

# Introdução à Programação Orientada a Objetos em Python (POO)

## *Objetivo:*

Apresentar os conceitos fundamentais da Programação Orientada a Objetos (POO) ter uma base teórica necessária antes de começar a implementação do projeto. É importante compreender a estrutura básica de uma classe, como definir atributos e métodos, e como criar objetos a partir de uma classe.

## *1 Conceitos Fundamentais da POO:*

### 1. **Classes:**

- Uma classe é um modelo ou "blueprint" que define as características e comportamentos de um objeto.
- Ela agrupa atributos (dados) e métodos (funções) que representam as propriedades e ações do objeto.

### 2. **Objetos:**

- Um objeto é uma instância de uma classe. Quando você cria um objeto, você está criando uma versão real do modelo definido pela classe.
- Cada objeto pode ter diferentes valores para os atributos definidos na classe.

### 3. **Atributos:**

- Atributos são variáveis que pertencem a uma classe ou a um objeto. Eles representam as características ou estados de um objeto.
- Exemplo: Cor, modelo, ano de um carro.

### 4. **Métodos:**

- Métodos são funções definidas dentro de uma classe que descrevem os comportamentos ou ações que um objeto pode realizar.
- Exemplo: Acelerar, frear, ligar o carro.

## Exemplo Prático: Classe `Carro`

Para ilustrar esses conceitos, vamos criar uma classe `Carro` que possui atributos como `cor`, `modelo` e `ano`, e métodos como `acelerar` e `frear`.

### Passo 1: Definindo a Classe `Carro`

```
python Copiar código  
  
class Carro:  
    def __init__(self, cor, modelo, ano):  
        self.cor = cor # Atributo que armazena a cor do carro  
        self.modelo = modelo # Atributo que armazena o modelo do carro  
        self.ano = ano # Atributo que armazena o ano de fabricação do carro
```

class `Carro`:

def `__init__`(self, cor, modelo, ano):

self.cor = cor # Atributo que armazena a cor do carro

self.modelo = modelo # Atributo que armazena o modelo do carro

self.ano = ano # Atributo que armazena o ano de fabricação do carro

- `__init__`: Este é o método inicializador (construtor) que é chamado automaticamente quando um novo objeto da classe `Carro` é criado. Ele recebe os parâmetros `cor`, `modelo` e `ano` para inicializar os atributos do carro.
- `self`: Referência ao próprio objeto que está sendo criado. É usado para acessar os atributos e métodos da classe dentro do próprio objeto.

## Passo 2: Adicionando Métodos à Classe `Carro`

```
python Copiar código

class Carro:
    def __init__(self, cor, modelo, ano):
        self.cor = cor
        self.modelo = modelo
        self.ano = ano

    def acelerar(self):
        print(f'O {self.modelo} está acelerando!')
```

class Carro:

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

self.cor = cor

self.modelo = modelo

self.ano = ano

def acelerar(self):

print(f'O {self.modelo} está acelerando!')


def frear(self):

print(f'O {self.modelo} está freando!')

- **acelerar**: Este método simula o comportamento de acelerar o carro. Ele usa o `self.modelo` para imprimir uma mensagem indicando qual carro está acelerando.
- **frear**: Este método simula o comportamento de frear o carro, imprimindo uma mensagem similar.

### Passo 3: Criando Objetos da Classe `Carro`

python

 Copiar código

```
# Criando objetos da classe Carro
meu_carro = Carro('Vermelho', 'Ferrari', 2022)
outro_carro = Carro('Azul', 'BMW', 2020)

# Chamando métodos nos objetos criados
meu_carro.acelerar() # Saída: O Ferrari está acelerando!
outro_carro.frear() # Saída: O BMW está freando!
```

# Criando objetos da classe `Carro`

`meu_carro = Carro('Vermelho', 'Ferrari', 2022)`

`outro_carro = Carro('Azul', 'BMW', 2020)`

# Chamando métodos nos objetos criados

`meu_carro.acelerar()` # Saída: O Ferrari está acelerando!

`outro_carro.frear()` # Saída: O BMW está freando!

- `meu_carro` e `outro_carro` são objetos instanciados a partir da classe `Carro`. Cada objeto tem seus próprios valores para os atributos `cor`, `modelo` e `ano`.
- Ao chamar os métodos `acelerar` e `frear`, os comportamentos definidos na classe são executados, mostrando a interação entre o objeto e seus métodos.

#### *Explicação dos Conceitos com Exemplos:*

- **Classe:** A classe `Carro` define como um carro deve ser representado, com atributos que descrevem suas propriedades (`cor`, `modelo`, `ano`) e métodos que descrevem seus comportamentos (`acelerar`, `frear`).
- **Objeto:** `meu_carro` e `outro_carro` são instâncias dessa classe, representando carros específicos com cores, modelos e anos diferentes.
- **Atributos:** Os atributos `cor`, `modelo` e `ano` são características que diferenciam um carro do outro.
- **Métodos:** Os métodos `acelerar` e `frear` permitem que esses carros realizem ações, como acelerar e frear.

### Aplicação no Projeto Final:

- **Classe `Avaliacao` e Classe `Restaurante`:** No projeto final, os alunos criarão classes como `Avaliacao` e `Restaurante`, onde os conceitos de POO serão aplicados de forma similar, mas em um contexto mais complexo e relacionado a um sistema de gestão de restaurantes.
- **Objetos:** Serão instanciados objetos dessas classes para manipular avaliações e restaurantes, similar ao exemplo do carro.
- **Métodos:** Os métodos serão usados para realizar operações como adicionar uma avaliação, listar restaurantes, alterar atributos, entre outros.

---

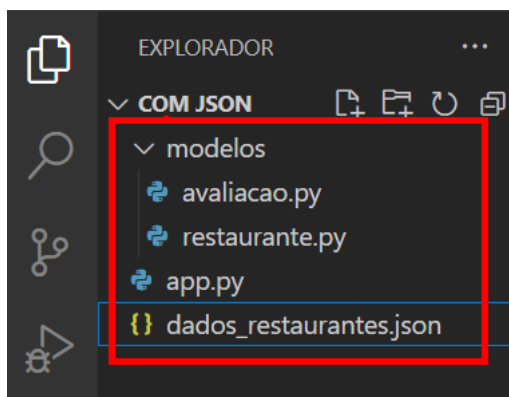
## Início (Projeto Restaurante Expresso versão POO)

### 1. Criar pasta `modelos` com os arquivos de classe:

- `avaliacao.py`
- `restaurante.py`


### 2. Na raiz criar arquivos:

- `app.py`
- `dados_restaurantes.json`



### 3. Criando a Classe `Avaliacao`

- **Objetivo:** Ensinar como criar uma classe simples que represente uma avaliação de restaurante.
- **Código:**

modelos >  avaliacao.py > ...

```
1 class Avaliacao:
2     def __init__(self, cliente, nota):
3         self._cliente = cliente # Nome do cliente que fez a avaliação
4         self._nota = nota # Nota dada pelo cliente
5
6     # Método para converter o objeto Avaliacao em um dicionário para salvar em JSON
7     def __dict__(self):
8         return {
9             'cliente': self._cliente,
10            'nota': self._nota
11        }
12
```

- **Explicação:**

- `__init__`: Método inicializador que define os atributos `cliente` e `nota`.
- `__dict__`: Método que converte os atributos do objeto em um dicionário, útil para salvar em JSON.

#### 4. Criando a Classe *Restaurante*

- **Objetivo:** Mostrar como criar uma classe que encapsula dados e comportamentos de um restaurante.
- **Código:**

```

modelos > restaurante.py > ...
1  from modelos.avaliacao import Avaliacao
2
3  class Restaurante:
4      # Lista que armazena todos os restaurantes cadastrados
5      restaurantes = []
6
7      def __init__(self, nome, categoria):
8          self._nome = nome.title() # Nome do restaurante, formatado com a primeira letra maiúscula
9          self._categoria = categoria.upper() # Categoria do restaurante, formatada em letras maiúsculas
10         self._ativo = False # Estado inicial do restaurante, definido como inativo
11         self._avaliacao = [] # Lista que armazena as avaliações do restaurante
12         Restaurante.restaurantes.append(self) # Adiciona o restaurante à lista global de restaurantes
13
14         def __str__(self):
15             # Retorna uma representação textual do restaurante, mostrando nome e categoria
16             return f'{self._nome} | {self._categoria}'
17
18         @classmethod
19         def listar_restaurantes(cls):
20             # Exibe a lista de todos os restaurantes cadastrados
21             print("\n")
22             print(f"{'Nome do restaurante'.ljust(25)} | {'Categoria'.ljust(25)} | {'Avaliação'.ljust(25)} | {'Status'}")
23             for restaurante in cls.restaurantes:
24                 print(f"{'restaurante._nome.ljust(25)} | {restaurante._categoria.ljust(25)} | {str(restaurante.
25                     media_avaliacoes).ljust(25)} | {restaurante.ativo}")
26
27         @property
28         def ativo(self):
29             # Retorna um símbolo visual representando se o restaurante está ativo ou inativo
30             return '☒' if self._ativo else '☐'
31
32         def alternar_estado(self):
33             # Alterna o estado do restaurante entre ativo e inativo
34             self._ativo = not self._ativo
35
36         def receber_avaliacao(self, cliente, nota):
37             # Adiciona uma nova avaliação ao restaurante, desde que a nota esteja entre 0 e 10
38             if 0 <= nota <= 10:
39                 avaliacao = Avaliacao(cliente, nota)
40                 self._avaliacao.append(avaliacao)
41             else:
42                 print("A nota deve estar entre 0 e 10.")
43
44         @property
45         def media_avaliacoes(self):
46             # Calcula e retorna a média das avaliações do restaurante
47             if not self._avaliacao:
48                 return '-'
49             soma_das_notas = sum(avaliacao._nota for avaliacao in self._avaliacao)
50             quantidade_de_notas = len(self._avaliacao)
51             media = round(soma_das_notas / quantidade_de_notas, 1)
52             return media

```

Símbolos para usar na classe (Copiar e colar) '☒' '☐'

- **Explicação:**

- **Atributos:** Explicar o uso de atributos como `_nome`, `_categoria`, `_ativo`, e `_avaliacao`.
- **Métodos:**

- `__init__`: Inicializa o restaurante e o adiciona à lista de restaurantes.
- `alternar_estado`: Alterna o estado de ativo para inativo e vice-versa.
- `receber_avaliacao`: Adiciona uma avaliação ao restaurante.
- `media_avaliacoes`: Calcula a média das avaliações do restaurante.
- `listar_restaurantes`: Lista todos os restaurantes cadastrados.
- **Propriedades:**
  - Explicar o uso de `@property` para definir métodos que agem como atributos.
- **Métodos de Classe:**
  - Explicar `@classmethod` para métodos que precisam acessar ou modificar a classe em si, não instâncias individuais.

## *5. Manipulando Dados com JSON e Construindo a Interface de Linha de Comando*

- **Objetivo:** Ensinar como carregar e salvar dados em arquivos JSON, simulando um banco de dados. Criar uma interface de linha de comando (CLI) para interagir com o sistema de restaurantes.

No arquivo `app.py` digitar:



```

app.py > ...
1  import os
2  import json
3  import sys
4  from modelos.restaurante import Restaurante
5  from modelos.avaliacao import Avaliacao
6
7  # Função para obter o caminho do diretório de dados
8  def get_data_dir():
9      if getattr(sys, 'frozen', False):
10         # Se estiver executando como um executável
11         return os.path.dirname(sys.executable)
12     else:
13         # Se estiver executando como script
14         return os.path.dirname(os.path.abspath(__file__))
15
16 # Nome do arquivo onde os dados dos restaurantes são armazenados
17 ARQUIVO_DADOS = os.path.join(get_data_dir(), 'dados_restaurantes.json')
18
19 # Função para carregar dados dos restaurantes a partir de um arquivo JSON
20 def carregar_dados():
21     try:
22         with open(ARQUIVO_DADOS, 'r', encoding='utf-8') as arquivo:
23             dados = json.load(arquivo)
24             Restaurante.restaurantes.clear() # Limpa a lista de restaurantes antes de carregar os novos dados
25             for restaurante_dados in dados:
26                 restaurante = Restaurante(
27                     restaurante_dados['nome'],
28                     restaurante_dados['categoria']
29                 )
30                 # Configura o estado ativo e as avaliações do restaurante
31                 restaurante._ativo = restaurante_dados['ativo']
32                 restaurante._avaliacao = [Avaliacao(**avaliacao) for avaliacao in restaurante_dados
33                                         ['avaliacao']]
34     except FileNotFoundError:
35         print(f"Arquivo de dados não encontrado. Criando um novo arquivo em {ARQUIVO_DADOS}")
36         salvar_dados() # Cria um arquivo vazio se não existir
37
38 # Função para salvar dados dos restaurantes em um arquivo JSON
39 def salvar_dados():
40     dados = []
41     for restaurante in Restaurante.restaurantes:
42         dados.append({
43             'nome': restaurante._nome,
44             'categoria': restaurante._categoria,
45             'ativo': restaurante._ativo,
46             'avaliacao': [avaliacao.__dict__() for avaliacao in restaurante._avaliacao]
47         })
48     with open(ARQUIVO_DADOS, 'w', encoding='utf-8') as arquivo:
49         json.dump(dados, arquivo, indent=4, ensure_ascii=False) # Salva os dados no arquivo com indentação
50         para melhor leitura

```

```

49
50 # Função principal do programa, que exibe o menu e executa as ações selecionadas pelo usuário
51 def main():
52     carregar_dados()
53
54     while True:
55         os.system('cls' if os.name == 'nt' else 'clear') # Limpa a tela antes de exibir o menu
56         print("== Restaurant Expresso ==")
57         print("\n1. Cadastrar restaurante")
58         print("2. Listar restaurantes")
59         print("3. Habilitar restaurante")
60         print("4. Avaliar restaurante")
61         print("5. Alterar restaurante")
62         print("6. Excluir restaurante")
63         print("7. Sair")
64
65         opcao = input("\nEscolha uma opção: ")
66
67         # Chama a função correspondente à opção escolhida
68         if opcao == '1':
69             cadastrar_restaurante()
70         elif opcao == '2':
71             listar_restaurantes()
72         elif opcao == '3':
73             habilitar_restaurante()
74         elif opcao == '4':
75             avaliar_restaurante()
76         elif opcao == '5':
77             alterar_restaurante()
78         elif opcao == '6':
79             excluir_restaurante()
80         elif opcao == '7':
81             salvar_dados() # Salva os dados antes de sair
82             print("\nDados salvos. Obrigado por usar o sistema. Até logo!")
83             break
84         else:
85             print("Opção inválida. Tente novamente.")
86
87         input("\nPressione Enter para continuar...")
88
89 # Função para cadastrar um novo restaurante
90 def cadastrar_restaurante():
91     nome = input("Digite o nome do restaurante: ")
92     categoria = input("Digite a categoria do restaurante: ")
93     novo_restaurante = Restaurante(nome, categoria) # Cria um novo objeto Restaurante
94     print(f"\nRestaurante {nome} cadastrado com sucesso!")
95     salvar_dados() # Salva os dados após o cadastro
96
97 # Função para listar todos os restaurantes cadastrados
98 def listar_restaurantes():
99     print("Lista de Restaurantes:")
100     Restaurante.listar_restaurantes()
101
102 # Função para habilitar ou desabilitar um restaurante
103 def habilitar_restaurante():
104     nome = input("Digite o nome do restaurante que deseja habilitar/desabilitar: ")
105     for restaurante in Restaurante.restaurantes:
106         if restaurante._nome.lower() == nome.lower():
107             restaurante.alternar_estado() # Altera o estado do restaurante (ativo/inativo)
108             print(f"Estado do restaurante {restaurante._nome} alterado para {restaurante.ativo}")
109             salvar_dados() # Salva os dados após a alteração
110             return
111     print("Restaurante não encontrado.")
112
113

```

```

112
113 # Função para adicionar uma avaliação a um restaurante
114 def avaliar_restaurante():
115     nome = input("Digite o nome do restaurante que deseja avaliar: ")
116     for restaurante in Restaurante.restaurantes:
117         if restaurante._nome.lower() == nome.lower():
118             cliente = input("Digite seu nome: ")
119             while True:
120                 try:
121                     nota = float(input("Digite a nota (de 0 a 10): "))
122                     if 0 <= nota <= 10:
123                         restaurante.receber_avaliacao(cliente, nota) # Adiciona a avaliação ao restaurante
124                         print("Avaliação registrada com sucesso!")
125                         salvar_dados() # Salva os dados após a avaliação
126                         return
127                     else:
128                         print("A nota deve estar entre 0 e 10.")
129                 except ValueError:
130                     print("Por favor, digite um número válido.")
131             print("Restaurante não encontrado.")
132
133 # Função para alterar as informações de um restaurante
134 def alterar_restaurante():
135     nome = input("Digite o nome do restaurante que deseja alterar: ")
136     for restaurante in Restaurante.restaurantes:
137         if restaurante._nome.lower() == nome.lower():
138             novo_nome = input(f"Digite o novo nome do restaurante (atual: {restaurante._nome}): ")
139             nova_categoria = input(f"Digite a nova categoria do restaurante (atual: {restaurante._categoria}): ")
140
141             # Atualiza o nome e a categoria do restaurante se forem fornecidos novos valores
142             if novo_nome:
143                 restaurante._nome = novo_nome.title()
144             if nova_categoria:
145                 restaurante._categoria = nova_categoria.upper()
146
147             print(f"Restaurante alterado com sucesso para: {restaurante}")
148             salvar_dados() # Salva os dados após a alteração
149             return
150     print("Restaurante não encontrado.")
151
152 # Função para excluir um restaurante da lista
153 def excluir_restaurante():
154     nome = input("Digite o nome do restaurante que deseja excluir: ")
155     for restaurante in Restaurante.restaurantes:
156         if restaurante._nome.lower() == nome.lower():
157             confirmacao = input(f"Tem certeza que deseja excluir o restaurante '{restaurante._nome}'? (S/N): ")
158             if confirmacao.lower() == 's':
159                 Restaurante.restaurantes.remove(restaurante) # Remove o restaurante da lista
160                 print(f"Restaurante '{restaurante._nome}' excluído com sucesso.")
161                 salvar_dados() # Salva os dados após a exclusão
162             else:
163                 print("Operação de exclusão cancelada.")
164             return
165     print("Restaurante não encontrado.")
166
167 # Verifica se o script está sendo executado diretamente e chama a função principal
168 if __name__ == '__main__':
169     main()
170

```

- **Explicação:**

- **carregar\_dados:** Carrega a lista de restaurantes a partir de um arquivo JSON.
- **salvar\_dados:** Salva a lista de restaurantes no arquivo JSON.
- **json.load** e **json.dump:** Explicar como manipular arquivos JSON em Python.
- **Menu Principal:** Apresentar um menu interativo e coletar a opção do usuário.
- **Funções CRUD:**
  - **Cadastrar:** Adiciona um novo restaurante à lista.
  - **Listar:** Exibe todos os restaurantes cadastrados.
  - **Habilitar:** Altera o estado ativo de um restaurante.
  - **Avaliar:** Permite ao usuário adicionar uma avaliação a um restaurante.
  - **Alterar:** Permite alterar os dados de um restaurante existente.
  - **Excluir:** Remove um restaurante da lista.
  -

## 6. Arquivo de Dados formato JSON

- **Objetivo:** Manter dados em arquivos JSON, simulando um banco de dados.
- No arquivo **dados\_restaurantes.json** colar:

```
[  
  {  
    "nome": "Sabor Caseiro",  
    "categoria": "COMIDA CASEIRA",  
    "ativo": true,  
    "avaliacao": []  
  },  
  {
```

```
"nome": "Cantina Italiana",
"categoria": "ITALIANA",
"ativo": false,
"avaliacao": []
},
{
  "nome": "Sushi Zen",
  "categoria": "JAPONESA",
  "ativo": false,
  "avaliacao": []
}
]
```

```
{ } dados_restaurantes.json > { } 2
1  [
2    {
3      "nome": "Sabor Caseiro",
4      "categoria": "COMIDA CASEIRA",
5      "ativo": true,
6      "avaliacao": []
7    },
8    {
9      "nome": "Cantina Italiana",
10     "categoria": "ITALIANA",
11     "ativo": false,
12     "avaliacao": []
13   },
14   {
15     "nome": "Sushi Zen",
16     "categoria": "JAPONESA",
17     "ativo": false,
18     "avaliacao": []
19   }
20 ]
```

## 7. Integração e Execução Completa

- **Objetivo:** Unir todas as partes do projeto e demonstrar como elas funcionam em conjunto.

- **Tarefa:**

- Execute o projeto completo.
- Testar como cada função interage com as classes e como os dados são manipulados e persistidos.