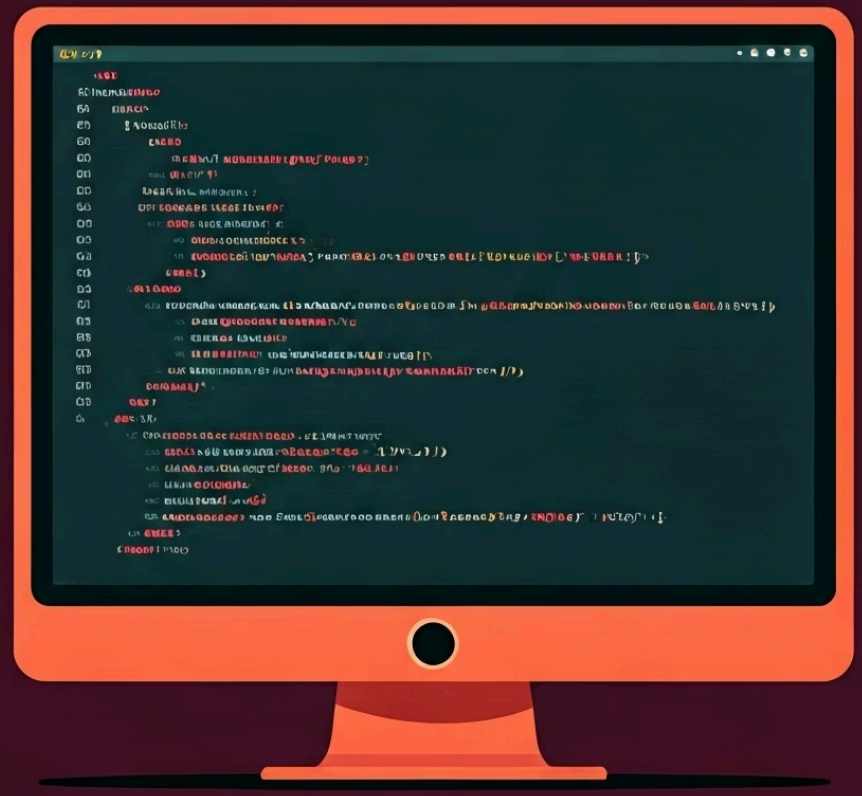
Aula 05 – Funções II: Avançando com Argumentos

Explorando técnicas avançadas de programação funcional em Python



Made with GAMMA

Objetivos da Aula

Aprofundar Funções

Focar em argumentos e modularização para código mais organizado

Explorar Argumentos

Dominar argumentos nomeados, padrão e funções como parâmetros

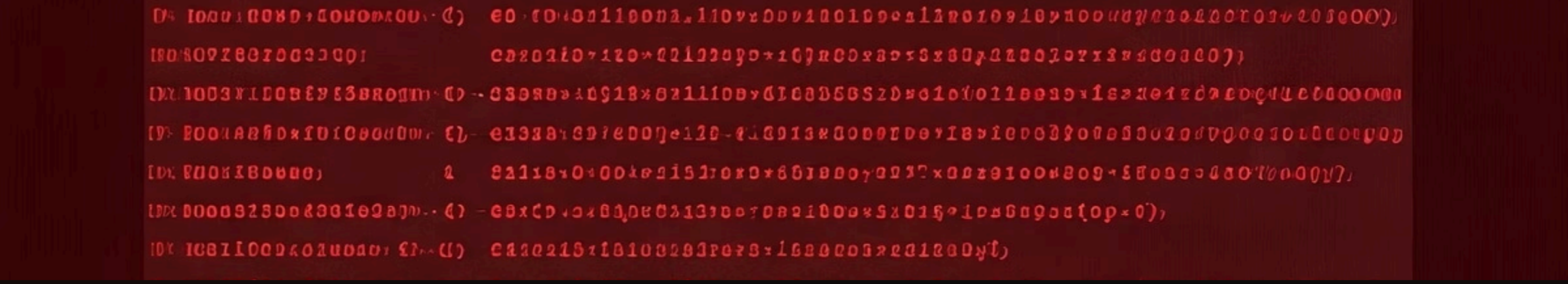
Organizar Código

Estruturar programas em módulos funcionais reutilizáveis

Prática Aplicada

Desenvolver menus funcionais e simulador bancário completo

Made with GAMMA



Revisão Rápida

O que já sabemos sobre funções:

Declaração com def

Definindo funções usando a palavra-chave `def`

Argumentos posicionais

Passando valores na ordem correta dos parâmetros

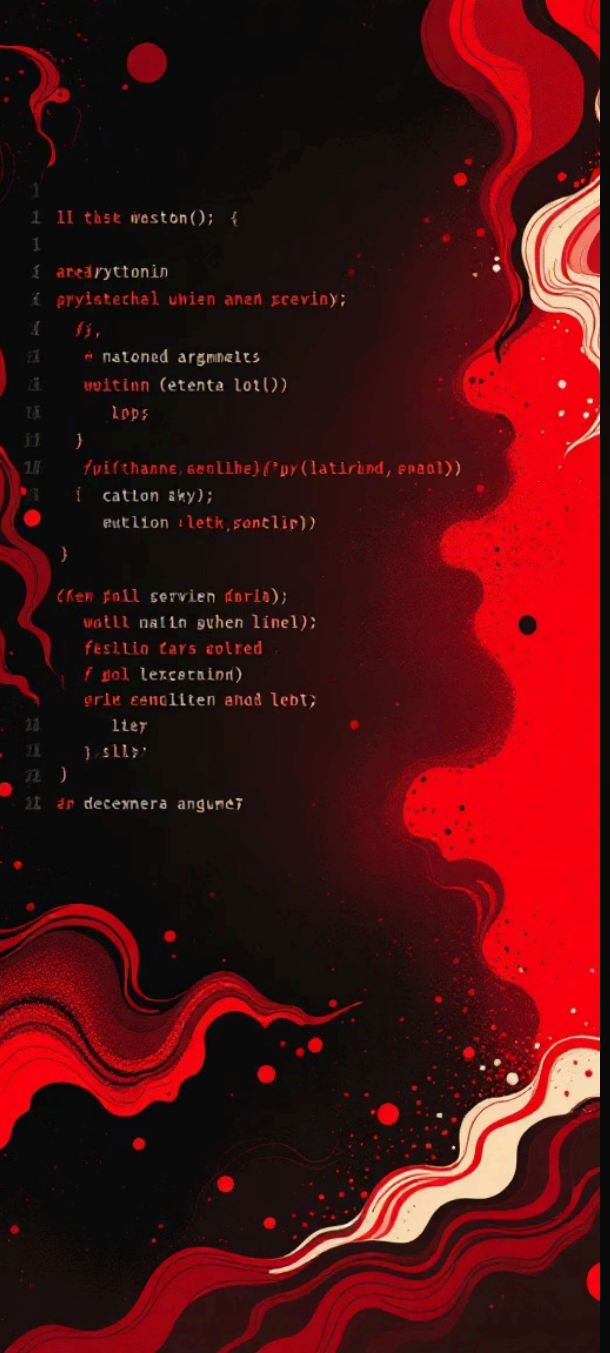
Retorno com return

Devolvendo valores processados pela função

Escopo local e global

Compreendendo onde as variáveis existem no código

Made with GAMMA



Argumentos Nomeados

Vantagens principais:

- Permitem passar valores explicitamente pelo nome do parâmetro
- Aumentam significativamente a clareza do código
- Reduzem erros de ordem dos argumentos

Exemplo prático:

```
def saudacao(nome, mensagem):
    print(f"{mensagem}, {nome}!")

# Usando argumentos nomeados
saudacao("Ana", "Bom dia")
saudacao(mensagem="Boa tarde", nome="João")
```

Dica: Use argumentos nomeados quando tiver muitos parâmetros!

Made with GAMMA

Argumentos Padrão

Definição

Valores automáticos quando o usuário não passa o argumento específico

Flexibilidade

Tornam a função mais versátil e fácil de usar em diferentes situações

Exemplo em ação:

```
def boas_vindas(nome, curso="Python"):
  print(f"Bem-vindo(a) {nome} ao curso de {curso}!")

# Usando valor padrão
boas_vindas("Carlos")

# Sobrescrevendo o padrão
boas_vindas("Maria", "JavaScript")
```

Made with GAMMA

Funções como Argumentos

Conceito poderoso:

Em Python, funções também podem ser passadas como parâmetros para outras funções. Isso permite:

- Maior modularidade no código
- Reuso inteligente de lógica
- Programação mais dinâmica

Exemplo prático:

```
def aplicar_operacao(a, b, operacao):
    return operacao(a, b)

# Usando funções lambda
print(aplicar_operacao(10, 5, lambda x, y: x + y))
print(aplicar_operacao(10, 5, lambda x, y: x * y))
```

Modularização com Funções

Blocos Reutilizáveis

Separar o código em pequenos módulos funcionais que podem ser usados em diferentes partes do programa

Facilita Manutenção

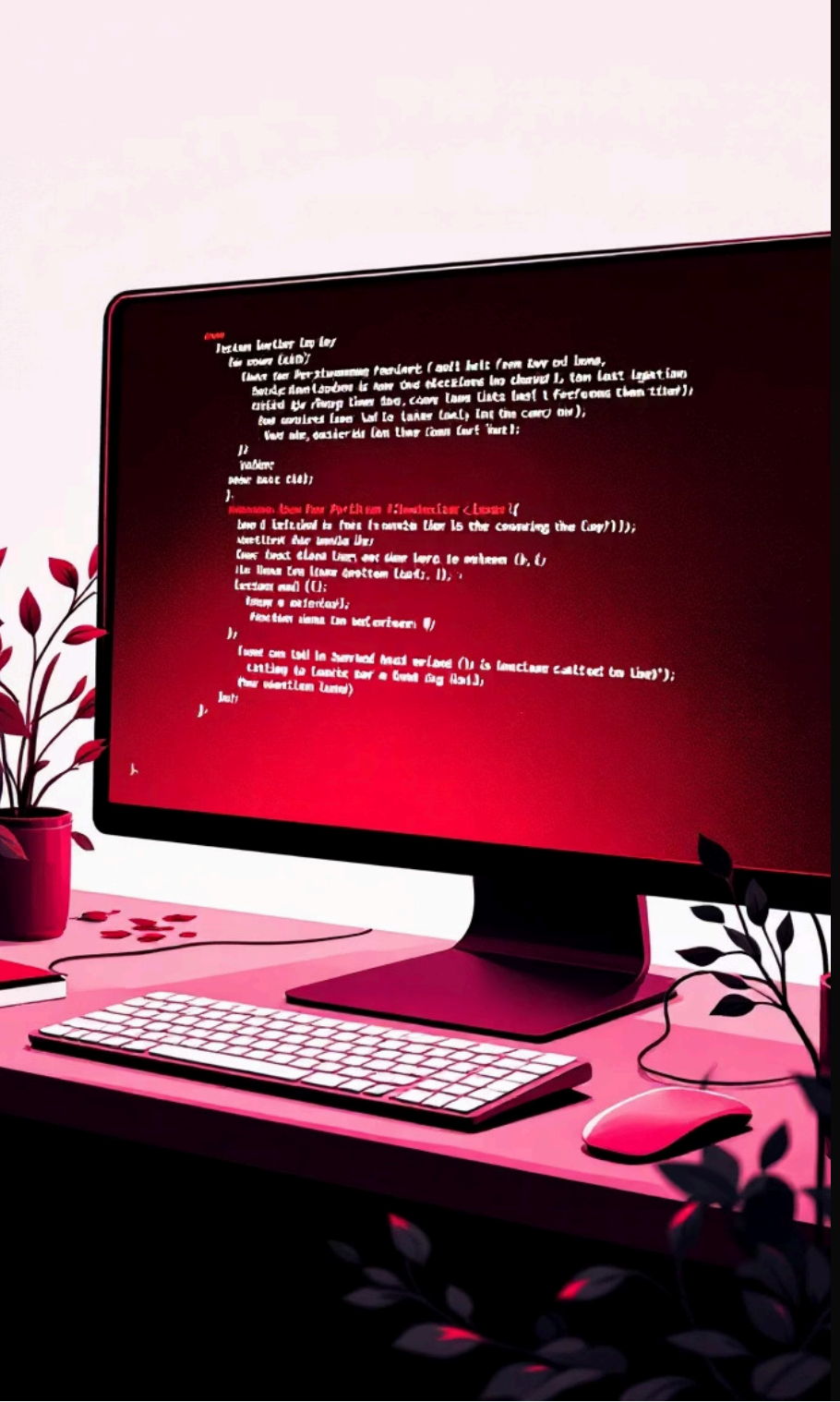
Evita repetição desnecessária e torna muito mais fácil fazer alterações e correções no código

Melhora Legibilidade

Código organizado em funções específicas é mais fácil de entender e debugar

Exemplo prático: Criar funções como `menu_principal()`, `depositar()`, `sacar()` em vez de concentrar tudo no mesmo bloco gigante.

Made with GAMMA



Atividades Práticas

01

Função de Boas-vindas

Criar uma função que receba um nome e mostre uma mensagem personalizada de boas-vindas

02

Calcular Quadrado

Criar uma função que calcule o quadrado de um número (com argumento e retorno claro)

03

Saudação Padrão

Criar uma função com argumento padrão que exiba uma saudação (ex.: "Olá, visitante!")

04

Soma de Números

Criar uma função que receba dois números e exiba a soma deles de forma clara

Made with GAMMA

Prática Guiada

Atividades em sala de aula:

1

Menu Funcional

Criar um sistema de menu interativo usando funções organizadas e modulares

2

Operações Matemáticas

Implementar operações básicas: soma, subtração, multiplicação com interface amigável

3

Simulador Bancário

Desenvolver um sistema completo com: depositar valor, sacar valor, verificar saldo

Made with GAMMA

Resumo da Aula:

Argumentos Avançados

Dominamos argumentos nomeados e padrão para código mais flexível

Funções como Parâmetros

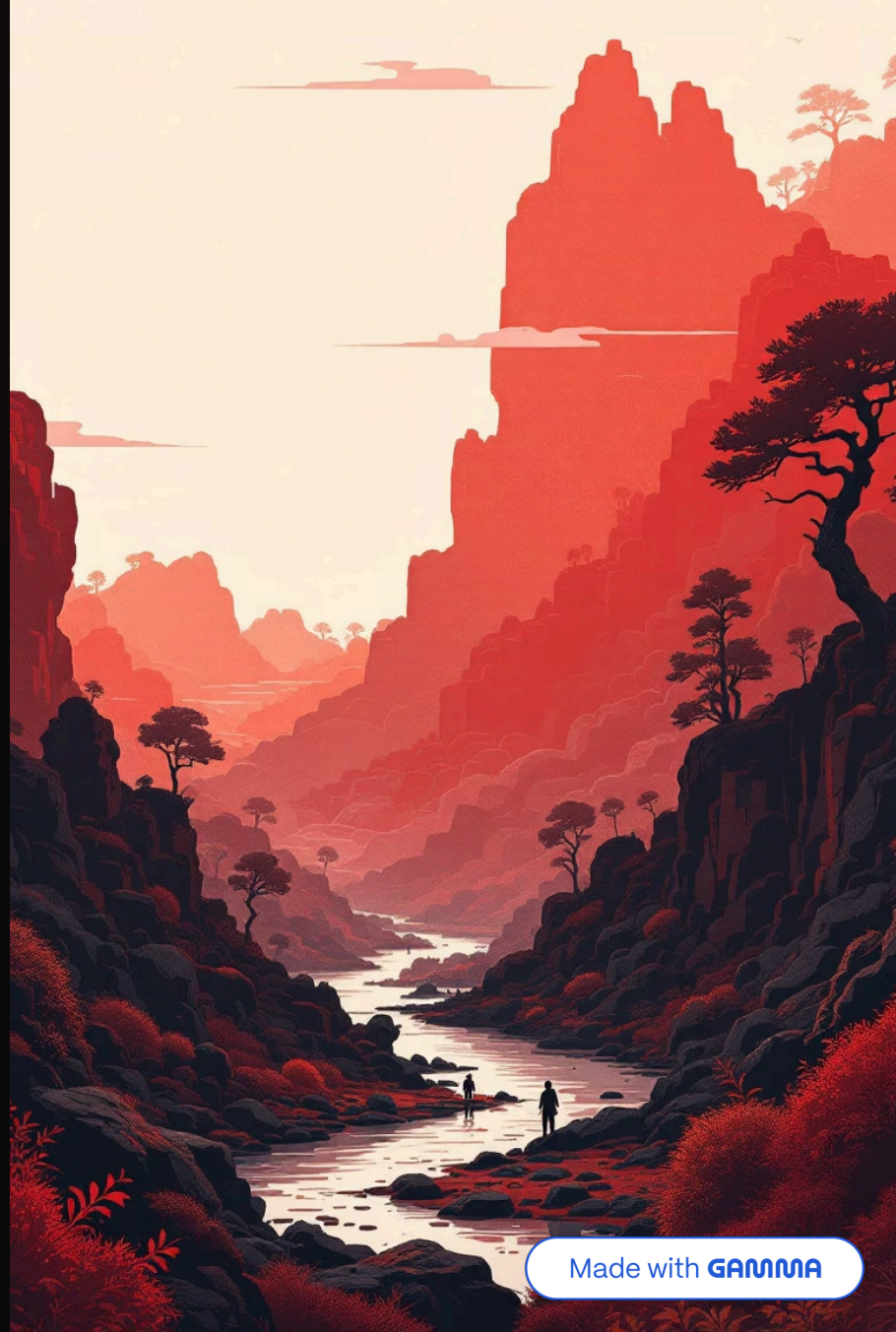
Aprendemos a passar funções como argumentos para maior modularidade

Modularização

Compreendemos a importância de organizar código em blocos funcionais

Aplicação Prática

Criamos menus funcionais e simuladores bancários reais



Made with GAMMA