Classes e Objetos em Python

Representar entidades em Python requer o uso de classes e objetos. Entender esses conceitos envolve:

- **Classe** : Define um molde que especifica atributos (dados) e métodos (comportamentos) de uma entidade. No sistema acadêmico de frequência, modelar a classe `Aluno` abstrata características como nome e matrícula.
- **Objeto** : Cria uma instância específica de uma classe, contendo valores próprios para os atributos. Por exemplo, instanciar `Aluno("Ana", "2023001")` gera um objeto representando um aluno específico.

Exemplo completo

Estruturar uma Classe

Construir uma classe em Python utiliza a palavra-chave `class` , seguida por um nome (geralmente em CamelCase). A estrutura inclui:

- **Construtor** : implementa o método `__init__` para inicializar atributos da instância.
- **Atributos** : Declarar variáveis de instância (usando `self`) ou de classe (definidas diretamente na classe).
- **Métodos** : Definir funções que operam nos atributos, incluindo métodos de instância, de classe ou estáticos.

Exemplo básico:

```
class Exemplo:
    def __init__(self, valor):
```

```
        self.atributo_instancia = valor

    def executar_acao(self):
        return f"Valor: {self.atributo_instancia}"
```

Métodos Construtores

Definir um construtor com `__init__` permite inicializar ao criar atributos um objeto. O parâmetro `self` refere-se à instância sendo criada. No sistema de frequência, a classe `Aluno` usa um construtor para definir `nome`, `matricula` e `presencas`:

```
class Aluno:
    def __init__(self, nome, matricula):
        self.nome = nome
        self.matricula = matricula
        self.presencas = []
```

Exemplo funcional:

```
aluno = Aluno("Ana", "2023001")
print(aluno.nome)  # Saída: Ana
print(aluno.matricula)  # Saída: 2023001
print(aluno.presencas)  # Saída: []
```

Personalizar o construtor permite inicializar objetos com diferentes configurações. Por exemplo, adicione um parâmetro opcional para presença inicial:

```
class Aluno:
    def __init__(self, nome, matricula, presencas_iniciais=None):
        self.nome = nome
        self.matricula = matricula
        self.presencas = presencas_iniciais if presencas_iniciais is not No
```

Exemplo funcional:

```
aluno1 = Aluno("Bruno", "2023002")
aluno2 = Aluno("Clara", "2023003", ["2025-07-30"])
print(aluno1.presencas)  # Saída: []
print(aluno2.presencas)  # Saída: ["2025-07-30"]
```

Variáveis de Instância

Declarar variáveis com `self` associação como uma instância específica. Cada objeto possui seus próprios valores. No sistema de frequência, `nome`, `matricula` e `presencas` são variáveis de instância:

```
class Aluno:
    def __init__(self, nome, matricula):
        self.nome = nome
        self.matricula = matricula
        self.presencas = []

    def adicionar_presenca(self, data):
        self.presencas.append(data)
```

Exemplo funcional:

```
aluno1 = Aluno("Ana", "2023001")
aluno2 = Aluno("Bruno", "2023002")
aluno1.adicionar_presenca("2025-07-30")
print(aluno1.presencas)  # Saída: ["2025-07-30"]
print(aluno2.presencas)  # Saída: []
```

Variáveis de Classe

Definir variáveis diretamente na classe criada atributos compartilhados por todas as instâncias. Nenhum sistema de frequência, contar o total de alunos requer uma variável de classe:

```
class Aluno:
    total_alunos = 0  # Variável de classe

    def __init__(self, nome, matricula):
        self.nome = nome
        self.matricula = matricula
        self.presencas = []
        Aluno.total_alunos += 1
```

Exemplo funcional:

```
aluno1 = Aluno("Ana", "2023001")
aluno2 = Aluno("Bruno", "2023002")
print(Aluno.total_alunos)  # Saída: 2
print(aluno1.total_alunos)  # Saída: 2
```

Métodos de Instância

Definir métodos que `self` permitem operar nos atributos da instância. Sem sistema de frequência, o método `adicionar_presenca` modifica a lista `presencas` do objeto:

```
class Aluno:
    def __init__(self, nome, matricula):
        self.nome = nome
        self.matricula = matricula
        self.presencas = []

    def adicionar_presenca(self, data):
        if data not in self.presencas:
            self.presencas.append(data)
            return f"Presença registrada para {self.nome} em {data}."
        return f"Presença já registrada para {self.nome} em {data}."
```

Exemplo funcional:

```
aluno = Aluno("Ana", "2023001")
print(aluno.adicionar_presenca("2025-07-30"))  # Saída: Presença registrada
print(aluno.adicionar_presenca("2025-07-30"))  # Saída: Presença já registr
print(aluno.presencas)  # Saída: ["2025-07-30"]
```

Métodos de Classe

Usar o decorador `@classmethod` define métodos que operam na classe, recebendo `cls` como primeiro parâmetro. No sistema de frequência, consultar o total de alunos pode ser implementado assim:

```
class Aluno:
    total_alunos = 0

    def __init__(self, nome, matricula):
        self.nome = nome
        self.matricula = matricula
        self.presencas = []
        Aluno.total_alunos += 1

    @classmethod
    def obter_total_alunos(cls):
        return f"Total de alunos cadastrados: {cls.total_alunos}"
```

Exemplo funcional:

```
aluno1 = Aluno("Ana", "2023001")
aluno2 = Aluno("Bruno", "2023002")
print(Aluno.obter_total_alunos())  # Saída: Total de alunos cadastrados: 2
```

Métodos Estáticos

Definir métodos de `@staticmethod` criação de funções que não dependam de instâncias ou de classe. Nenhum sistema de frequência, validar o formato de um dado pode ser um método estático:

```
class Aluno:
    def __init__(self, nome, matricula):
        self.nome = nome
        self.matricula = matricula
        self.presencas = []

    @staticmethod
    def validar_data(data):
        from datetime import datetime
        try:
            datetime.strptime(data, "%Y-%m-%d")
            return True
        except ValueError:
            return False
```

Exemplo funcional:

```
print(Aluno.validar_data("2025-07-30"))  # Saída: True
print(Aluno.validar_data("invalido"))  # Saída: False
```

Aplicar Abstração

O resumo consiste em modelar apenas os aspectos relevantes de uma entidade. Sem sistema de frequência:

- **Aluno** : Representar com `nome` , `matricula` e `presencas` , ignorando detalhes como endereço.
- **Professor** : Definir com `nome` , `id_professor` e o método `registrar_presenca` .
- **Turma** : Estruturar com `codigo` , `professor` e uma lista de `alunos` .

Exemplo funcional no sistema de frequência:

```python
class Aluno:
    total_alunos = 0

    def __init__(self, nome, matricula):
        self.nome = nome
        self.matricula = matricula
        self.presencas = []
        Aluno.total_alunos += 1

    def adicionar_presenca(self, data):
        if data not in self.presencas:
            self.presencas.append(data)
            return f"Presença registrada para {self.nome} em {data}."
        return f"Presença já registrada para {self.nome} em {data}."

    @classmethod
    def obter_total_alunos(cls):
        return f"Total de alunos: {cls.total_alunos}"

    @staticmethod
    def validar_data(data):
        from datetime import datetime
        try:
            datetime.strptime(data, "%Y-%m-%d")
            return True
        except ValueError:
            return False

class Professor:
    def __init__(self, nome, id_professor):
        self.nome = nome
        self.id_professor = id_professor

    def registrar_presenca(self, aluno, turma, data):
        if Aluno.validar_data(data) and aluno in turma.alunos:
            return aluno.adicionar_presenca(data)
        return f"Erro: Data inválida ou aluno {aluno.nome} não matriculado

class Turma:
    def __init__(self, codigo, professor):
        self.codigo = codigo
        self.professor = professor
        self.alunos = []

    def matricular_aluno(self, aluno):
        if aluno not in self.alunos:
            self.alunos.append(aluno)
            return f"Aluno {aluno.nome} matriculado na turma {self.codigo}.
        return f"Aluno {aluno.nome} já matriculado na turma {self.codigo}."

# Testar o sistema
professor = Professor("Dr. Carlos", "P001")
```

```
turma = Turma("T101", professor)
aluno = Aluno("Ana", "2023001")
print(turma.matricular_aluno(aluno))  # Saída: Aluno Ana matriculado na tur
print(professor.registrar_presenca(aluno, turma, "2025-07-30"))  # Saída: P
print(Aluno.obter_total_alunos())  # Saída: Total de alunos: 1
print(Aluno.validar_data("2025-07-30"))  # Saída: True
```

Conceitos-chave

Classes e objetos de compreensão em Python envolvem:

1. Definir classes como moldes com atributos e métodos.
2. Crie objetos como instâncias com valores específicos.
3. Usar construtores para inicializar atributos.
4. Declarar variáveis de instância para dados específicos e de classe para dados compartilhados.
5. Implemente métodos de instância, de classe e estáticos para diferentes funcionalidades.
6. Aplicar abstração para focar no essencial.
7. Explorar encapsulamento, herança e polimorfismo para designs robustos.

Exercício

Pegue o código no exemplo e aprimore:

1. Aluno: remove um dado da lista de frequências
2. Professor: remover uma presença
3. Turma: remover aluno da turma, substituir professor, retornar o aluno que tem mais frequência