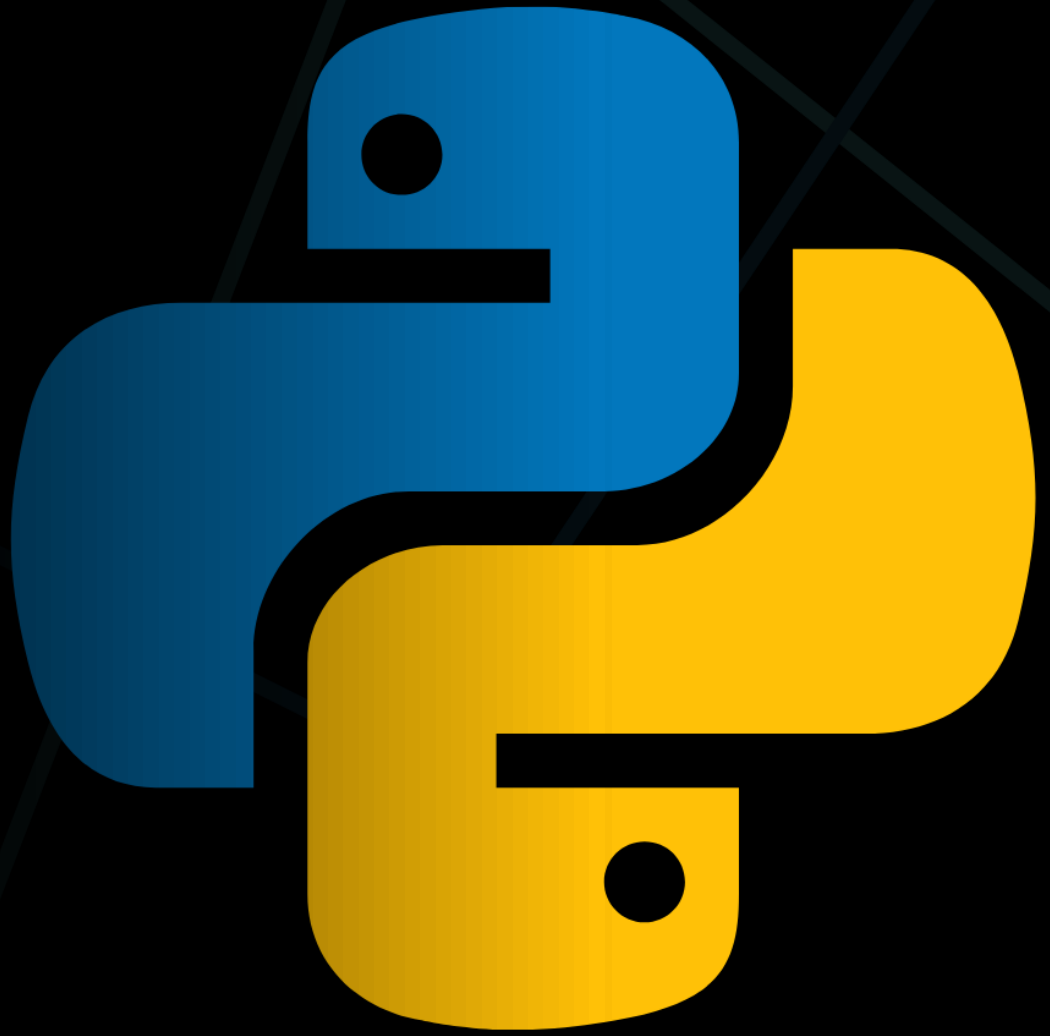
O que são funções em python

Programação em python



Fundamento Técnico: o conceito de Função

Uma função em Python é um bloco independente de código criado para executar uma tarefa específica. Ela é projetada para receber valores, processar dados e retornar resultados, de forma organizada, reutilizável e modular.

Em termos técnicos, as funções evitam repetição de código e aumentam a manutenibilidade dos sistemas, sendo um dos fundamentos da engenharia de software estruturada.

Fundamento Técnico: o conceito de Função

Estrutura sintática básica:

```
def nome_da_funcao():
    # bloco de código
```

def → palavra reservada que indica a criação de uma função;

nome_da_funcao → identificador (segue as regras de nomeação vistas na PEP 8);

() → define o espaço para parâmetros de entrada;

: → marca o início do bloco indentado.

Explicação conceitual simples

Pense em uma função como uma máquina de tarefas:

- Ela recebe algo (dados, valores ou informações);
- Realiza um processamento interno;
- E retorna um resultado.

Em um ambiente corporativo, funções são usadas em cadastros automáticos, cálculos de relatórios, validação de formulários, controle logístico e sistemas de auditoria.

Diferença entre: código linear e código funcional

```
# Código linear (sem função):
# Exibe uma mensagem informando o início do cálculo de comissão
print("Iniciando cálculo de comissão...")
# Define o valor da venda
venda = 1500
# Calcula a comissão, que é 10% do valor da venda (0.1 representa 10%)
comissao = venda * 0.1
# Exibe o valor da comissão calculada
print("Comissão calculada:", comissao)
# Exibe novamente a mensagem de início do cálculo de comissão
print("Iniciando cálculo de comissão...")
# Define um novo valor de venda
venda = 2300
# Calcula a nova comissão com base na venda atual
comissao = venda * 0.1
# Exibe o valor da nova comissão calculada
print("Comissão calculada:", comissao)

#Esse tipo de código repete blocos iguais, o que aumenta o risco de erro e dificulta a manutenção.
```

```python
# Código linear (sem função):
print("Iniciando cálculo de comissão...")
venda = 1500
comissao = venda * 0.1
print("Comissão calculada:", comissao)

print("Iniciando cálculo de comissão...")
venda = 2300
comissao = venda * 0.1
print("Comissão calculada:", comissao)

#Esse tipo de código repete blocos iguais, o que aumenta o risco de erro e dificulta a manutenção.
```

```
# Código funcional (com função):
# Define uma função chamada 'calcular_comissao' que recebe um parâmetro 'valor_venda'
def calcular_comissao(valor_venda):
    # Calcula a comissão multiplicando o valor da venda por 10% (0.1)
    comissao = valor_venda * 0.1
    # Exibe o valor da comissão calculada na tela
    print("Comissão calculada:", comissao)
# Chama a função 'calcular_comissao' passando 1500 como valor de venda
# Isso executa o cálculo de 10% sobre 1500 e mostra o resultado
calcular_comissao(1500)
# Chama novamente a função, agora com o valor de 2300
# A função é reutilizada sem precisar reescrever o cálculo
calcular_comissao(2300)
# O uso de função transforma o processo em estrutura modular, permitindo que o mesmo bloco seja executado várias vezes com diferentes valores de entrada. Esse é o princípio da reutilização de código, base da programação profissional.
```

```python
# Código funcional (com função):
def calcular_comissao(valor_venda):
    comissao = valor_venda * 0.1
    print("Comissão calculada:", comissao)

calcular_comissao(1500)
calcular_comissao(2300)

# O uso de função transforma o processo em estrutura modular, permitindo que o mesmo bloco seja executado várias vezes com diferentes valores de entrada. Esse é o princípio da reutilização de código, base da programação profissional.
```

Código fácil (função simples sem parâmetros)

```
def mensagem_boas_vindas():
    print("Sistema de Cadastro iniciado.")
    print("Conectando ao servidor principal...")
    print("Ambiente pronto para uso.")


mensagem_boas_vindas()
```

Análise técnica:

- A função não recebe parâmetros;
- Serve como inicialização padrão de sistemas;
- É comum em rotinas de abertura de aplicações corporativas.

```python
# Define uma função chamada 'mensagem_boas_vindas' que não recebe parâmetros
def mensagem_boas_vindas():
    # Exibe uma mensagem indicando que o sistema de cadastro foi iniciado
    print("Sistema de Cadastro iniciado.")

    # Exibe uma mensagem informando que o sistema está se conectando ao servidor
    print("Conectando ao servidor principal...")

    # Exibe uma mensagem confirmando que o ambiente está pronto para ser utilizado
    print("Ambiente pronto para uso.")


# Chama (executa) a função 'mensagem_boas_vindas'. Ao ser chamada, todas as mensagens dentro da função são exibidas na tela
mensagem_boas_vindas()
```

Código intermediário (função com parâmetros)

```
def registrar_usuario(nome, setor):
    print("Usuário cadastrado:", nome)
    print("Setor associado:", setor)
    print("Cadastro realizado com sucesso.\n")

registrar_usuario("Marina Silva", "Financeiro")
registrar_usuario("Carlos Mendes", "Logística")
```

Análise técnica:

- A função possui dois parâmetros (nome e setor);
- Os valores são passados diretamente na chamada;
- Esse formato é amplamente usado em rotinas administrativas e de RH, simulando cadastros em sistemas empresariais.

ython.py > ...

```python
# Define uma função chamada 'registrar_usuario' que recebe dois parâmetros: 'nome' e 'setor'
def registrar_usuario(nome, setor):
    # Exibe uma mensagem informando o nome do usuário que foi cadastrado
    print("Usuário cadastrado:", nome)
    # Exibe o setor ao qual o usuário está associado
    print("Setor associado:", setor)
    # Exibe uma mensagem final confirmando que o cadastro foi concluído com sucesso. O caractere '\n' cria uma linha em branco após a mensagem, melhorando a visualização na tela
    print("Cadastro realizado com sucesso.\n")

# Chama a função 'registrar_usuario' passando o nome e o setor como argumentos. Aqui, o sistema cadastra a usuária "Marina Silva" no setor "Financeiro"
registrar_usuario("Marina Silva", "Financeiro")
# Chama novamente a função para registrar outro usuário. Agora, o sistema cadastra "Carlos Mendes" no setor "Logística"
registrar_usuario("Carlos Mendes", "Logística")
```

Código avançado (função com processamento interno e retorno)

```
def calcular_desconto(valor_compra):
    if valor_compra > 1000:
        desconto = valor_compra * 0.10
    elif valor_compra >= 500:
        desconto = valor_compra * 0.05
    else:
        desconto = 0
    return valor_compra - desconto


valor_final = calcular_desconto(1200)
print("Valor final após desconto:", valor_final)
```

Explicação técnica:

- A função processa dados internamente e retorna um valor;
- O return envia o resultado para o ponto onde a função foi chamada;
- O controle condicional interno torna a função inteligente e adaptativa.

Esse padrão é utilizado em sistemas de vendas, e-commerce e cálculos tributários.

ython.py > ...

```python
# Define uma função chamada 'calcular_desconto' que recebe o parâmetro 'valor_compra'
def calcular_desconto(valor_compra):
    # Verifica se o valor da compra é maior que 1000 reais. Se for, aplica um desconto de 10% (0.10)
    if valor_compra > 1000:
        desconto = valor_compra * 0.10
    # Caso o valor não seja maior que 1000, mas seja igual ou superior a 500, aplica um desconto de 5% (0.05)
    elif valor_compra >= 500:
        desconto = valor_compra * 0.05
    # Caso o valor da compra seja inferior a 500, não aplica desconto
    else:
        desconto = 0
    # Retorna o valor final após subtrair o desconto do valor original
    return valor_compra - desconto
# Chama a função 'calcular_desconto' passando o valor de 1200 como argumento. O resultado será armazenado na variável 'valor_final'
valor_final = calcular_desconto(1200)
# Exibe o valor final após o cálculo do desconto
print("Valor final após desconto:", valor_final)
```

Código avançado (função integrada com estruturas condicionais e lógicas)

```
def validar_login(usuario, senha):
    if usuario.lower() == "admin" and senha == "1234":
        return "Acesso autorizado."
    else:
        return "Credenciais inválidas."

print(validar_login("admin", "1234"))
print(validar_login("joao", "1111"))
```

Explicação técnica:

- Este exemplo integra condições lógicas e manipulação de strings.
- É utilizado em sistemas de autenticação, portais administrativos e validação de acessos corporativos.

```python
# Define uma função chamada 'validar_login' que recebe dois parâmetros: 'usuario' e 'senha'
def validar_login(usuario, senha):
    # Converte o nome do usuário para letras minúsculas (com .lower()) e verifica se ele é igual a "admin". Também verifica se a senha é exatamente "1234". As duas condições precisam ser verdadeiras para o acesso ser autorizado
    if usuario.lower() == "admin" and senha == "1234":
        # Retorna a mensagem de sucesso se o usuário e a senha estiverem corretos
        return "Acesso autorizado."
    else:
        # Caso contrário, retorna uma mensagem informando que as credenciais são inválidas
        return "Credenciais inválidas."
# Chama a função 'validar_login' passando o usuário "admin" e a senha "1234". Como ambos estão corretos, o retorno será "Acesso autorizado."
print(validar_login("admin", "1234"))
# Chama novamente a função, mas agora com credenciais incorretas. O usuário "joao" e a senha "1111" não correspondem aos dados esperados. O retorno será "Credenciais inválidas."
print(validar_login("joao", "1111"))
```

Aplicação corporativa

Simulação de um fluxo de cadastro e controle administrativo usando múltiplas funções. Aqui as funções se comunicam, representando divisão de responsabilidades — técnica essencial em programação modular.

```
def verificar_dados(nome, idade):
    if nome == "" or idade <= 0:
        return False
    return True

def calcular_idade_apos_anos(idade, anos):
    return idade + anos

def cadastrar_funcionario(nome, idade, anos):
    if verificar_dados(nome, idade):
        nova_idade = calcular_idade_apos_anos(idade, anos)
        print(f"Funcionário {nome.title()} cadastrado. Idade atualizada: {nova_idade} anos.")
    else:
        print("Erro no cadastro: dados inválidos.")

cadastrar_funcionario("Carlos", 29, 3)
cadastrar_funcionario("", 22, 5)
```

Análise corporativa:

- Uso real de funções interconectadas;
- Cada função resolve uma tarefa específica;
- Estrutura típica em sistemas de RH, ERP e automação empresarial.

```python
# Define uma função chamada 'verificar_dados' que recebe dois parâmetros: nome e idade.
def verificar_dados(nome, idade):
    # Verifica se o nome está vazio ("") ou se a idade é menor ou igual a zero. Caso qualquer uma dessas condições seja verdadeira, os dados são considerados inválidos.
    if nome == "" or idade <= 0:
        return False  # Retorna False indicando que os dados não são válidos.
    return True  # Caso contrário, retorna True, indicando que os dados estão corretos.

# Define uma função chamada 'calcular_idade_apos_anos' que recebe dois parâmetros: idade e anos.
def calcular_idade_apos_anos(idade, anos):
    # Retorna a soma da idade atual com a quantidade de anos informada.
    return idade + anos

# Define uma função chamada 'cadastrar_funcionario' que recebe três parâmetros: nome, idade e anos.
def cadastrar_funcionario(nome, idade, anos):
    # Verifica se os dados do funcionário são válidos, chamando a função 'verificar_dados'.
    if verificar_dados(nome, idade):
        # Caso os dados sejam válidos, calcula a nova idade chamando 'calcular_idade_apos_anos'.
        nova_idade = calcular_idade_apos_anos(idade, anos)
        # Exibe uma mensagem formatada com o nome do funcionário e sua nova idade. O método .title() coloca a primeira letra de cada palavra em maiúscula.
        print(f"Funcionário {nome.title()} cadastrado. Idade atualizada: {nova_idade} anos.")
    else:
        # Caso os dados sejam inválidos, exibe uma mensagem de erro.
        print("Erro no cadastro: dados inválidos.")

# Chama a função 'cadastrar_funcionario' passando um nome, idade e número de anos. Neste caso, os dados são válidos.
cadastrar_funcionario("Carlos", 29, 3)

# Chama novamente a função, mas com o nome vazio, o que causará erro no cadastro.
cadastrar_funcionario("", 22, 5)
```