Teóricas:

1 - Classe Anêmica: Um sistema onde as classes são projetadas para terem atributos e métodos de acesso, mas sem muita lógica de negócios embutida. Nesse cenário, uma classe anêmica faz sentido ao seguir o princípio de separação de responsabilidades, mantendo as operações de negócios em outras classes ou serviços. Facilita a manutenção e evolução do código.

Copy code

class PessoaAnemica:

def \_\_init\_\_(self, nome, idade, salario):

self.nome = nome

self.idade = idade

self.salario = salario

A classe lida com armazenamento de dados.

Classe Rica: Uma situação em que uma classe é responsável por não só armazenar dados, mas também contém lógica de negócios e funcionalidades específicas, pode-se considerar uma classe rica. Isso é apropriado quando entre dados e comportamentos é fundamental para o design.

Copy code

class ContaBancariaRica:

def \_\_init\_\_(self, saldo):

self.saldo = saldo

def depositar(self, valor):

self.saldo += valor

def sacar(self, valor):

if valor <= self.saldo:

self.saldo -= valor

else: print("Saldo insuficiente!")

A classe não apenas armazena dados, mas também incorpora lógica de negócios.

2 - É importante para facilitar a manutenção, compreensão e evolução do código.

Exemplo: Um sistema de gerenciamento de relatórios.

Copy code

class GeradorRelatorio:

def gerar\_relatorio(self, dados):

class EnviadorEmail:

def enviar\_email(self, destinatario, mensagem):

Cada classe tem uma única responsabilidade. Se houver mudanças nas regras de geração de relatórios, isso não afetará a classe de envio de e-mails. Isso torna o sistema mais flexível e fácil de manter.

3 - 1. Validação de formulários em sites:

* Importante: Garante que os dados inseridos por usuários em formulários de site estão corretos e atendem os requisitos esperados.
* Exemplo: Validar se um endereço de e-mail possui um formato válido antes de ser usado.

2. Validação de dados de entrada em sistemas:

* Importante: Previne a inserção de dados errados que podem comprometer a integridade e segurança do sistema.
* Exemplo: Verificar se os dados fornecidos em um campo numérico são realmente números e estão dentro de uma faixa aceitável.

3. Validação de campos em bancos de dados:

* Importante: Assegura a consistência e integridade dos dados armazenados no banco de dados, evitando informações inconsistentes.
* Exemplo: Certificar-se de que um campo que armazena datas contenha datas válidas e no formato adequado, evitando datas futuras em um contexto que não permite.

Motivos para nos preocuparmos com a validação de dados:

* Integridade dos Dados
* Segurança
* Experiência do Usuário
* Eficiência do Sistema