

# Programação Orientada a Objetos

As classes fornecem um meio de agrupar dados e funcionalidades. A criação de uma nova classe cria um novo tipo de objeto, permitindo que novas instâncias desse tipo sejam feitas. Cada instância de classe pode ter atributos anexados a ela para manter seu estado. As instâncias de classe também podem ter métodos (definidos por sua classe) para modificar seu estado.

Como vimos na aula teórica, as classes são os esqueletos dos objetos, o fato de criarmos uma classe, não cria automaticamente um objeto. Podemos comparar uma classe com uma planta de um projeto, o projeto de um carro, o esqueleto de todo objeto é uma classe.

#### Padrão de nomenclatura

para escrever classes, iremos adotar aqui a Pascal case. No padrão Pascal case, toda primeira letra de cada palavra é sempre maiúscula e as palavras não tem separadores de espaços, como no exemplo a seguir: Pessoas, ProjetoCarro, etc.

O primeiro método que todas as classes devem fornecer é o construtor. O construtor define a maneira como os objetos de dados são criados. Para criar um objeto Pessoa precisaremos fornecer 3 dados: o nome, o peso e a altura. Em Python, o método construtor é sempre chamado init (com dois underscores antes e dois depois de init) Também chamado Dunder init

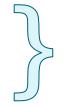
Sintaxe de Python para criar uma classe

class NomeClasse:

Sintaxe de Python para criar uma classe

class NomeClasse:

def \_\_init\_\_(self): Método Construtor



Vamos Criar nossa primeira Classe, usaremos como exemplo a Classe Pessoa. a classe Pessoa, tem nome, idade e peso como atributos e seus métodos são: falar, comer e dormir.

#### Pessoa

- nome
- idade
- peso
- falar()
- comer()
- dormir()

# CLASSES Sintaxe de Python para criar uma classe

#### class Pessoa:

```
def __init__(self, nome, peso, idade,comendo=False):
    self.nome=nome
    self.peso = peso
    self.idade = idade
```

Agora que nossa classe já foi criada, iremos alterá-la para implementar os métodos. Toda vez que chamarmos o método comer, por exemplo, o método deve mostrar o nome da pessoa e que ele foi comer.

Agora altere o método comer para mostrar também o que a pessoa está comendo, passando como parâmetro para o método o nome do alimento. Exemplo de saída:

João foi comer pipoca

Repita as alterações para implementar os métodos os demais métodos. Implemente mais uma alteração, se um pessoa já estiver comendo, por exemplo e você chamar o método comer, você deve informar que ela já está comendo.

Implemente os métodos parar, para cada métodos existente, exemplo, pararcomer()

## conta bancária

Crie uma classe que tenha os atributos, número da conta, saldo, status da conta( se ela está ativa ou não), nome do cliente, tipo da conta. e que possibilite ao cliente, depositar, sacar, verificar saldo e possibilidade de ativar a conta ou desativar a conta. para desativar uma conta, é necessário que o saldo este zerado.

## conta bancária

#### atributos:

- \*número
- \*saldo
- \*nome
- \*tipo
- \*status
- \*limite

#### métodos

- \*depositar
- \*sacar
- \*ativarconta
- \*verificarsaldo

## Data e hora em Python

Podemos importar um módulo chamado datetime e trabalhar com hora e data.

O módulo datetime tem muitos métodos para exibição de data e horas.

# Data e hora em Python import datetime

```
x = datetime.datetime.now()
print(x.hour)
print(x.date())
print(x.now())
print(x.year)
print(x.strftime("%A"))
```

## Data e hora em Python

```
from datetime import datetime
dias=["segunda", "Terça", "Quarta", "quinta",
"Sexta", "sábado", "domingo"]
x= datetime.now()
print(x.day)
print(x.month)
print(x.year)
print(x.weekday())
print(f'Hoje é dia {x.day} de {x.month} de {x.year} e
hojé é {dias[x.weekday()]}')
```

## Data e hora em Python

```
o método strftime() serve para formatarmos melhor a
impressão de datas e horas, conforme exemplo abaixo.
from datetime import datetime
agora = datetime.now()
data_formatada=agora.strftime("%d/%m/%Y %H:%M:%S")
print(data_formatada)
```

Como vimos na teoria de POO, a herança é a capacidade de uma classe herdar métodos e atributos de outras classes..

**Superclasses** ou **Classes Pai** são as classes que doam

**subclasses** ou **classes filhas**, são as classes que herdam atributos e métodos.

Mas como criar uma Superclasse?

Toda classe pode ser considerada uma superclasse, ou seja, não existe uma sintaxe específica para isso.

Crie uma Classe, Animal, com nome e cor como atributos e com um método Comer(), quando o método comer foi chamado, deverá printar na tela que o animal, foi comer.

```
class Animal():
    def __init__(self, nome , cor):
        self.nome=nome
        self.cor=cor

def comer(self):
    print( f' O {self.nome} foi comer...')
```

```
class Gato(Animal):
    def __init__(self, nome, cor):
        super().__init__(nome, cor)

def miar(self):
    print( f' O {self.nome} foi miando...')
```

```
class Gato(Animal):
    def __init__(self, nome, cor):
        super().__init__(nome, cor)

def miar(self):
    print( f' O {self.nome} foi miando...')
```

Complemente este exercício sobre herança, criando as classes cachorro, coelho e vaca, herdando da Classe Animal.

Complemente este exercício sobre herança, criando as classes cachorro, coelho e vaca, herdando da Classe Animal.

## exercícios 02

- 1. Crie uma classe chamada Ingresso, que possui um valor em reais e um método imprimeValor()
- Crie uma classe VIP, que herda de Ingresso e possui um valor adicional. Crie um método que retorne o valor do ingresso VIP (com o adicional incluído)

## exercícios 03

Crie uma classe chamada Forma, que possui os atributos area e perimetro.

- Implemente as subclasses Retangulo e Triangulo, que devem conter os métodos calculaArea e calculaPerimetro. A classe Triangulo deve ter também o atributo altura.
- No código de teste crie um objeto da classe
   Triangulo e outro da Classe Retangulo. calcule a área de cada um.

