. Slide 1

Programação Orientada a Objetos (POO) para Iniciantes: Uma Abordagem Didática

Imagine que você está construindo um Lego.

- **A classe é como o manual de instruções:** Ela define o tipo de peça, sua cor, tamanho e como ela se encaixa com outras peças.
- **O objeto é a peça de Lego em si:** É uma instância física da peça, criada a partir do manual. Cada peça tem suas próprias características (cor, tamanho) e pode interagir com outras peças de acordo com as instruções.

Na POO, é a mesma coisa:

- **A classe é um modelo para criar objetos:** Ela define os atributos (dados) e métodos (comportamentos) que os objetos terão.
- **O objeto é uma instância da classe:** É um elemento individual criado a partir desse modelo, com seus próprios valores para os atributos e capaz de executar os métodos.

1. "Uma classe é um modelo para criar objetos."

- **O que significa:**
  - Imagine um cortador de biscoitos. Ele é um molde que define a forma de todos os biscoitos que você vai fazer. A classe é como esse cortador: um modelo que define como os objetos (os "biscoitos") serão.
  - Em termos de código, a classe é um "projeto" que define quais características (atributos) e comportamentos (métodos) os objetos terão.

2. "Pense nela como um molde que define um conjunto de características para os objetos."

- **O que significa:**
  - O molde do cortador de biscoitos define a forma do biscoito (redondo, quadrado, etc.). A classe define quais "características" os objetos terão.
  - Essas "características" são chamadas de atributos: variáveis que armazenam dados sobre o objeto. Por exemplo, a classe "Carro" pode ter atributos como "cor", "modelo" e "ano".

3. " Exemplo prático: Um 'Carro' pode ser uma classe. Cada carro específico (ex: 'Fusca', 'Ferrari') é um objeto dessa classe."

- **O que significa:**
  - "Carro" é um conceito geral. A classe "Carro" define o que todo carro tem em comum: rodas, motor, etc.
  - "Fusca" e "Ferrari" são carros específicos, ou seja, objetos da classe "Carro". Cada um tem suas próprias características: o Fusca pode ser azul e o Ferrari vermelho, mas ambos são "Carros".

4. "A classe define características que todos os carros têm, como cor e modelo."

- **O que significa:**
  - A classe "Carro" define que todo objeto "Carro" terá atributos como "cor" e "modelo".
  - Quando você cria um objeto "Fusca", você está criando um "Carro" específico, e pode definir os valores dos atributos: "cor = azul", "modelo = Fusca".
  - Do mesmo jeito, quando você cria um objeto "Ferrari" você esta criando outro "Carro" com suas caracteristicas únicas.

Em resumo:

- A classe é o "projeto"
- O objeto é a "construção" feita a partir desse projeto.

Slide 2

1. `class Carro:`

- `class`: Esta palavra-chave é essencial. Ela sinaliza ao Python que estamos definindo uma nova classe.
- `Carro`: Este é o nome da nossa classe. Por convenção, nomes de classes em Python começam com letra maiúscula.
- `:`: Os dois pontos indicam que o bloco de código indentado a seguir faz parte da definição da classe.

2. `cor = "Vermelho"`

- `cor`: Este é um atributo da classe `Carro`. Atributos são como variáveis que armazenam dados sobre os objetos criados a partir da classe.
- `"Vermelho"`: Este é o valor padrão do atributo `cor`. Isso significa que, se não especificarmos uma cor diferente ao criar um objeto `Carro`, ele será vermelho por padrão.

3. `modelo = "Sedan"`

- `modelo`: Outro atributo da classe `Carro`, representando o modelo do carro.
- `"Sedan"`: O valor padrão para o atributo `modelo`.

4. `ano = 2022`

- `ano`: Mais um atributo, representando o ano de fabricação do carro.
- `2022`: O valor padrão para o atributo `ano`.

5. "Os valores são iguais para todos os carros criados a partir dessa classe."

- **O que significa**:
  - Quando você cria um objeto a partir desta classe `Carro`, se você não informar o contrario, todos os objetos criados terão esses mesmos valores padrão para os atributos `cor`, `modelo` e `ano`.
  - Pense nisso como um "molde" que produz cópias idênticas, a menos que você especifique o contrário.

Em resumo:

Este código cria uma classe chamada `Carro` com três atributos: `cor`, `modelo` e `ano`. Os atributos são definidos diretamente dentro da classe, o que significa que todos os objetos criados a partir dessa classe terão os mesmos valores padrão para esses atributos.

Slide 4

`carro1 = Carro()` e `carro2 = Carro()`

- **Criação de Objetos:**
  - Estas linhas são o ponto crucial. Elas criam duas instâncias da classe `Carro`. Pense em "instância" como um carro físico real, construído a partir do "molde" definido pela classe `Carro`.
  - `carro1` e `carro2` são agora variáveis que armazenam referências a esses objetos recém-criados.
  - Cada objeto é independente. Mudanças em `carro1` não afetam `carro2`, e vice-versa.

2. Acessando os Atributos

- **`print(carro1.cor):`**
  - Esta linha acessa o atributo `cor` do objeto `carro1`. Usamos a notação de ponto (`.`) para acessar atributos de um objeto.
  - Como não especificamos uma cor diferente ao criar `carro1`, ele assume o valor padrão definido na classe: "Vermelho".
  - A função `print()` exibe esse valor na tela.
- **`print(carro2.modelo):`**
  - Semelhante ao anterior, esta linha acessa o atributo `modelo` do objeto `carro2`.
  - `carro2` também assume o valor padrão "Sedan" para o modelo.
- **`print(carro1.ano):`**
  - Aqui, acessamos o atributo `ano` do objeto `carro1`.
  - O valor exibido será o valor padrão definido na classe: 2022.

3. "Ambos os objetos carro1 e carro2 têm os mesmos valores para cor, modelo e ano."

- **Valores Padrão:**
    - Isso acontece porque, na definição da classe `Carro`, definimos valores padrão para os atributos `cor`, `modelo` e `ano`.
    - Quando criamos os objetos `carro1` e `carro2` sem especificar valores diferentes, eles herdam esses valores padrão da classe.
    - Para que os objetos tenham caracteristicas diferentes entre si, seria necessario que na criação do objeto fosse passado os parametros que diferencia um objeto do outro.

Em resumo:

Este código demonstra como criar objetos a partir de uma classe e como acessar seus atributos. Os objetos criados compartilham os mesmos valores de atributo padrão, pois esses valores foram definidos na classe e não foram substituídos durante a criação do objeto.

`carro1.cor = "Azul"`

Slide 5

- **Modificação Direta:**
    - Esta linha mostra a flexibilidade do Python. Você pode alterar o valor de um atributo de um objeto diretamente, sem precisar de métodos especiais (como "setters").
    - `carro1.cor` acessa o atributo `cor` do objeto `carro1`.
    - `= "Azul"` atribui o novo valor "Azul" a esse atributo.
    - A partir deste ponto, o objeto `carro1` terá a cor "Azul", substituindo o valor padrão "Vermelho" definido na classe.

2. `carro2.ano = 2020`

- **Modificação de Outro Atributo:**
    - Similarmente, esta linha modifica o atributo `ano` do objeto `carro2`.
    - O valor 2020 é atribuído ao atributo `ano`, substituindo o valor padrão 2022.

3. `print(carro1.cor)` e `print(carro2.ano)`

- **Verificação das Mudanças:**
    - Estas linhas imprimem os valores dos atributos modificados.
    - `print(carro1.cor)` exibe "Azul", confirmando que a cor de `carro1` foi alterada.
    - `print(carro2.ano)` exibe 2020, confirmando a alteração no ano de `carro2`.

4. "Agora carro1 tem a cor 'Azul' e carro2 tem o ano alterado para 2020."

- **Objetos Independentes:**

- Este ponto reforça que `carro1` e `carro2` são objetos independentes.
- A alteração na cor de `carro1` não afeta `carro2`, e a alteração no ano de `carro2` não afeta `carro1`.
- Cada objeto mantém seus próprios valores de atributo.

Em resumo:

Este código demonstra como modificar diretamente os atributos de objetos em Python. Essa flexibilidade permite que você personalize objetos individuais de acordo com suas necessidades, mesmo após a criação dos objetos.

OUTRO ASSUNTO AQUI, ANALISAR SOBRE CONSTRUTOR E METODO

Vamos detalhar o código que utiliza o método construtor `__init__` em Python:

1. **`def __init__(self, nome, idade):`**

- **`def __init__(...)`**:
  - Este é o método construtor. Ele é um método especial que é executado automaticamente quando um novo objeto é criado a partir da classe.
  - O nome `__init__` é fixo. Python reconhece esse nome e sabe que ele deve ser executado na criação do objeto.
- **`self`**:
  - O primeiro parâmetro de qualquer método de classe em Python é sempre `self`.
  - `self` é uma referência ao próprio objeto que está sendo criado. Ele permite que o método acesse e modifique os atributos do objeto.
- **`nome, idade`**:
  - Estes são os parâmetros do construtor. Eles representam os valores que serão usados para inicializar os atributos do objeto.
  - Neste caso, estamos definindo que cada objeto `Pessoa` terá um `nome` e uma `idade`.

2. **`self.nome = nome`** e **`self.idade = idade`**

- **Inicialização dos Atributos:**
  - Estas linhas atribuem os valores dos parâmetros `nome` e `idade` aos atributos correspondentes do objeto.
  - `self.nome = nome` significa: "Atribua o valor do parâmetro `nome` ao atributo `nome` deste objeto (`self`)."
  - `self.idade = idade` significa: "Atribua o valor do parâmetro `idade` ao atributo `idade` deste objeto (`self`)."
  - Desta forma, cada objeto `Pessoa` terá seus próprios valores de `nome` e `idade`.

3. **`joao = Pessoa("João", 30)`**

- **Criação do Objeto:**
  - Esta linha cria um novo objeto chamado `joao` da classe `Pessoa`.

- `Pessoa("João", 30)` chama o construtor `__init__` da classe `Pessoa`, passando os argumentos "João" e 30.
- O construtor inicializa os atributos `nome` e `idade` do objeto `joao` com esses valores.

4. **`print(joao.nome, joao.idade)`**

- **Acesso aos Atributos:**
  - Esta linha acessa os atributos `nome` e `idade` do objeto `joao` e os exibe na tela.
  - A saída será: `João 30`.

Em resumo:

- O método construtor `__init__` é uma ferramenta poderosa para inicializar objetos com valores específicos no momento da criação.
- `self` é essencial para que o metodo construtor possa criar os atributos dentro do objeto.
- Ele garante que cada objeto tenha seus próprios valores de atributo, tornando os objetos independentes e personalizados.

Slide 2

1. "Métodos são funções dentro de uma classe que realizam ações nos objetos."

- **O que significa:**
  - Métodos são como funções, mas eles pertencem a uma classe e operam nos objetos (instâncias) dessa classe.
  - Eles definem os comportamentos dos objetos, ou seja, as ações que os objetos podem realizar.
  - No exemplo, `apresentar()` é um método que define o comportamento de "apresentar-se" para um objeto `Pessoa`.

2. "Métodos sempre recebem self como primeiro argumento, representando o próprio objeto."

- **O que significa:**
  - `self` é um parâmetro especial que se refere ao próprio objeto que está chamando o método.
  - Quando você chama `maria.apresentar()`, `self` dentro do método `apresentar()` se refere ao objeto `maria`.
  - `self` permite que o método acesse e modifique os atributos do objeto.

3. **`class Pessoa:`**

- **Definição da Classe:**

- Define uma classe chamada `Pessoa`, que serve como um modelo para criar objetos do tipo "pessoa".

4. **`def __init__(self, nome):`**

- **Método Construtor:**
  - O construtor `__init__` inicializa os atributos do objeto quando ele é criado.
  - `self.nome = nome` atribui o valor do parâmetro `nome` ao atributo `nome` do objeto.

5. **`def apresentar(self):`**

- **Definição do Método `apresentar()`:**
  - Define um método chamado `apresentar()` que não recebe outros parâmetros além de `self`.
  - `print(f"Olá, meu nome é {self.nome}!")` usa um f-string para formatar uma mensagem que inclui o nome do objeto, acessado através de `self.nome`.

6. **`maria = Pessoa("Maria")`**

- **Criação do Objeto:**
  - Cria um objeto chamado `maria` da classe `Pessoa`, passando o argumento "Maria" para o construtor.

7. **`maria.apresentar()`**

- **Chamada do Método:**
  - Chama o método `apresentar()` do objeto `maria`.
  - Isso executa o código dentro do método, que imprime a mensagem "Olá, meu nome é Maria!".

Em resumo:

- Métodos são funções que pertencem a uma classe e operam nos objetos dessa classe.
- `self` é um parâmetro especial que permite que os métodos acessem e modifiquem os atributos do objeto.
- O código demonstra como definir um método em uma classe, criar um objeto e chamar o método do objeto.