



Turtle Track

**Apresentação de Sistema
Controle de Estoque**

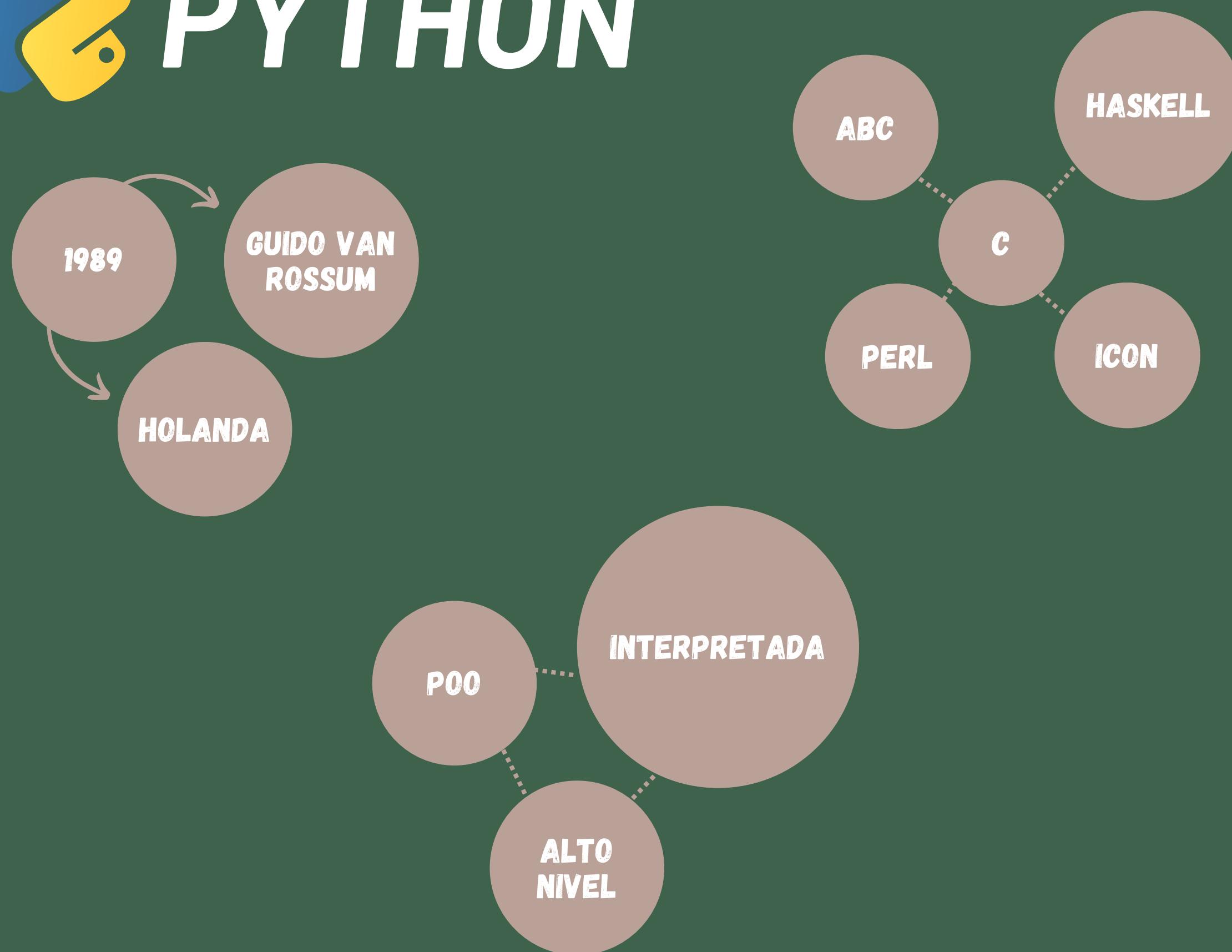
INTRODUÇÃO



- 1 PYTHON
- 2 BIBLIOTECA SQLITE
- 3 CÓDIGO FONTE
- 4 CONCLUSÃO



PYTHON



BIBLIOTECAS



BIBLIOTECA SQLITE

FACILIDADE
DE USO

AUTOCONTIDO

SEM
SERVIDOR

CONFIGURAÇÃO
ZERO

TRANSACIONAL

PORTABILIDADE

AMPLA ADOÇÃO

BAIXO CONSUMO
DE RECURSOS

IMPORTAÇÕES

```
1 ✓ import tkinter as tk  
2 from tkinter import ttk, messagebox  
3 import sqlite3
```

import → todo
from → funções
as → apelida

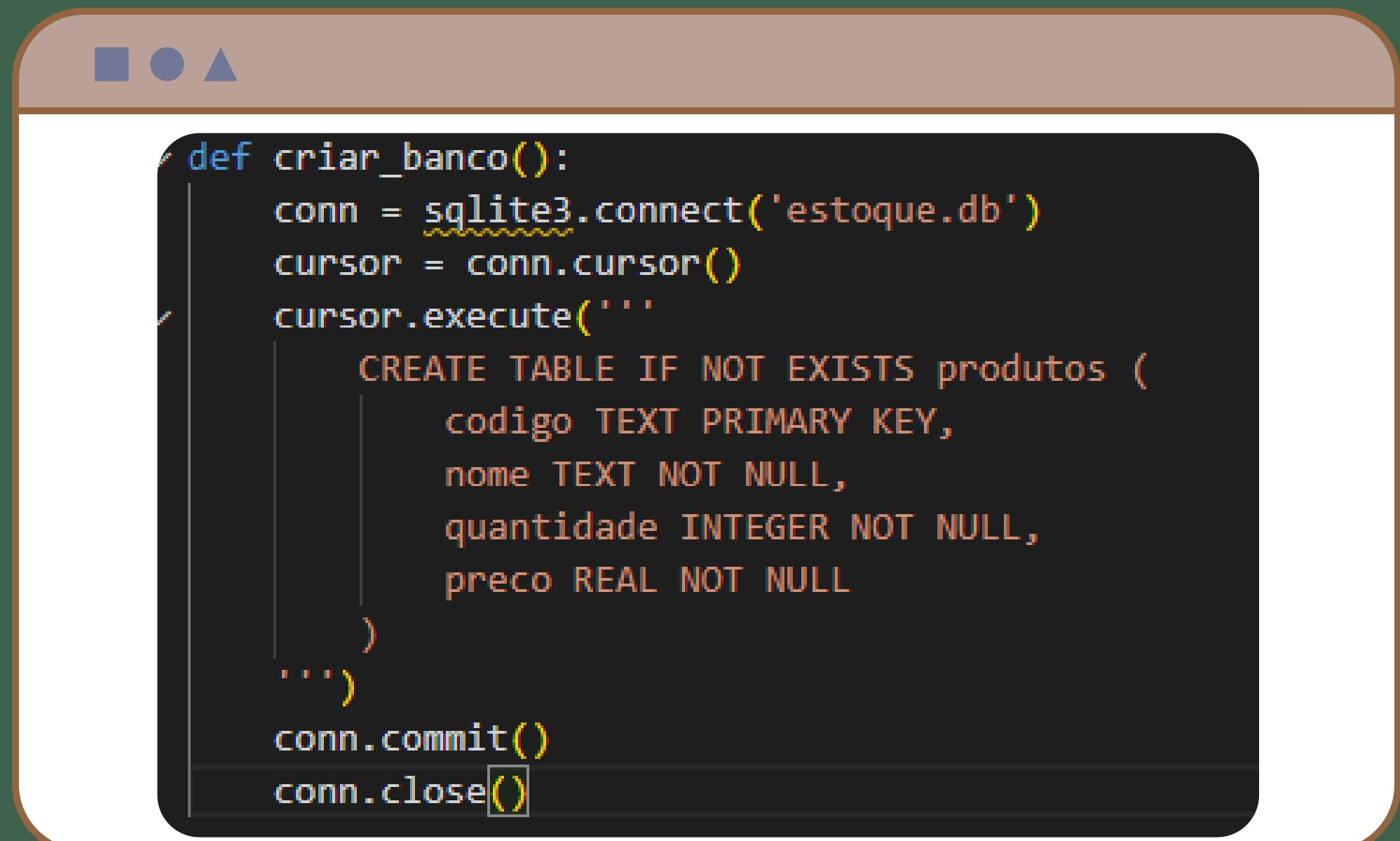
BANCO DE DADOS

conn → variável de conexão

cursor = → **conn.cursor()**
cria um objeto
para interagir
com o banco
de dados

.commit → salva as
configurações

.close → encerra a conexão



```
def criar_banco():
    conn = sqlite3.connect('estoque.db')
    cursor = conn.cursor()
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS produtos (
            codigo TEXT PRIMARY KEY,
            nome TEXT NOT NULL,
            quantidade INTEGER NOT NULL,
            preco REAL NOT NULL
        )
    ''')
    conn.commit()
    conn.close()
```

FUNÇÕES

```
def adicionar_produto_interface():
    codigo = cod_entry.get()
    nome = name_entry.get()
    try:
        quantidade = int(qtd_entry.get())
        preco = float(preco_entry.get())
    except ValueError:
        messagebox.showerror("Erro", "Quantidade e preço devem ser números válidos.")
        return

    # Jogando dados pra função add BD
    salvar_produto_bd(codigo, nome, quantidade, preco)
```

Pega os dados da interface através do nome das entrys, alocando estes dados em variáveis. Chama a função *salvar_produto_bd* dando as variáveis criadas como parâmetros.

TRY



transforma os valores recebidos nos tipos certos

EXCEPT



acaso o try falhar, mostra uma mensagem de erro

ADD PRODUTO

```
def salvar_produto_bd(codigo, nome, quantidade, preco):
    conn = sqlite3.connect('estoque.db')
    cursor = conn.cursor()
    try:
        cursor.execute('''
            INSERT INTO produtos (codigo, nome, quantidade, preco)
            VALUES (?, ?, ?, ?)
            ''', (codigo, nome, quantidade, preco))
        conn.commit()
        messagebox.showinfo("Sucesso", f"Produto {nome} salvo com sucesso!")
    except sqlite3.IntegrityError:
        messagebox.showerror("Erro", f"Produto com código {codigo} já existe.")
    conn.close()
```

Cria um comando SQL de
INSERT na Tabela Produtos
com os dados codigo,
nome, quantidade, preco

INSERT



insere os dados
na tabela

INTO



indica a tabela
na qual os
dados serão
alocados

FUNÇÕES



```
def busca_produto_interface_edit():
    codigo = cod_edit_entry.get()
    produto = buscar_produto_bd(codigo)

    if produto:
        # Exibir os dados do produto na interface de busca
        nome_edit_entry.config(state='normal') # Permite edição
        nome_edit_entry.delete(0, tk.END)
        nome_edit_entry.insert(0, produto[1]) # Nome
        nome_edit_entry.config(state='readonly') # Modo somente leitura

        qtd_edit_entry.config(state='normal')
        qtd_edit_entry.delete(0, tk.END)
        qtd_edit_entry.insert(0, produto[2]) # Quantidade
        qtd_edit_entry.config(state='readonly')

        preco_edit_entry.config(state='normal')
        preco_edit_entry.delete(0, tk.END)
        preco_edit_entry.insert(0, produto[3]) # Preço
        preco_edit_entry.config(state='readonly')
    else:
        messagebox.showerror("Erro", f"Produto com código {codigo} não encontrado.")

    nome_edit_entry.config(state='normal') # Permite edição
    qtd_edit_entry.config(state='normal')
    preco_edit_entry.config(state='normal')
```

Pega os dados da interface através do nome das entry, alocando estes dados em uma variável. Chama a função *buscar_produto_bd* dando a variável criada como parâmetro.

.CONFIG
STATE



muda o estado da entry, permitindo ou não edição

.DELETE



apaga os dados que estiverem na entry

.INSERT



pega determinado dado da variável, e insere na entry

BUSCAR PRODUTO

Cria um comando SQL de
SELECT na Tabela Produtos
filtrando pelo código



```
def buscar_produto_bd(codigo):
    conn = sqlite3.connect('estoque.db')
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM produtos WHERE codigo = ?", (codigo,))
    produto = cursor.fetchone()
    conn.close()
    return produto
```

- SELECT → procura os dados desejados na tabela
- * → todos; procura por todos os dados
- FETCHONE() → retorna a ultima tupla selecionada
- RETURN → retorna a variavel com os dados obtidos

FUNÇÕES



```
def editar_produto_interface():
    codigo = cod_edit_entry.get()
    nome = nome_edit_entry.get()
    try:
        quantidade = int(qtd_edit_entry.get())
        preco = float(preco_edit_entry.get())
    except ValueError:
        messagebox.showerror("Erro", "Quantidade e preço devem ser números válidos.")
        return

    # Jogando dados pra função edit BD
    edit_produto_bd(codigo, nome, quantidade, preco)

    # Limpa os campos da interface
    cod_busca_entry.delete(0, tk.END)
    nome_busca_entry.config(state='normal')
    nome_busca_entry.delete(0, tk.END)
    qtd_busca_entry.config(state='normal')
    qtd_busca_entry.delete(0, tk.END)
    preco_busca_entry.config(state='normal')
    preco_busca_entry.delete(0, tk.END)
```

Pega os dados da interface através do nome das entrys, alocando estes dados em variáveis. Chama a função `edit_produto_bd` dando as variáveis criadas como parâmetros.

EDITAR PRODUTO

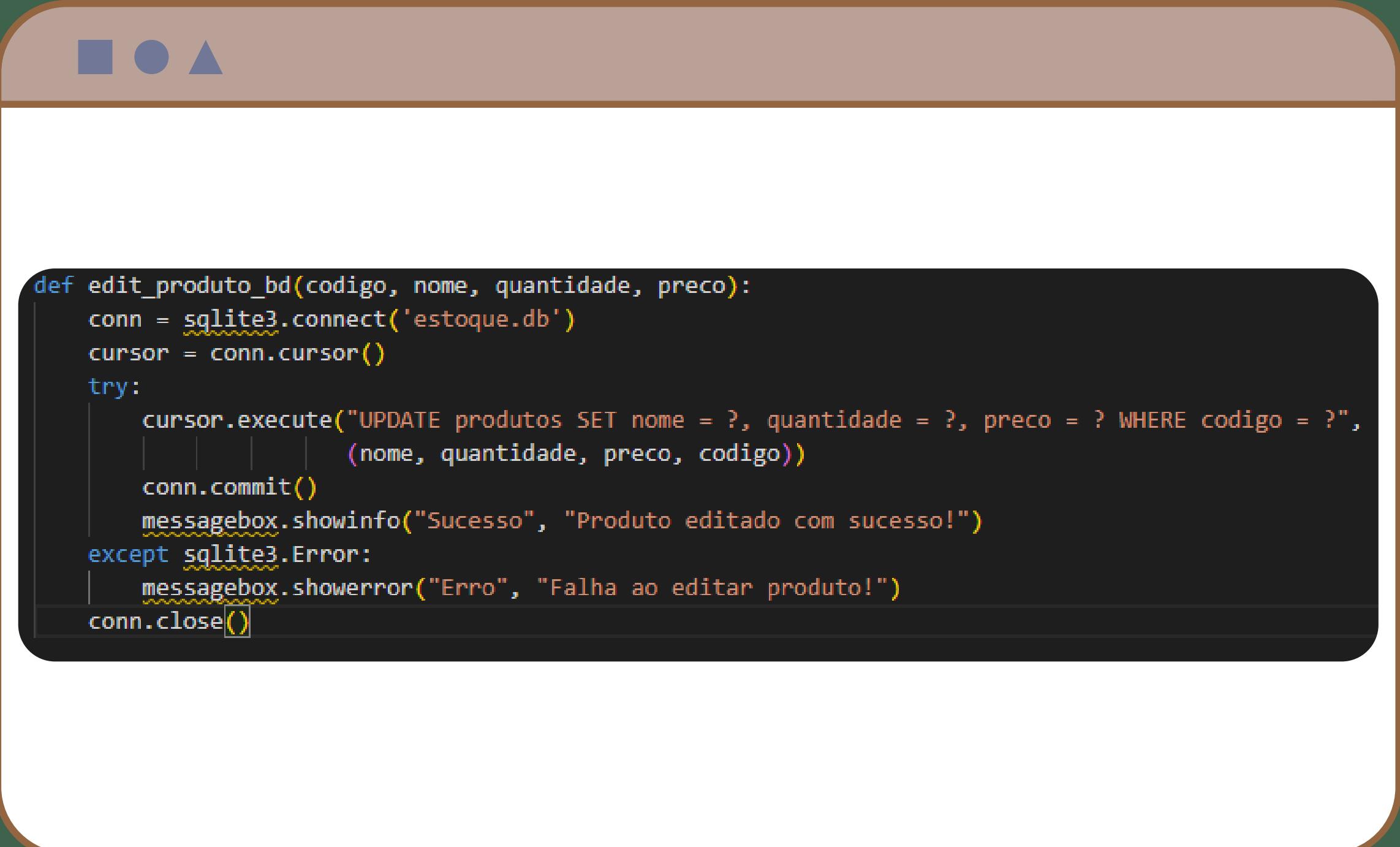
Cria um comando SQL de
UPDATE na Tabela Produtos
com os dados codigo, nome,
quantidade, preco

UPDATE

atualiza os
dados na tabela

SET

indica quais os
dados que serão
atualizados



```
def edit_produto_bd(codigo, nome, quantidade, preco):
    conn = sqlite3.connect('estoque.db')
    cursor = conn.cursor()
    try:
        cursor.execute("UPDATE produtos SET nome = ?, quantidade = ?, preco = ? WHERE codigo = ?",
                      (nome, quantidade, preco, codigo))
        conn.commit()
        messagebox.showinfo("Sucesso", "Produto editado com sucesso!")
    except sqlite3.Error:
        messagebox.showerror("Erro", "Falha ao editar produto!")
    conn.close()
```

FUNÇÕES



```
def deletar_produto_interface():
    codigo = cod_edit_entry.get()

    # Jogando dados pra função delete BD
    delete_produto_bd(codigo)

    # Limpa os campos da interface
    cod_edit_entry.delete(0, tk.END)
    nome_edit_entry.config(state='normal')
    nome_edit_entry.delete(0, tk.END)
    qtd_edit_entry.config(state='normal')
    qtd_edit_entry.delete(0, tk.END)
    preco_edit_entry.config(state='normal')
    preco_edit_entry.delete(0, tk.END)
```

Pega o dado da interface através do nome das entry, alocando este dado em uma variável. Chama a função `delete_produto_bd` dando as variáveis criadas como parâmetros.

EXCLUIR PRODUTO

Cria um comando SQL de DELETE de todos os dados na Tabela Produtos filtrando pelo código do produto

```
def delete_produto_bd(codigo):
    conn = sqlite3.connect('estoque.db')
    cursor = conn.cursor()
    try:
        cursor.execute("DELETE FROM produtos WHERE codigo = ?", (codigo,))
        conn.commit()
        if cursor.rowcount > 0:
            messagebox.showinfo("Sucesso", "Produto excluído com sucesso!")
        else:
            messagebox.showwarning("Aviso", "Produto não encontrado!")
    except sqlite3.Error:
        messagebox.showerror("Erro", "Falha ao excluir produto!")
    conn.close()
```

DELETE → deleta os dados na tabela

FROM → indica a tabela

WHERE → filtro; qual o dado deve ser decisivo no delete

.ROWCOUNT → retorna a quantidade de linhas afetadas



INTERFACE GRAFICA

```
# Aba de Buscar Produto
aba_buscar = ttk.Frame(notebook)
notebook.add(aba_buscar, text="Buscar Produto")

# Elementos da aba de buscar
# Botão para buscar produto
botao_fechar_busca = tk.Button(aba_buscar, text="X", command=exit_busca)
botao_fechar_busca.grid(row=0, column=3, columnspan=2, pady=10)
cod_busca_label = tk.Label(aba_buscar, text="Código do Produto: ")
cod_busca_label.grid(row=0, column=0, padx=10, pady=6)
cod_busca_entry = tk.Entry(aba_buscar)
cod_busca_entry.grid(row=0, column=1, padx=10, pady=6)

# Botão para buscar produto
botao_buscar = tk.Button(aba_buscar, text="Buscar Produto", command=busca_produto_interface_busca)
botao_buscar.grid(row=1, column=1, columnspan=2, pady=10)

nome_busca_label = tk.Label(aba_buscar, text="Nome do Produto: ")
nome_busca_label.grid(row=2, column=0, padx=10, pady=6)
nome_busca_entry = tk.Entry(aba_buscar, state='readonly')
nome_busca_entry.grid(row=2, column=1, padx=10, pady=6)

qtd_busca_label = tk.Label(aba_buscar, text="Quantidade do Produto: ")
qtd_busca_label.grid(row=3, column=0, padx=10, pady=6)
qtd_busca_entry = tk.Entry(aba_buscar, state='readonly')
qtd_busca_entry.grid(row=3, column=1, padx=10, pady=6)

preco_busca_label = tk.Label(aba_buscar, text="Preço do Produto: ")
preco_busca_label.grid(row=4, column=0, padx=10, pady=6)
preco_busca_entry = tk.Entry(aba_buscar, state='readonly')
preco_busca_entry.grid(row=4, column=1, padx=10, pady=6)

# Botão para adicionar produto
botao_add_busca = tk.Button(aba_buscar, text="Adicionar Produto", command=chama_add)
botao_add_busca.grid(row=5, column=1, columnspan=2, pady=10)
```

INTERFACE

```
# Configuração da interface
tela = tk.Tk()
tela.title("Controle de Estoque")

# Criando o notebook (abas)
notebook = ttk.Notebook(tela)
notebook.pack(pady=10, expand=True)
```



CONFIG ACESSO



```
def configurar_abas():
    nivel_acesso = cod_acess_entry.get()
    if nivel_acesso == '01':
        notebook.add(aba_buscar, text="Menu Usuário")
        notebook.forget(aba_acesso)
    elif nivel_acesso == '02':
        notebook.add(aba_menu, text="Menu Admin")
        notebook.forget(aba_acesso)
```

Pega o dado da interface através do nome das entry, alocando este dado em uma variável (nivel_acesso). Realiza um teste lógico para saber se o usuário tem acesso aos privilégios de Admin ou não.



Na Prática



Conclusão



Obrigado!

