RESPOSTAS DAS ATIVIDADES DE PYTHON

Associação, Agregação, Composição, Encapsulamento, Polimorfismo e Abstração

1. Associação, Agregação e Composição

```
# 1. COMPOSIÇÃO → Motor pertence ao Carro
class Motor:
    def __init__(self, potencia):
        self.potencia = potencia

class Carro:
    def __init__(self, modelo, potencia_motor):
        self.modelo = modelo
        self.motor = Motor(potencia_motor)
    def info(self):
        print(f"Carro: {self.modelo}, Motor: {self.motor.potencia}cv")

# 2. AGREGAÇÃO
class Professor:
    def __init__(self, nome):
        self.nome = nome

class Universidade:
    def __init__(self, nome):
        self.nome = nome
        self.professores = []
    def adicionar_professor(self, professor):
        self.professores.append(professor)

# 3. ASSOCIAÇÃO
class Autor:
    def __init__(self, nome):
        self.nome = nome

class Livro:
    def __init__(self, titulo, autor):
        self.titulo = titulo
        self.autor = autor
```

2. Encapsulamento

```
# 1. Conta Bancária com saldo privado
class ContaBancaria:
    def __init__(self):
        self.__saldo = 0
    def depositar(self, valor):
        self.__saldo += valor
    def sacar(self, valor):
        if valor <= self.__saldo:
            self.__saldo -= valor
    def ver_saldo(self):
        return self.__saldo

# 2. Pessoa com atributo protegido
class Pessoa:
    def __init__(self, nome, anoNasceu):
        self.nome = nome
        self._anoNasceu = anoNasceu

# 3. Conta com @property
class ContaBancaria2:
    def __init__(self):
        self.__saldo = 0
    @property
    def saldo(self):
        return self.__saldo
    @saldo.setter
    def saldo(self, valor):
        if valor >= 0:
            self.__saldo = valor

# 4. Produto com @property
class Produto:
    def __init__(self, nome, preco):
        self.__nome = nome
        self.__preco = preco
    @property
    def nome(self):
        return self.__nome
    @property
    def preco(self):
        return self.__preco
    @preco.setter
    def preco(self, valor):
        if valor >= 0:
            self.__preco = valor
```

3. Polimorfismo

```
class Animal:
    def falar(self):
        pass

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

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

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

class Funcionario:
    def salario(self):
        pass

class Gerente(Funcionario):
    def salario(self):
        return 5000

class Estagiario(Funcionario):
    def salario(self):
        return 1500

class Veiculo:
    def mover(self):
        pass

class Carro(Veiculo):
    def mover(self):
        print("Dirigindo...")

class Bicicleta(Veiculo):
    def mover(self):
        print("Pedalando...")

class Aviao(Veiculo):
    def mover(self):
        print("Voando...")
```

4. Abstração

```
from abc import ABC, abstractmethod

class Pagamento(ABC):
    def __init__(self, valor):
        self.valor = valor
    @abstractmethod
    def processar_pagamento(self):
        pass

class Cancelavel(ABC):
    @abstractmethod
    def cancelar_pagamento(self):
        pass

class CartaoCredito(Pagamento, Cancelavel):
    def processar_pagamento(self):
        print(f"Pagamento de R${self.valor} no cartão aprovado.")
    def cancelar_pagamento(self):
        print("Pagamento no cartão cancelado.")

class Pix(Pagamento, Cancelavel):
    def processar_pagamento(self):
        print(f"Pagamento de R${self.valor} via Pix realizado.")
    def cancelar_pagamento(self):
        print("Pix cancelado.")

class Boleto(Pagamento, Cancelavel):
    def processar_pagamento(self):
        print(f"Boleto de R${self.valor} gerado.")
    def cancelar_pagamento(self):
        print("Boleto cancelado.")
```