Apostila 5: Python - Programação Orientada a Objetos em Python

1. Introdução à Programação Orientada a Objetos (POO)

A Programação Orientada a Objetos (POO) é um paradigma de programação baseado em objetos, que combina dados e funcionalidades. Os principais conceitos da POO são:

- **Classes e Objetos**
- **Atributos e Métodos**
- **Encapsulamento**
- **Herança**
- **Polimorfismo**

2. Classes e Objetos

Uma classe é um modelo para criar objetos. Um objeto é uma instância de uma classe.

```
class Pessoa:
    def __init__(self, nome, idade):
        self.nome = nome
        self.idade = idade

    def apresentar(self):
        print(f"Olá, meu nome é {self.nome} e eu tenho {self.idade} anos.")

# Criando um objeto
pessoa1 = Pessoa("Alice", 30)
pessoa1.apresentar()
```

3. Atributos e Métodos

Atributos são características do objeto e métodos são funções dentro da classe que definem comportamentos.

```
class Carro:
    def __init__(self, modelo, ano):
```

```
        self.modelo = modelo
        self.ano = ano

    def detalhes(self):
        return f"Modelo: {self.modelo}, Ano: {self.ano}"

# Criando um objeto
meu_carro = Carro("Fusca", 1975)
print(meu_carro.detalhes())
```

4. Encapsulamento

O encapsulamento protege os dados dentro da classe, restringindo o acesso direto a determinados atributos.

```
class ContaBancaria:
    def __init__(self, saldo):
        self.__saldo = saldo  # Atributo privado

    def depositar(self, valor):
        self.__saldo += valor
        print(f"Depósito realizado. Saldo atual: R$ {self.__saldo}")

    def sacar(self, valor):
        if valor <= self.__saldo:
            self.__saldo -= valor
            print(f"Saque realizado. Saldo restante: R$ {self.__saldo}")
        else:
            print("Saldo insuficiente!")

# Criando uma conta
conta = ContaBancaria(1000)
conta.depositar(500)
conta.sacar(200)
```

5. Herança

A herança permite que uma classe herde atributos e métodos de outra classe.

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

```
    def falar(self):
        pass  # Método abstrato

class Cachorro(Animal):
    def falar(self):
        return "Au au!"

class Gato(Animal):
    def falar(self):
        return "Miau!"

# Criando objetos
cachorro = Cachorro("Rex")
gato = Gato("Mimi")
print(cachorro.falar())
print(gato.falar())
```

6. Polimorfismo

O polimorfismo permite que métodos com o mesmo nome tenham comportamentos diferentes em classes diferentes.

```
class Forma:
    def area(self):
        pass

class Quadrado(Forma):
    def __init__(self, lado):
        self.lado = lado

    def area(self):
        return self.lado ** 2

class Circulo(Forma):
    def __init__(self, raio):
        self.raio = raio

    def area(self):
        return 3.14 * (self.raio ** 2)

# Criando objetos
quadrado = Quadrado(4)
circulo = Circulo(3)
print(quadrado.area())
```

```
print(circulo.area())
```

7. Exercícios

1. Crie uma classe `Aluno` com os atributos `nome` e `nota` e um método que retorna se o aluno foi aprovado (nota >= 7).
2. Desenvolva uma classe `Livro` que tenha os atributos `título` e `autor`, além de um método para exibir os detalhes do livro.
3. Implemente uma classe `Funcionario` com atributos `nome` e `salario`, e uma subclasse `Gerente` que tenha um método adicional para aumentar o salário.
4. Crie uma classe `Veiculo` e subclasses `Carro` e `Moto`, onde cada uma tenha um método `tipo()` que retorna "Carro" ou "Moto" respectivamente.

Essa apostila cobre os conceitos fundamentais de POO em Python. Caso precise de ajustes ou mais detalhes, estou à disposição!