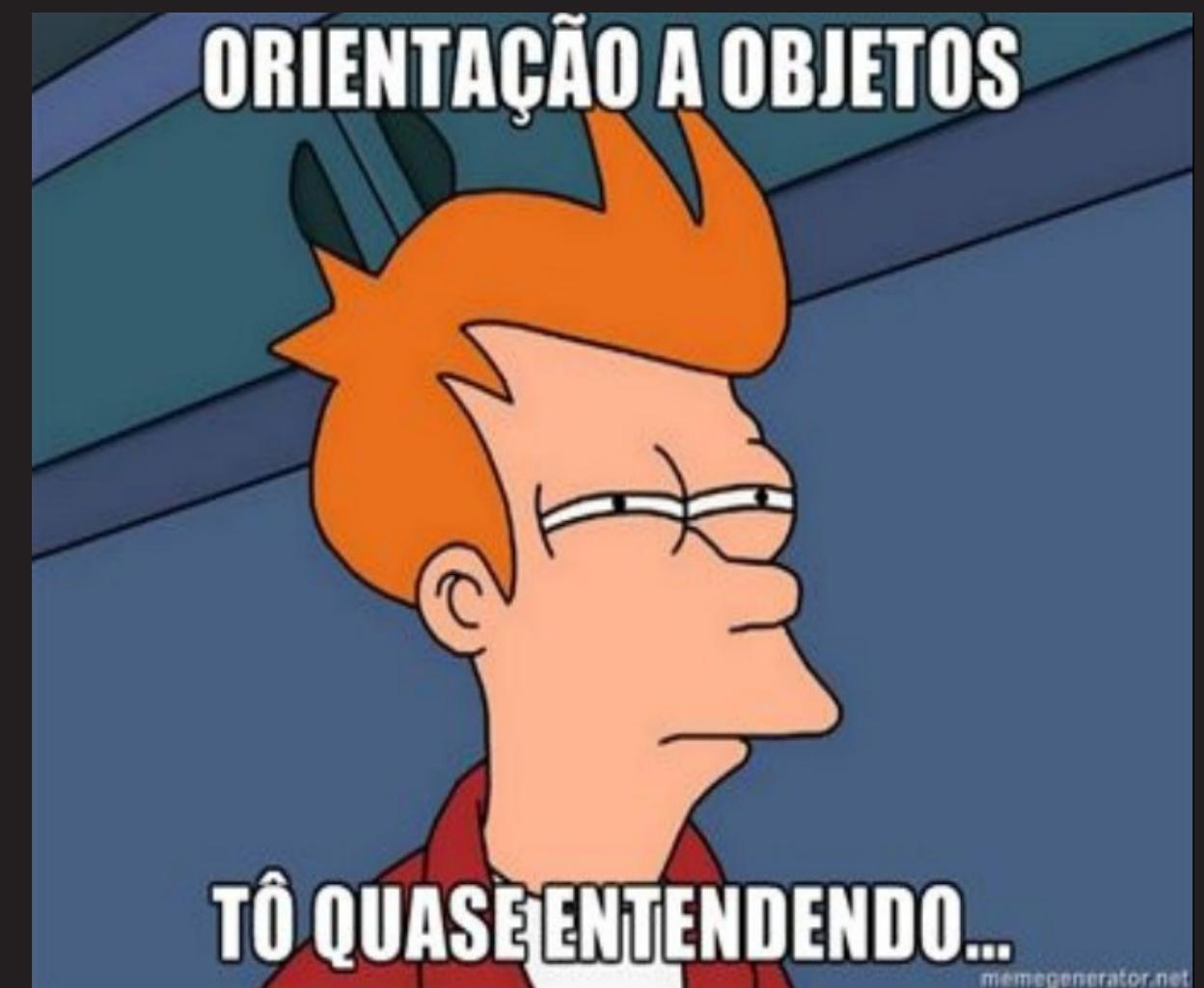
# Paradigma de Orientação a Objetos



Na POO o programador é responsável por moldar o mundo dos objetos, e explicar para estes objetos como eles devem interagir entre si.

# Abstração

Abstração envolve a criação de classes e objetos que representam entidades e conceitos do mundo real de forma simplificada. Isso permite que os programadores se concentrem apenas nos detalhes relevantes para o problema que estão resolvendo.



# Objetos - POO

Em POO tudo é **OBJETO**

Pense em um objeto como uma "super variável": O objeto armazena dados, também, pode-se fazer requisições a esse objeto, pedindo que ele execute operações.

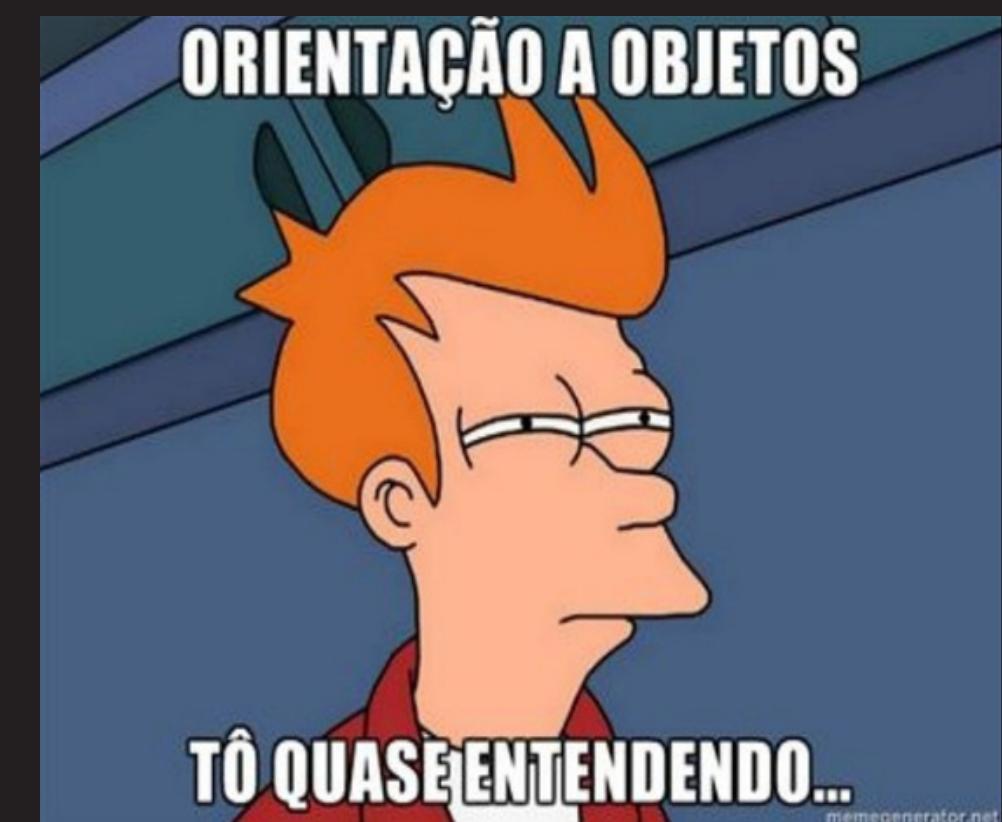
O exemplo abaixo é uma prova que usamos conceitos da programação orientada a objeto desde o começo de nossa jornada.

# Objetos - POO

EXEMPLO:

- ● ●

```
1 #Usando o método upper() da classe Str do python
2 my_name = 'Raama'.upper()
3 print(type(my_name)) #<class 'str'>
4
5 #Classe list
6 my_numbers = [4,3,2,1]
7
8 #Método append da class List do Python
9 my_numbers.append(0)
10 print(type(my_numbers)) #<class 'list'>
```



# Objetos - POO

Um programa é uma coleção de objetos dizendo uns aos outros o que fazer!

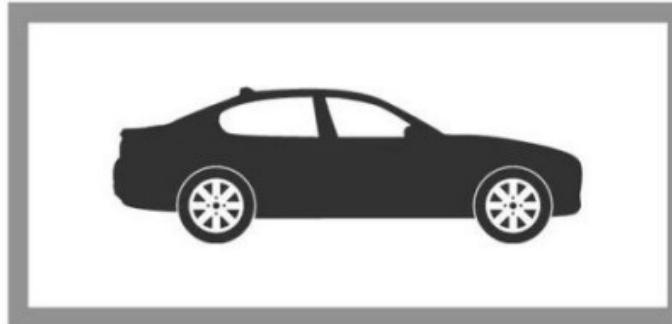
Para fazer uma requisição a um objeto envia-se uma mensagem a este objeto.

Uma mensagem é uma chamada de um método pertencente a um objeto em particular.

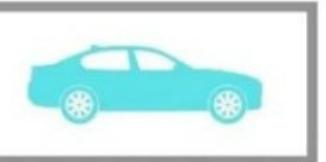
# Contextualizando POO

De uma **classe** se gera um **objeto**, esses objetos possuem dados e instruções sobre como manipular os dados que estão ligados à solução do problema.

**CLASSE CARRO**



**OBJETO**



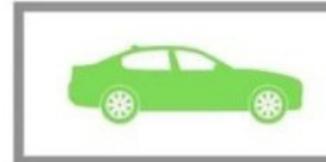
**CELTA**

**OBJETO**



**CORSA**

**OBJETO**

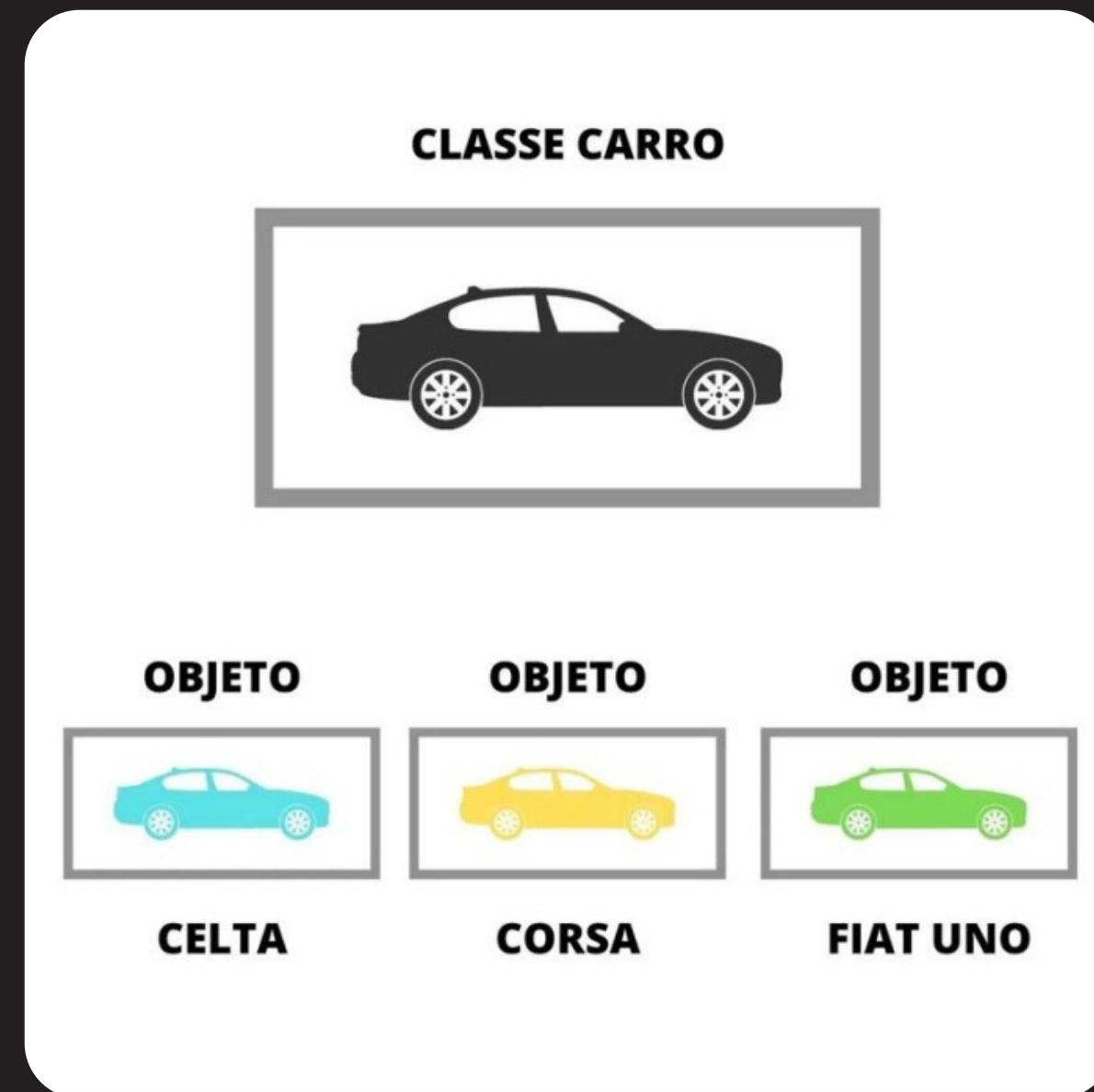


**FIAT UNO**

# Classe POO

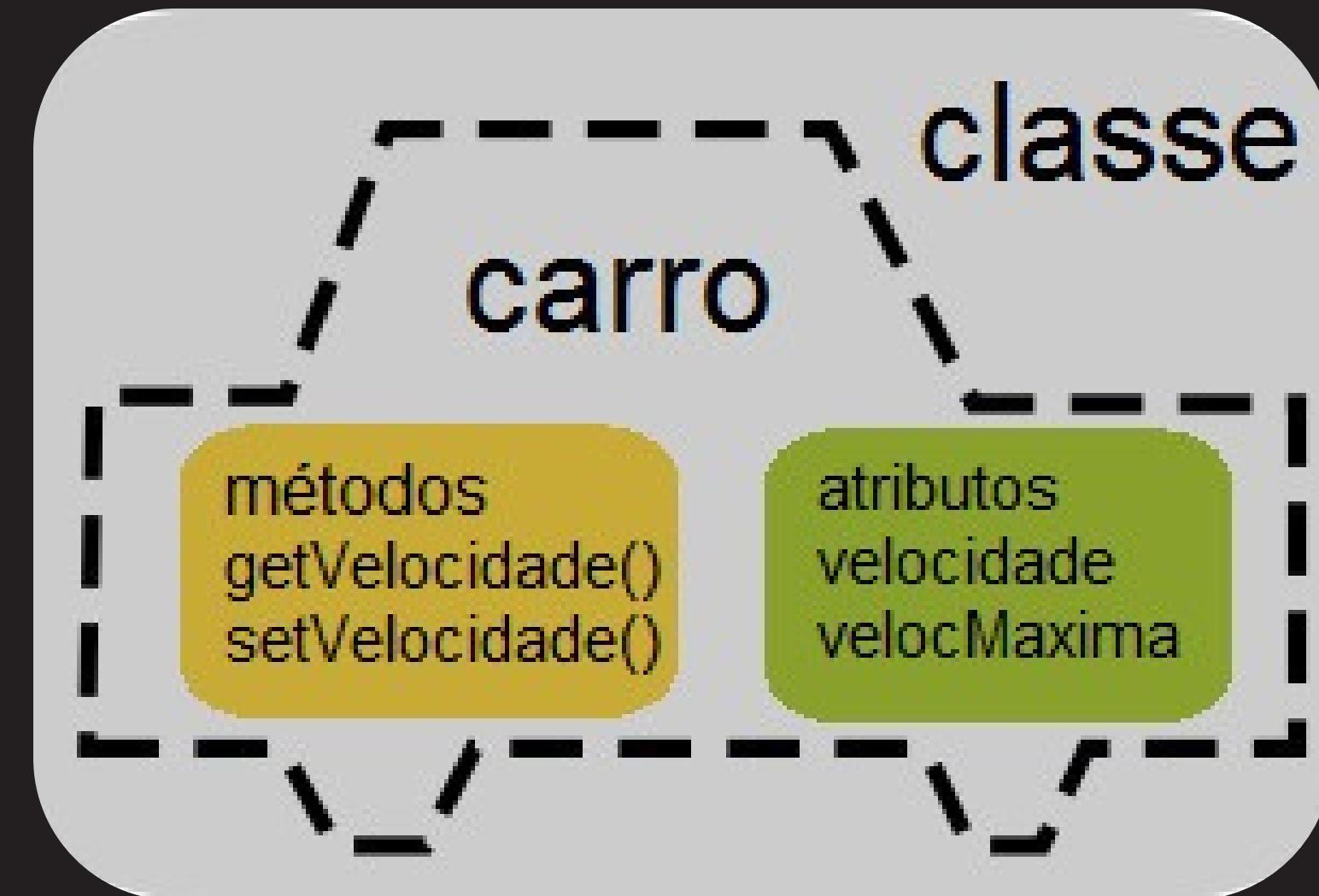
Podemos descrever um carro em termos de seus **atributos** que serão seus componentes ou características.

Conseguimos entender os **atributos** como as **variáveis** que pertencem àquela classe, ou seja, os **atributos** são todas as características daquela classe, no exemplo de um carro temos os atributos:



# Classe POO

```
● ● ●  
1 class Carro:  
2     def __init__(self, modelo, cor, ano):  
3         self.modelo = modelo  
4         self.cor = cor  
5         self.ano = ano  
6         self.velocidade = 0 # Velocidade inicial é zero  
7
```



# Classe POO

A classe pode ter quantos atributos quisermos e nós decidimos quais atributos farão parte, isso depende do que será importante para o nosso sistema e do nível de **abstração** que estamos fazendo da regra de negócio que o sistema irá atender.



# Classe POO

Podemos também descrever algumas ações que o carro faz (que serão seus métodos)

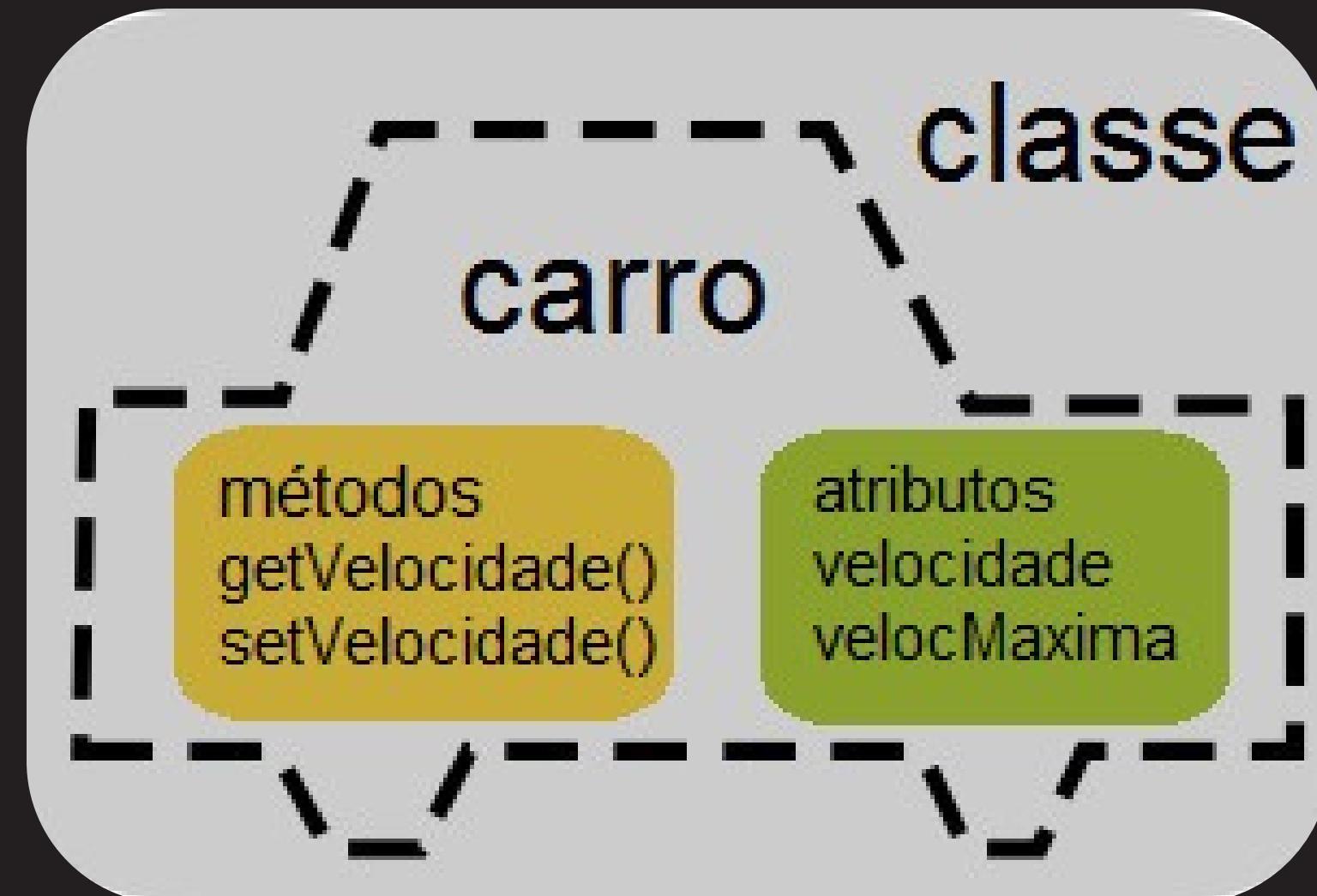
Conseguimos entender um **método** como uma função daquela classe. Essas funções podem retornar alguma coisa ou não.

No exemplo de uma string temos a função `.upper()` que é um **método** da classe `Str` que retorna a mesma string, maiúscula.

# Classe POO

Exemplos de métodos para Carro:

```
● ● ●  
1 def acelerar(self, velocidade):  
2     self.velocidade += velocidade  
3     print(f'O carro está agora a {self.velocidade} km/h.')  
4  
5 def frear(self, velocidade):  
6     if self.velocidade >= velocidade:  
7         self.velocidade -= velocidade  
8         print(f'O carro agora está a {self.velocidade} km/h após frear.')  
9     else:  
10        print('O carro já está parado.')
```



# Objetos POO

Vamos entender um pouco sobre os objetos

Já entendemos que as classes são abstrações que criamos de algo do mundo real, como fazemos para criar algo concreto dessas classes? Ou seja, criar os **OBJETOS**, que são instâncias da classe.

Para instanciar uma classe, utilizamos o método construtor.

# Classes e Objetos POO

Será o método construtor que irá construir algo concreto daquela classe: O objeto.

Em Python utilizamos `__init__` para criar o método construtor de determinada classe.



```
1 class Carro: #Primeira letra maiúscula
```

Utilizamos a palavra reservada `class` para criar uma classe no Python.  
Também definimos o nome da classe  
Sempre que criarmos uma classe, vamos definir o nome com a primeira letra maiúscula

# Classes e Objetos POO

O método construtor é composto por dois underline ou dunder-scores, essa função carregará parâmetros que farão referência aos dados atribuídos em nossos atributos. Podemos utilizar a palavra reservada **self** para referenciar cada **atributo** para objetos diferentes.

Sabendo que nossa classe é um modelo para gerar novos objetos, utilizamos um método construtor para passar esses dados a nossos objetos

```
● ● ●  
1 class Carro: #Primeira letra maiúscula  
2     def __init__(self, marca, modelo, ano, possui_4_rodas):  
3         self.marca = marca  
4         self.modelo = modelo  
5         self.ano = ano  
6         self.possui_4_rodas = possui_4_rodas  
7     #Método construtor da classe carro
```

# Classes e Objetos POO

Na mesma indentação do método construtor, vamos criar **métodos / funções**. Esses métodos trazem funcionalidades a nosso objeto, todo comportamento do nosso objeto é referenciado por self ela referencia nossa instância, por isso é padrão de estrutura utilizá-lo como primeiro parâmetro na criação de nossos métodos.

# Classes e Objetos POO



```
1 class Carro:  
2  
3     def __init__(self, marca, modelo, ano, possui_4_rodas):  
4         self.marca = marca  
5         self.modelo = modelo  
6         self.ano = ano  
7         self.possui_4_rodas = possui_4_rodas  
8  
9     def ligar(self):  
10        return f"O {self.modelo} está ligado"  
11  
12 orochinho = Carro("Renault", "Oroch", 2019, True)  
13  
14 print(orochinho.modelo)  
15 print(orochinho.ligar())
```

Agora é hora de criar nosso objeto/carro, para isso utilizamos **nome\_da\_classe()**



# Classes e Objetos POO



```
1 class Carro:  
2  
3     def __init__(self, marca, modelo):  
4         self.marca = marca  
5         self.modelo = modelo  
6         self.velocidade = 0 # Inicializar o atributo velocidade com 0  
7  
8     def acelerar(self):  
9         self.velocidade += 10  
10        print(f"{self.marca} {self.modelo} acelerando. Velocidade atual: {self.velocidade} km/h")  
11  
12    def frear(self):  
13        self.velocidade -= 5  
14        print(f"{self.marca} {self.modelo} freando. Velocidade atual: {self.velocidade} km/h")  
15  
16 # Criando um objeto da classe Carro  
17 meu_carro = Carro("Ford", "Mustang")  
18  
19 # Utilizando a abstração para interagir com o carro  
20 meu_carro.acelerar()  
21 meu_carro.frear()
```

# ATIVIDADE PRÁTICA 1

Crie um classe chamada cachorro com os atributos:  
nome, raça, idade

# ATIVIDADE PRÁTICA 2

Crie um classe chamada pessoa com os atributos: nome, idade, peso, gênero

# ATIVIDADE PRÁTICA 3

Crie uma classe Empresa que permita gerenciar funcionários. Os funcionários devem ter informações como nome, cargo e salário. A empresa deve ser capaz de adicionar, remover e listar funcionários.

# ATIVIDADE PRÁTICA 4

Crie uma classe Calculadora que tenha métodos para realizar operações matemáticas básicas (+ , - , \* , / ).

# ATIVIDADE PRÁTICA 5

Crie uma classe chamada Fatura , a classe Fatura deve incluir os seguintes atributos o nome do item; o preço unitário do item; quantidade de item a ser faturado; valor total da fatura; Sua classe deve ter um construtor que inicialize todos os atributos menos o valor total da fatura. Forneça um método chamado gerar\_fatura que calcula o valor da fatura (isto é, multiplicar a quantidade pelo preço por item).

# DESAFIO PRÁTICO

## Aplicativo de hotelaria

Crie uma classe Hotel que permita gerenciar funcionários, reservas e quartos de hotel. Os funcionários devem ter informações como nome, função e salário. O hotel deve ser capaz de receber reservas, atribuí-las a quartos e calcular a conta final.

# Material Complementar

- Exploração: Não tenha medo de explorar e testar diferentes códigos. A experimentação é uma grande aliada da aprendizagem.
- Perguntas: Faça perguntas, seja curioso! Entender o "porquê" das coisas ajuda a consolidar o conhecimento.
- Revisão: Revise o que aprendeu, tente explicar para si mesmo ou para outras pessoas. Ensinar é uma ótima forma de aprender.

Prática: A prática leva à perfeição. Quanto mais exercícios fizer, mais fácil será lembrar e entender os conceitos.



# SE LIGA NO CONTEÚDO DA PRÓXIMA AULA!

AULA 09 DE PYTHON:  
POO II - HERANÇA



**INFINITY SCHOOL**  
VISUAL ART CREATIVE CENTER

# Herança

A **herança** é um princípio próprio à programação orientada a objetos (POO) que permite criar uma nova classe a partir de uma já existente. Herança, também chamada de subclasses, provém da subclasse, da classe recém-criada que contém atributos e métodos da qual deriva. A principal vantagem da herança é a capacidade para definir novos atributos e métodos para a subclasse, que se somam aos atributos e métodos herdados.



```
    addEvent('on' + type, callback);
}

function decorate(event) {
  event = event || window.event;
  var target = event.target || event.srcElement;
  if (target && (target.getAttribute('action') || target.
    ga(function (tracker) {
      var linkerParam = tracker.get('linkerParam');
      document.cookie = '_shopify_ga=' + linkerParam + ';'
    }));
}

stener(window, 'load', function(){
  for (i=0; i < document.forms.length; i++) {
    if (document.forms[i].getAttribute('action') &&
        document.forms[i].getAttribute('method') == '/cart') {
      document.forms[i].onsubmit = decorate;
    }
  }
});
```

# Herança

Veja a seguir um exemplo de Herança em Python:

```
● ● ●  
1 class Animal:  
2  
3     def __init__(self, nome):  
4         self.nome = nome  
5  
6     def fazerSom(self):  
7         pass  
8  
9 class Cachorro(Animal):  
10    def fazerSom(self):  
11        return "Woof!"  
12  
13 class Gato(Animal):  
14    def fazerSom(self):  
15        return "Meow!"  
16  
17 rex = Cachorro("Rex")  
18 whiskers = Gato("whiskers")  
19  
20 print(rex.nome, "faz", rex.fazerSom())  
21 print(whiskers.nome, "faz", whiskers.fazerSom())
```





IN

INFINITY SCHOOL

VISUAL ART CREATIVE CENTER

AULA 08 - POO I