Professor: Ruan Chaves Machado

Data do Material Didático Criado: 30/04/2025

Data da Aula: 13/05/2025

Tópicos e Temas abordados nessa aula:

* Classes em Python – O que é uma classe em Python
* Classes em Python – Atributos, como iniciar atributos de uma Classe
* Classes em Python – O que é um Objeto em Python?
* Classes em Python – Métodos, O que são Métodos e Funções em Python?
* Classes em Python – Herança, O que é a Herança de Classes em Python?

**Classes em Python – O que é uma classe em Python**

Classes proporcionam uma forma de organizar dados e funcionalidades juntos. Criar uma nova classe cria um novo “tipo” de objeto, permitindo que novas “instâncias” desse tipo sejam produzidas. Cada instância da classe pode ter atributos anexados a ela, para manter seu estado. Instâncias da classe também podem ter métodos (definidos pela classe) para modificar seu estado.

Em Python (e em outras linguagens orientadas a objetos), **uma classe é um molde** que define o **comportamento** e as **características** que os objetos daquele tipo terão.

**Analogia simples:** pense em uma classe como o projeto de uma casa. Ela define a estrutura (quantos quartos, banheiros etc.), mas não é a casa em si. Uma vez que você “instancia” esse projeto, você cria casas reais — ou seja, **objetos**.

Sem classes, programamos de forma “procedural”, ou seja, só com funções soltas e variáveis. Isso funciona, mas à medida que o sistema cresce, essa abordagem se torna difícil de manter.

**Com classes, podemos:**

* Organizar o código de forma mais lógica e modular
* Reutilizar estruturas de dados complexas
* Modelar o mundo real com mais clareza
* Trabalhar com herança e polimorfismo (outros conceitos da POO)

**Exemplos de criação de classes no Visual Studio Code (aula01\_classes.py)**

**Classes em Python – Atributos, como iniciar atributos de uma Classe**

A classes podem ter atributos, que são variáveis que pertencem à classe e são compartilhadas por todas as instâncias criadas a partir dessa classe. Os atributos representam as características que o objeto da classe possui.

Atributos são variáveis associadas a um objeto, ou seja, cada instância da classe possui seus próprios valores para esses atributos.

Em termos simples: são as características de um objeto.

**class** **Pessoa:**

**def** \_\_init\_\_**(**self**,** nome**,** idade**):**

self**.***nome* **=** nome # atributo

self**.***idade* **=** idade # atributo

p1 **=** Pessoa**(**"Ruan"**,** 25**)**

p2 **=** Pessoa**(**"Maria"**,** 30**)**

**print(**p1**.***nome***)** # Ruan

**print(**p2**.***nome***)** # Maria

**O método \_\_init\_\_**

O \_\_init\_\_ é um **método especial** (construtor) chamado automaticamente **quando criamos um objeto da classe**.

Ele serve para **definir e inicializar os atributos** do objeto.

**class** **Carro:**

**def** \_\_init\_\_**(**self**,** modelo**,** cor**):**

self**.***modelo* **=** modelo

self**.***cor* **=** cor

**Modelo e cor são parâmetros do construtor -** self.modelo **e** self.cor **são atributos da instância.**

**O papel do ‘self’**

O self é **obrigatório em todos os métodos de instância** (exceto métodos @staticmethod).

Ele representa **o próprio objeto** e permite acessar seus atributos e métodos de dentro da classe.

Em outras palavras: self é como dizer **“este objeto”**.

**Analogia com a vida real:** Imagine que cada pessoa carrega um crachá com seu nome.

**class** **Pessoa:**

**def** \_\_init\_\_**(**self**,** nome**):**

self**.***nome* **=** nome # "o nome que pertence a mim (self)"

Quando criamos:

p **=** Pessoa**(**"Ruan"**)**

O Python traduz internamente como:

Pessoa**.***\_\_init\_\_***(**p**,** "Ruan"**)**

E dentro do método, self agora é o objeto p.

**Classes em Python – O que é um Objeto em Python?**

Em Python, tudo é um objeto. Isso inclui números, strings, listas, [funções](https://hub.asimov.academy/tutorial/funcoes-sao-objetos-em-python-um-guia-para-iniciantes/) e, claro, os tipos de dados que você mesmo pode criar. Um objeto é uma coleção de dados (variáveis) e métodos (funções) que atuam nesses dados. Em outras palavras, um objeto representa uma entidade ou conceito, com suas propriedades e ações que podem ser realizadas.

**Analogia Simples:** Um objeto é uma instância de uma classe. Ele é a "coisa real" que você cria com base no molde (classe). Ou seja, se a classe é o projeto, o objeto é a construção real baseada nesse projeto.

**Ponto-chave:**

* Toda vez que você chama a classe com parênteses (ex: *p1 = Pessoa("Ruan"))*, o Python:
* Cria um novo espaço na memória
* Chama o método *\_\_init\_\_*
* Atribui valores aos atributos daquele novo objeto

**Exemplo:**

**class** **Pessoa:**

**def** \_\_init\_\_**(**self**,** nome**,** idade**):**

self**.***nome* **=** nome

self**.***idade* **=** idade

**def** falar**(**self**):**

**print(**f"{self**.***nome*} está falando."**)**

# Criando objetos

p1 **=** Pessoa**(**"Ruan"**,** 25**)**

p2 **=** Pessoa**(**"Maria"**,** 30**)**

p1**.***falar***()** # Ruan está falando.

p2**.***falar***()** # Maria está falando.

Aqui neste exemplo:

* *p1* e *p2* são objetos **distintos**.
* Ambos seguem o molde da classe ***Pessoa***.
* Cada um carrega seus ***próprios dados e comportamentos***.

**Analogia Didática: “Molde de Forminhas”**

Imagine que você tem um molde de forminhas de gelo (a classe).  
Cada cubo de gelo que você faz com esse molde é um objeto.

Apesar de todos seguirem o mesmo formato, cada cubo:

* Existe de forma independente
* Pode ter uma temperatura ou tamanho ligeiramente diferente
* Pode ser manipulado separadamente

**Como saber se algo é um objeto?**

Em Python, tudo é objeto — inclusive números, strings, listas e funções.

* **print(type(**5**))** # <class 'int'>
* **print(type(**"texto"**))** # <class 'str'>
* **print(type([**1**,** 2**,** 3**]))** # <class 'list'>

Inclusive todas as variáveis que você cria com MinhaClasse() são objetos.

**O que um objeto pode ter?**

|  |  |
| --- | --- |
| **Componente** | **O que faz** |
| Atributos | Guardam os dados do objeto (ex: nome, idade) |
| Métodos | Definem comportamentos (ex: falar, andar, somar) |
| Estado | Representa a situação atual do objeto (ex: “ativo”, “logado”) |

**Frase para memorizar:**

"Se a classe é o **molde**, o objeto é a ***realidade concreta***."

**Classes em Python – Métodos, O que são Métodos e Funções em Python?**

**Funções:**

Uma função é um bloco de código reutilizável que é projetado para realizar uma tarefa específica. Funções são úteis porque nos permitem dividir nosso programa em partes menores e mais gerenciáveis. Isso não apenas torna o código mais legível e organizado, mas também facilita a manutenção e o teste.

Em Python, definimos uma função usando a palavra-chave ***def***, seguida pelo nome da função e parênteses que podem conter parâmetros. Aqui está um exemplo simples de uma função que soma dois números:

**def** somar\_dois\_numeros**(**a**,** b**):**

**return** a **+** b

resultado **=** somar\_dois\_numeros**(**3**,** 5**)**

**print(**resultado**)** # Saída: 8

Neste exemplo, somar\_dois\_numeros é uma função que aceita dois argumentos e retorna a soma deles.

**Pontos para se observar:**

* Não está dentro de uma classe
* É um bloco solto, reutilizável

**Métodos:**

Um **método** é uma **função definida dentro de uma classe**. Ele age **em nome de um objeto**, ou seja, usa o self para acessar os **atributos e outros métodos** da instância.

**Exemplo:**

**class** **Pessoa:**

**def** saudacao**(**self**):**

**print(**"Olá, tudo bem?"**)**

Chamando:

p **=** Pessoa**()**

p**.***saudacao***()**

# Saida no terminal: Olá, tudo bem?

**Analogia prática:**

| **Conceito** | **Analogia** |
| --- | --- |
| **Função** | Uma **ferramenta solta** que você usa onde quiser (ex: uma chave de fenda) |
| **Método** | Uma **ferramenta específica de um objeto**, que só funciona com ele (ex: um botão do seu carro) |

**Classes em Python – Herança, O que é a Herança de Classes em Python?**

Herança é um recurso da POO (Programação Orientada a Objetos) que permite criar uma nova classe baseada em outra já existente.

Ou seja: você pode reaproveitar código, especializar comportamentos e evitar repetição.

Imagine que você tem uma classe genérica, como Animal. Vários tipos de animais compartilham comportamentos comuns: andar, comer, dormir.

Então, em vez de repetir tudo em cada classe (Cachorro, Gato, etc), você coloca o comum em Animal e herda esse comportamento nas outras.

**Exemplo:**

**class** **Animal:**

**def** \_\_init\_\_**(**self**,** nome**):**

self**.***nome* **=** nome

**def** comer**(**self**):**

**print(**f"{self**.***nome*} está comendo."**)**

# Herança simples

**class** **Cachorro(**Animal**):**

**def** latir**(**self**):**

**print(**f"{self**.***nome*} está latindo."**)**

Agora podemos criar um cachorro com os métodos **da própria classe** (latir) e **herdados** (comer):

dog **=** Cachorro**(**"Bolt"**)**

dog**.***comer***()** # herdado da classe Animal

dog**.***latir***()** # definido em Cachorro

**Como funciona a herança em Python?**

**Sintaxe da herança:**

**class** **SubClasse(**NomeDaClassePai**):**

# código específico da subclasse

**Termos:**

| **Termo** | **Significado** |
| --- | --- |
| Classe Pai | Também chamada de **superclasse** |
| Classe Filha | Também chamada de **subclasse** |
| Herança Simples | A classe filha herda de **uma única** classe pai |

**Usando *super()*:**

Se você quiser chamar o construtor da classe pai explicitamente dentro da filha, use ***super()***:

**class** **Animal:**

**def** \_\_init\_\_**(**self**,** nome**):**

self**.***nome* **=** nome

**class** **Gato(**Animal**):**

**def** \_\_init\_\_**(**self**,** nome**,** cor**):**

**super().***\_\_init\_\_***(**nome**)** # chama \_\_init\_\_ de Animal

self**.***cor* **=** cor

Isso garante que a parte "genérica" seja construída corretamente antes de você adicionar as partes específicas.

**Sobrescrita de métodos (override):**

Você pode reescrever um método da classe pai dentro da filha se quiser comportamento diferente:

**class** **Animal:**

**def** falar**(**self**):**

**print(**"Animal falando..."**)**

**class** **Gato(**Animal**):**

**def** falar**(**self**):**

**print(**"Miau!"**)**

a **=** Animal**()**

g **=** Gato**()**

a**.***falar***()** # Animal falando...

g**.***falar***()** # Miau!

**Resumo geral:**

| **Conceito** | **Explicação** |
| --- | --- |
| Herança | Mecanismo de **reaproveitar** e **especializar** classes |
| Classe pai | Classe base, mais genérica |
| Classe filha | Especializa a classe pai, herda seus métodos e atributos |
| super() | Acessa métodos da superclasse (comum no \_\_init\_\_) |

**Fontes para aprimoramento pessoal nos temas abordados:**

Classes em Python – O que é uma classe em Python

<https://docs.python.org/pt-br/3/tutorial/classes.html>

Classes em Python – Atributos, como iniciar atributos de uma Classe (Ingês)

<https://www.toptal.com/python/python-class-attributes-an-overly-thorough-guide>

Classes em Python – O que é um Objeto em Python?

<https://docs.python.org/pt-br/dev/reference/datamodel.html>