

POO, Concorrência e Boas Práticas

CATALOGUE

Contents

- Introdução
- Programação Orientada a Objetos (POO)
- Aplicações em Sistemas Operacionais
- Boas Práticas de Programação

Introdução

PART 01

Introdução

Objetivo

A Programação Orientada a Objetos (POO) foca em modelar o mundo real, definindo atributos e comportamentos para criar soluções alinhadas ao contexto real e facilitar o desenvolvimento de sistemas complexos.

Benefícios

Este paradigma melhora a legibilidade, escalabilidade e manutenção do código, fornecendo uma base sólida para desenvolver softwares estruturados.



Introdução

Aplicações em Sistemas Operacionais

POO e concorrência representam pilares técnicos avançados para lidar com threads, escalonamento e sincronização, ajudando a otimizar o desempenho de sistemas operacionais modernos.

Boas Práticas

Abordagens organizadas, como clareza de nomes, modularidade, uso de padrões de estilo e comentários significativos, são fundamentais para garantir código legível e sustentável.



Programação Orientada a Objetos (POO)

PART 02

Aula



Moldes para objetos

A classe funciona como um molde que encapsula atributos e métodos que posteriormente irão definir um objeto.



Definição técnica em Python

Classes em Python são criadas com a palavra-chave `class`, utilizando normas de nomenclatura CamelCase.



Estrutura organizada

Facilita a criação de objetos coerentes e bem estruturados para representar componentes reais em um programa.

Objeto (Instância)

Concretização das classes

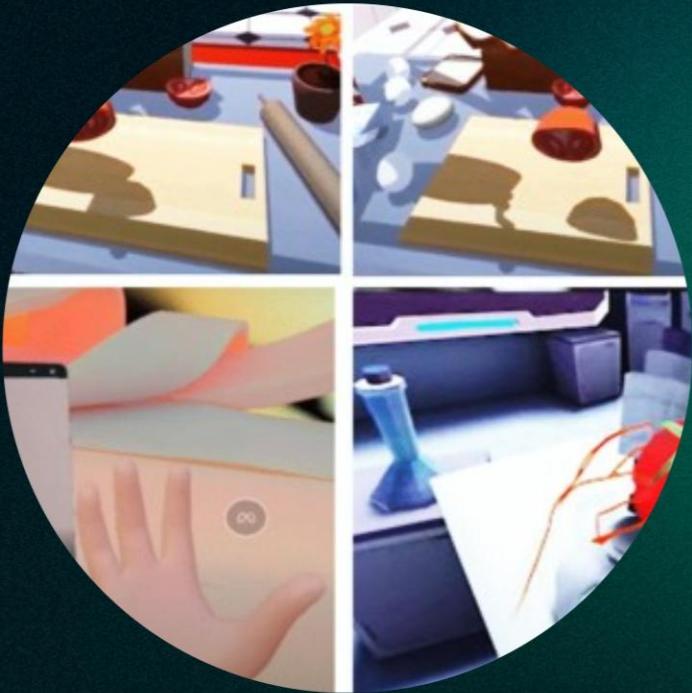
Objetos são concretizações no código, permitindo fins práticos ao que foi definido pela classe.

Especificidade em atributos

Cada objeto possui valores independentes para seus atributos, garantindo sua singularidade.

Interação programada

Objetos podem interagir dentro de um sistema fornecendo ações específicas e personalizáveis.



auto

Definição de classes

Uma classe é a base da programação orientada a objetos, estabelecendo a estrutura e características dos objetos com atributos e métodos, permitindo a modelagem do mundo real.

Objetos e suas instâncias

Objetos são instâncias concretas de classes, configuradas com valores específicos para seus atributos, proporcionando flexibilidade e reuso no desenvolvimento.

Uso do parâmetro self

O parâmetro `self` conecta os métodos a um objeto criado ou em operação, garantindo acesso ao seu estado interno.

Método construtor (`__init__`)

Este método especial permite configurar os atributos iniciais do objeto ao ser criado, facilitando sua personalização e inicialização.

Alinhamento com cenários reais

POO promove uma abordagem intuitiva e estruturada ao modelar problemas reais, melhorando tanto o desenvolvimento como a manutenção de sistemas complexos.

Método Construtor (`__init__`)

Inicialização controlada

Este método especial permite configurar atributos iniciais no momento da criação do objeto.

Flexibilidade na criação de objetos

Oferece parâmetros opcionais ou obrigatórios para personalizar cada instância na criação.

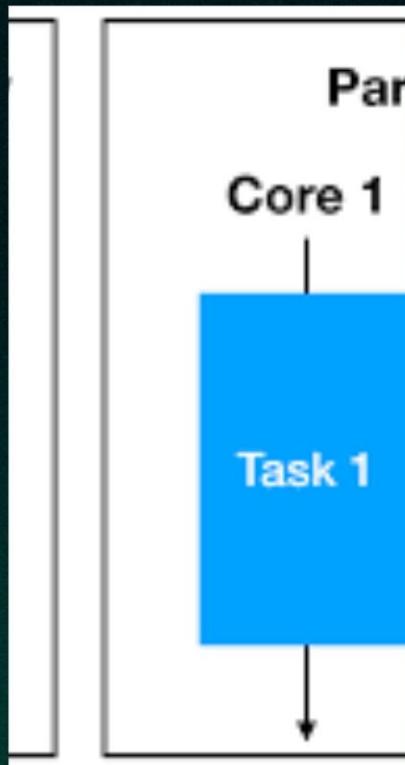
Estruturação de dados

Impõe uma organização inicial ao objeto, facilitando sua utilização e prolongando sua eficiência em aplicações complexas.

Aplicações em Sistemas Operacionais

PART 03

Concorrência (Threads)



Sincronização de recursos compartilhados

Utiliza métodos como `join()` e objetos como `Lock` e `Condition` para garantir acesso controlado às regiões críticas, evitando corrupção de dados.

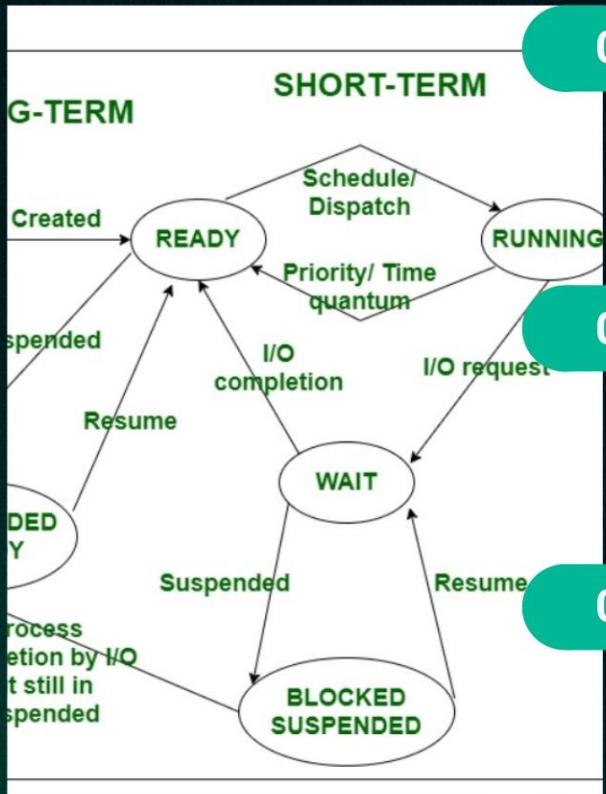
Problemas clássicos de concorrência

Exemplos como Produtor e Consumidor ou o Jantar dos Filósofos demonstram desafios importantes e a necessidade de estratégias robustas para evitar deadlocks.

Execução paralela eficiente

Threads possibilitam dividir tarefas e utilizar melhor os recursos computacionais, aumentando a velocidade e a capacidade de processamento de sistemas.

Escalonamento



01

Algoritmo FIFO (First-In, First-Out)

Prioriza a execução de processos na ordem de chegada, ideal para sistemas com cargas previsíveis e tarefas simples.

02

Algoritmo SJF (Shortest Job First)

Escolhe os processos de menor duração para execução, reduzindo o tempo médio de espera e maximizando a eficiência do sistema.

03

Avaliação de desempenho do sistema

Variáveis como tempo de conclusão, duração e tempo de retorno são essenciais para medir os impactos e ajustar os algoritmos de escalonamento.

Boas Práticas de Programação

PART 04

Clareza de Nomes

Escolher nomes que representem claramente o propósito dos elementos no código

Permite que o leitor comprehenda rapidamente o significado de variáveis, funções e classes, facilitando o entendimento da lógica geral.

Evitar abreviações excessivas

Nomes abreviados podem gerar confusão. Por exemplo, usar data_validacao em vez de dt_val.

Usar convenções consistentes

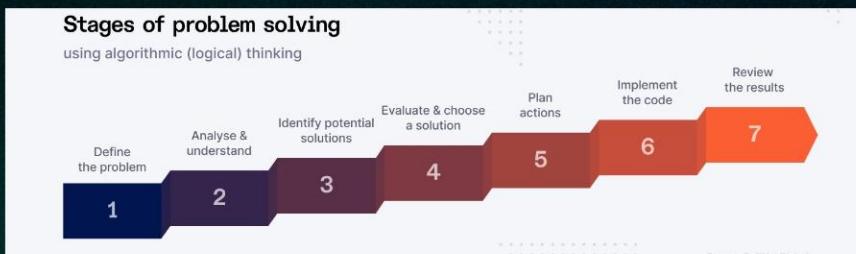
Adoção de padrões formais como camelCase ou snake_case garante uniformidade no projeto e evita ambiguidades.



Comentários

Explicar decisões complexas

Comentários devem focar em esclarecer o raciocínio por trás de uma lógica difícil, garantindo entendimento futuro por outros desenvolvedores.



Relacionar código a requisitos

Vincular o comentário ao objetivo ou requisito funcional seja essencial, contribuindo para uma documentação comprehensível.



Evitar redundâncias

Comentários que apenas repetem o código devem ser evitados; escreva apenas quando adicionar valor ao entendimento.



Organização

Modularizar em funções específicas

Dividir o código em pequenas funções que executam tarefas específicas melhora a reutilização e a legibilidade.

Estruturar por responsabilidade

Organizar o código baseado em responsabilidades e separar cada módulo por contexto funcional.

Facilitar testes e manutenção

Ao manter o código dividido logicamente, testes tornam-se mais precisos e ajustes posteriores são implementados com maior facilidade.



Padrão de Estilo

Seguir o PEP 8 em projetos Python



Usar indentação, espaçamento e convenções claras definidas por guias oficiais da linguagem.

Utilizar ferramentas automáticas de checagem

Ferramentas como o `pylint` podem garantir aderência ao estilo e manter o código limpo.



Priorizar a legibilidade sobre otimização prematura



Um código legível e padronizado é mais valioso a longo prazo do que ganhos marginais de performance que complicam a compreensão.

Obrigado