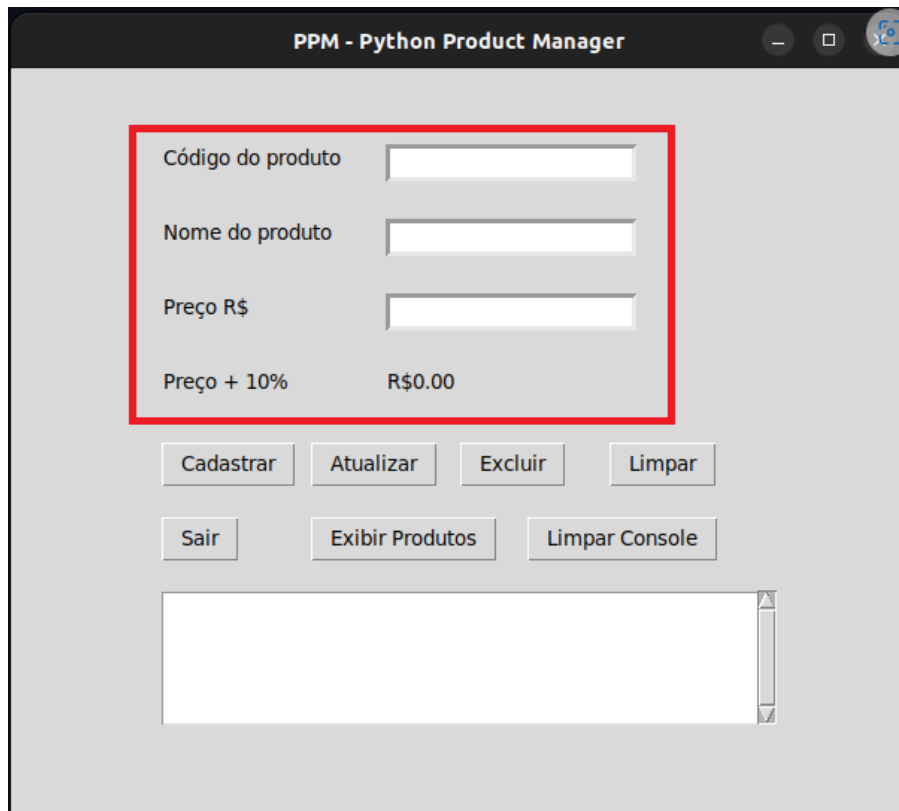


Python Product Manager - PPM

Alan Perdomo

1 - Campos para entrada dos dados (Código do produto, Nome do produto e Preço) e exibição do valor do produto com o acréscimo de 10%.



Cria rótulos (labels) e campos de entrada (entry) para código, nome, preço e porcentagem extra.

```
self.lbCodigo = tk.Label(win, text="Código do produto")
self.lbNome = tk.Label(win, text="Nome do produto")
self.lbPreco = tk.Label(win, text="Preço R$")
self.lbPorcentagemExtra = tk.Label(win, text="Preço + 10%")
```

Cria um campo de entrada para o código, nome e o preço.

```
self.txtCodigo = tk.Entry(bd=3)
self.txtNome = tk.Entry(bd=3)
self.txtPreco = tk.Entry(bd=3)
self.txtPorcentagemExtra = tk.Label(win, text="R$0.00", relief="flat")
```

Posiciona os rótulos e campos de entrada

```
self.lbCodigo.place(x=100, y=50)  
self.txtCodigo.place(x=250, y=50)
```

```
self.lbNome.place(x=100, y=100)  
self.txtNome.place(x=250, y=100)
```

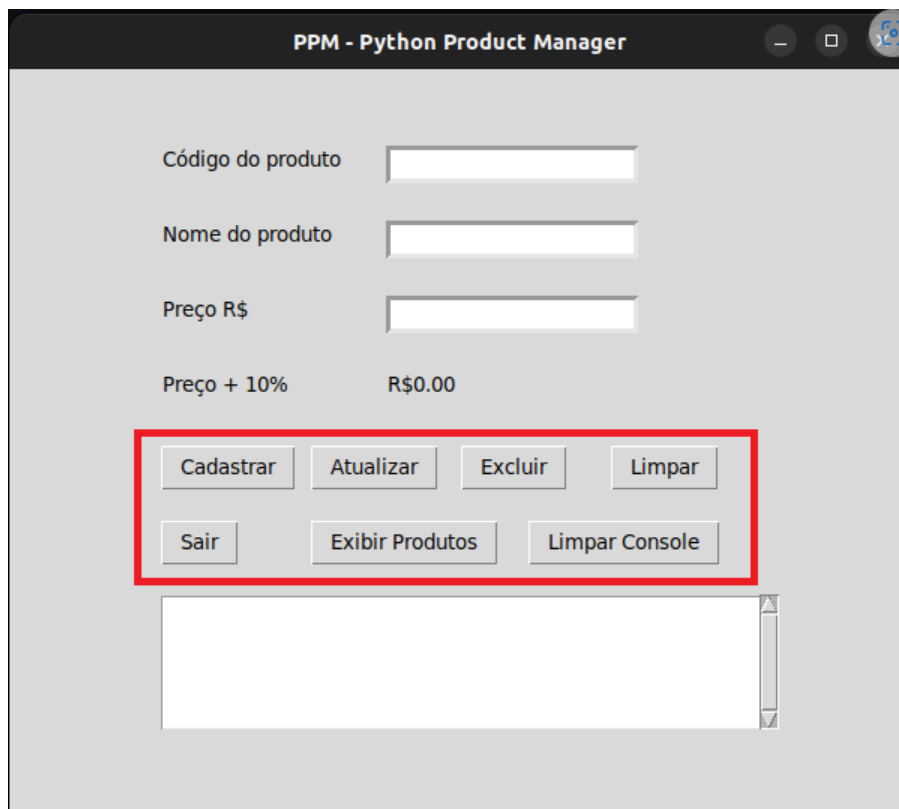
```
self.lbPreco.place(x=100, y=150)  
self.txtPreco.place(x=250, y=150)
```

```
self.lbPorcentagemExtra.place(x=100, y=200)  
self.txtPorcentagemExtra.place(x=250, y=200)
```

Associa um evento de tecla (KeyRelease) ao campo de preço para calcular a porcentagem extra.

```
self.txtPreco.bind("<KeyRelease>", self.calcularPorcentagemExtra)
```

2 - Botões do programa



Criar botões para cada ação: cadastrar, atualizar, excluir, limpar, sair, exibir produtos e limpar console.

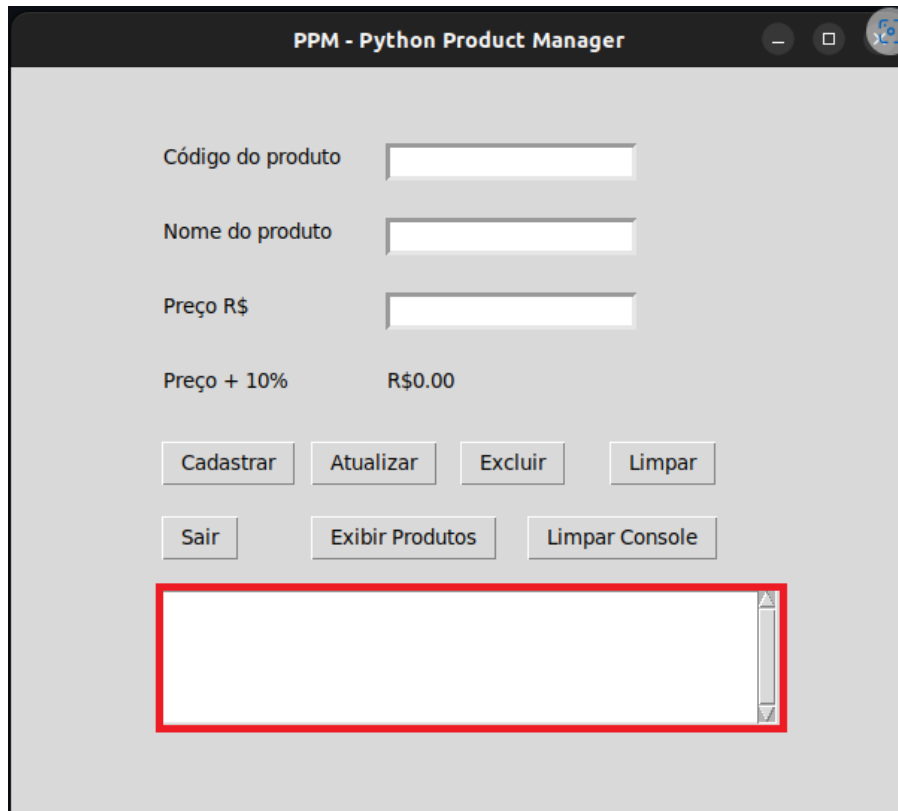
```
self.btnCadastrar = tk.Button(
```

```
win, text="Cadastrar", command=self.cadastrarProduto
)
self.btnAtualizar = tk.Button(
    win, text="Atualizar", command=self.atualizarProduto
)
self.btnExcluir = tk.Button(win, text="Excluir", command=self.excluirProduto)
self.btnLimpar = tk.Button(win, text="Limpar", command=self.limparTela)
self.btnSair = tk.Button(win, text="Sair", command=self.sair)
self.btnExibirProdutos = tk.Button(
    win, text="Exibir Produtos", command=self.exibirProdutos
)
self.btnLimparConsole = tk.Button(
    win, text="Limpar Console", command=self.limparConsole
)
```

Posicionar os botões

```
self.btnCadastrar.place(x=100, y=250)
self.btnAtualizar.place(x=200, y=250)
self.btnExcluir.place(x=300, y=250)
self.btnLimpar.place(x=400, y=250)
self.btnSair.place(x=100, y=300)
self.btnExibirProdutos.place(x=200, y=300)
self.btnLimparConsole.place(x=345, y=300)
```

3 - Console do programa



Cria um campo de texto (Text) para exibir informações e um scrollbar para rolar o texto.

```
self.console_text = tk.Text(win, wrap=tk.WORD, height=5, width=50)
self.scrollbar = tk.Scrollbar(win, command=self.console_text.yview)
self.console_text.configure(yscrollcommand=self.scrollbar.set)
```

Posicionar o console na tela e o Scrollbar

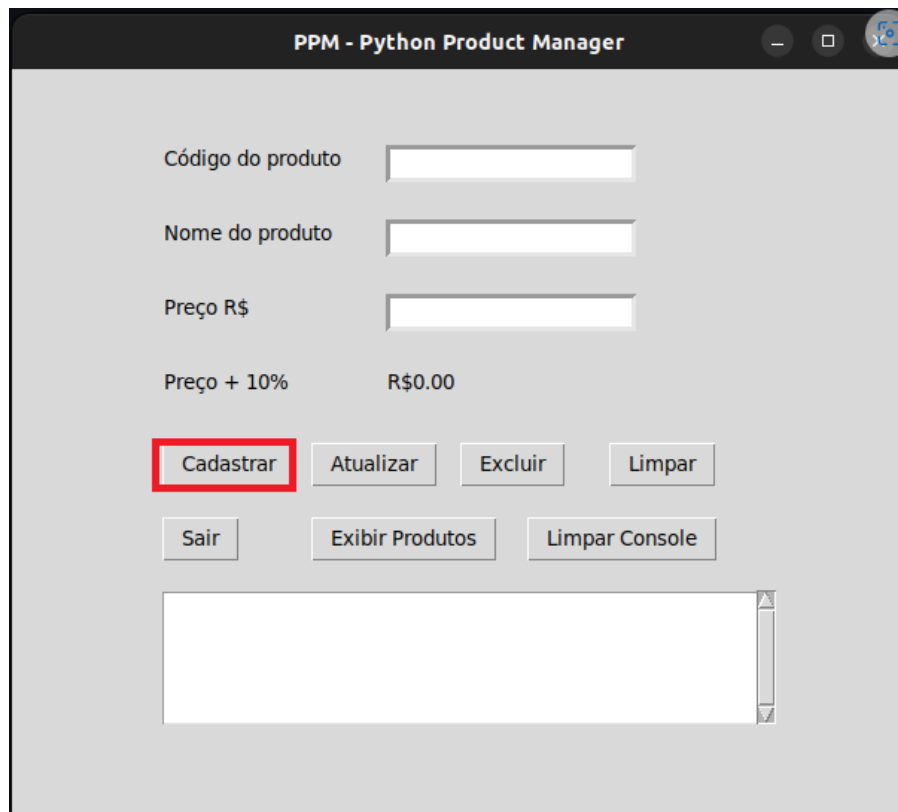
```
self.console_text.place(x=100, y=350)
self.scrollbar.place(x=500, y=350, height=90)
```

Redireciona a saída do console padrão para o campo de texto.

```
sys.stdout = TextRedirector(self.console_text, "stdout")
```

4- Botão “Cadastrar”

envia os dados informados dos campos do produto para o Banco de dados.



Método para cadastrar o produto.

```
def cadastrarProduto(self):
    try:
        codigo, nome, preco = self.lerCampos()
        preco_com_extra = preco * 1.10
        self.objBD.inserirDados(codigo, nome, preco_com_extra)
        self.txtPorcentagemExtra.config(text=f"{preco_com_extra:.2f}")
        self.limparTela()
    except Exception as e:
        print(f"Erro ao cadastrar o produto: {e}")
```

Método para ler os campos.

```
def lerCampos(self):
    try:
        codigo = int(self.txtCodigo.get())
        if codigo is None or codigo < 0:
            raise ValueError("Código inválido")
        nome = self.txtNome.get()
        preco = float(self.txtPreco.get())
        if preco is None or preco < 0:
            raise ValueError("Preço inválido")
        print("Campos lidos com sucesso")
        return (codigo, nome, preco)
    except ValueError as e:
```

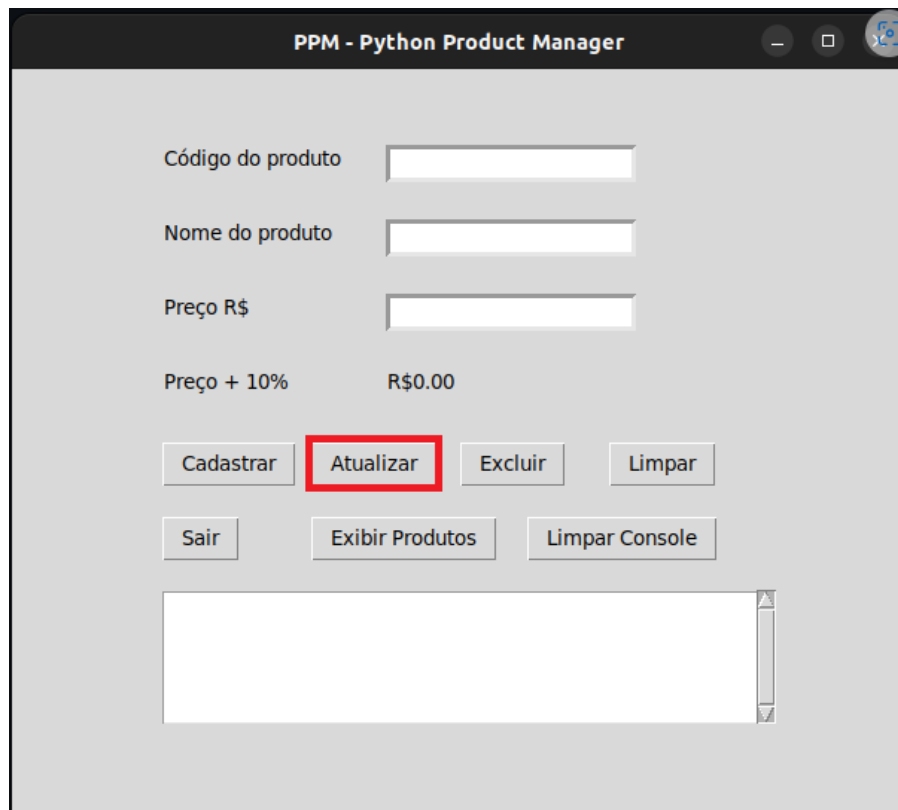
```

self.txtCodigo.config(fg="red")
self.txtPreco.config(fg="red")
self.txtCodigo.insert(tk.END, "Código inválido")
self.txtPreco.insert(tk.END, "Preço inválido")
raise ValueError("Erro ao ler os campos: ", e)

```

5 - Botão “Atualizar”

Atualiza as informações de um produto existente.



Método para atualizar o produto.

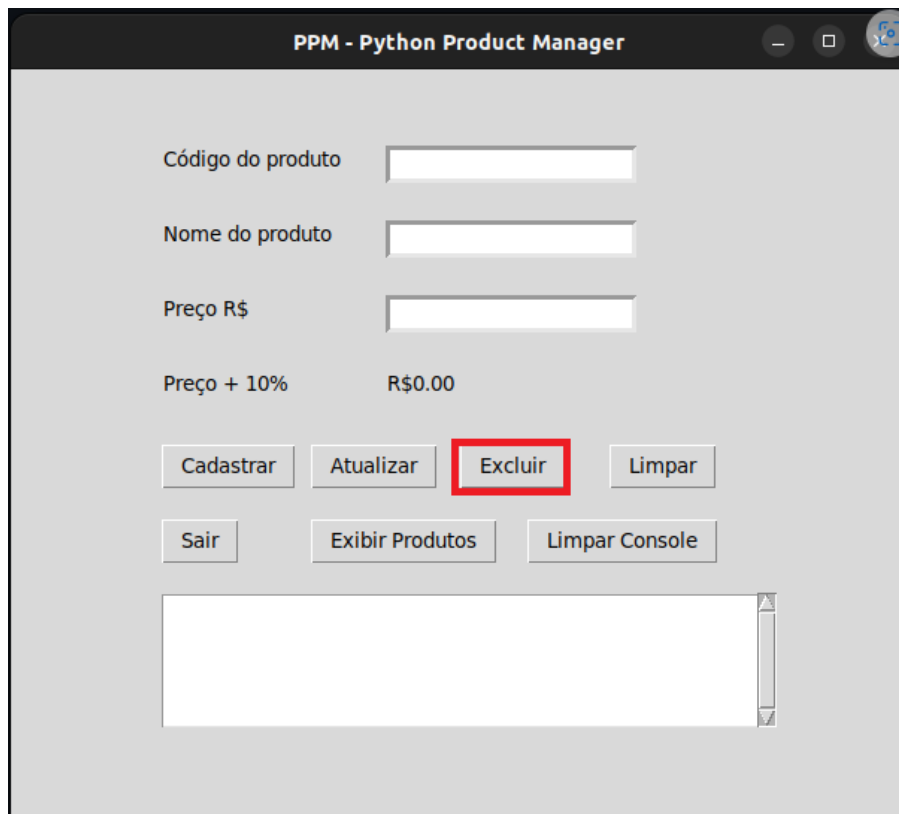
```

def atualizarProduto(self):
    try:
        codigo, nome, preco = self.lerCampos()
        preco_com_extra = preco * 1.10
        self.objBD.atualizarDados(codigo, nome, preco_com_extra)
        self.limparTela()
    except Exception as e:
        print(f"Erro ao atualizar o produto: {e}")

```

6 - Botão “Excluir”

Exclui um registro de produto do banco de dados.

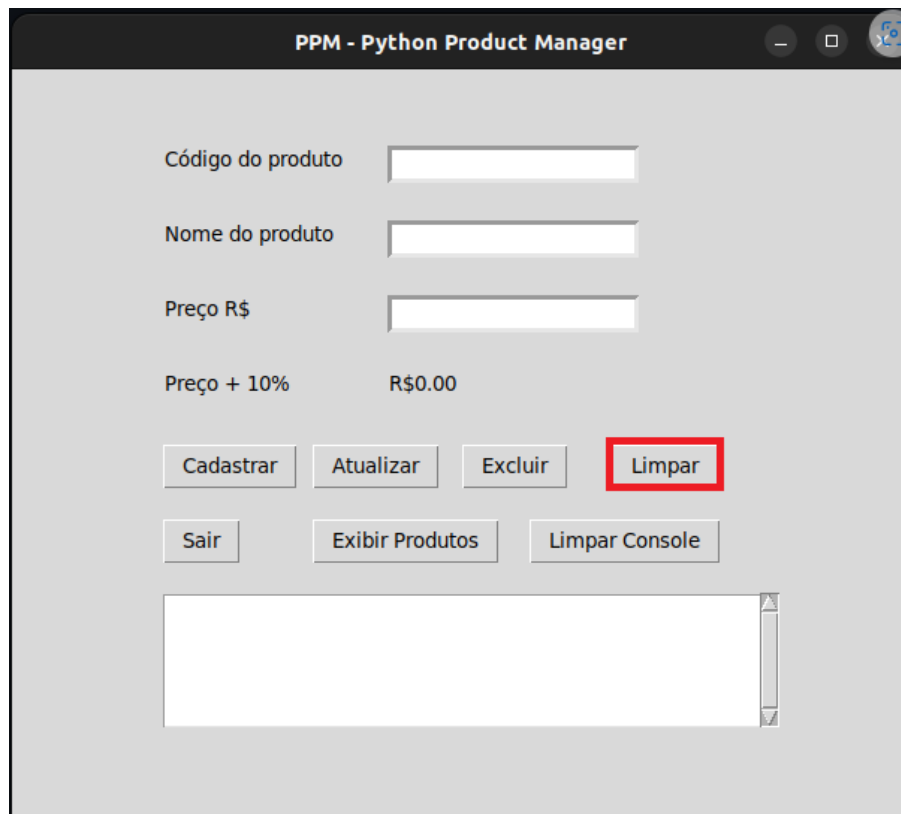


Método para excluir o produto.

```
def excluirProduto(self):
    try:
        codigo = int(self.txtCodigo.get())
        if codigo is not None and codigo > 0:
            self.objBD.excluirDados(codigo)
            self.limparTela()
        else:
            print("Nenhum código de produto informado.")
    except ValueError as ve:
        print(f"Erro ao excluir o produto: Código inválido - {ve}")
    except Exception as e:
        print(f"Erro ao excluir o produto: {e}")
```

7 - Botão “Limpar”

Limpar os campos para que o usuário possa preencher com os dados do produto.

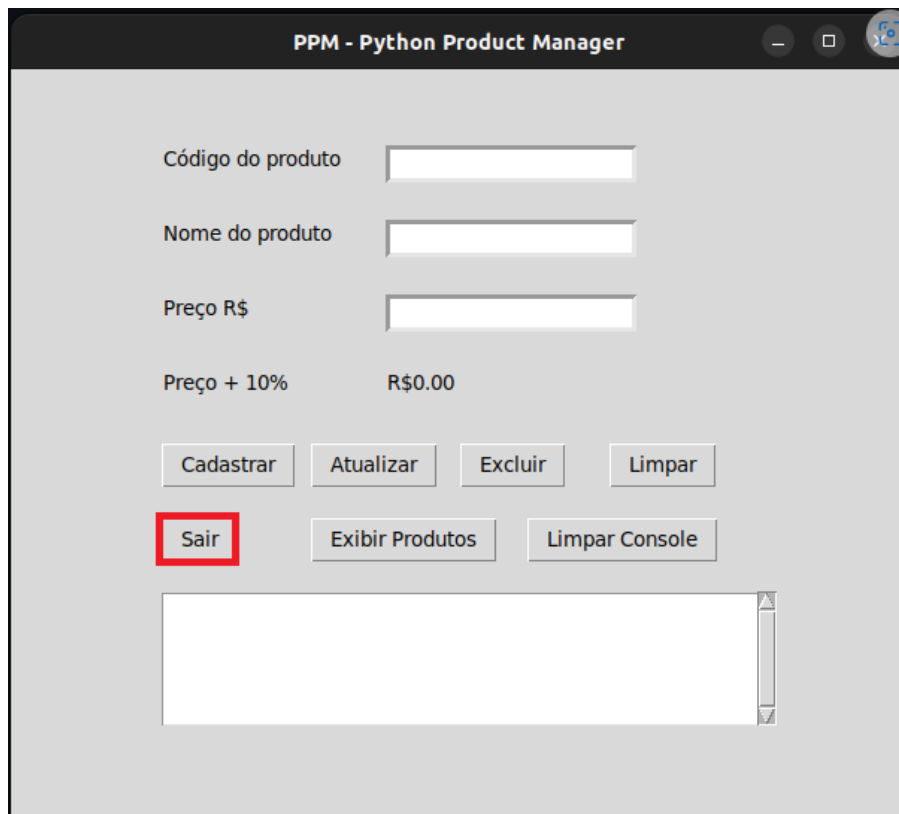


Método para limpar os campos.

```
def limparTela(self):
    try:
        if (
            self.txtCodigo.get() == ""
            and self.txtNome.get() == ""
            and self.txtPreco.get() == ""
        ):
            print("Campos ja estão limpos")
        else:
            self.txtCodigo.delete(0, tk.END)
            self.txtNome.delete(0, tk.END)
            self.txtPreco.delete(0, tk.END)
            self.txtPorcentagemExtra.config(text="R$0.00")
            print("Campos limpos com sucesso")
    except Exception as e:
        print(f"Erro ao limpar os campos: {e}")
```

8 - Botão “Sair”

Termina a execução da aplicação.

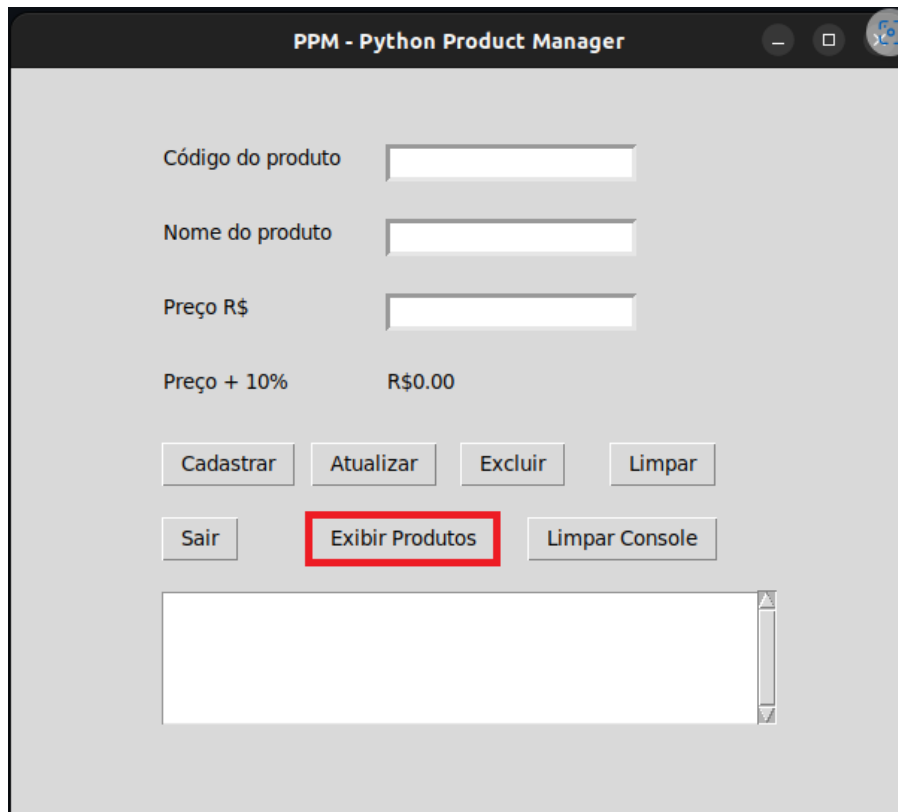


Método para sair.

```
def sair(self):  
    janela.destroy()
```

9 - Botão “Exibir Produtos”

Exibe uma nova tela com a lista dos produtos registrados no banco de dados.



Método para exibir os produtos.

```
def exibirProdutos(self):
    try:
        produtos = self.objBD.getProdutos()
        for produto in produtos:
            print(
                f"Código: {produto[0]}, Nome: {produto[1]}, Preço: {produto[2]:.2f}"
            )

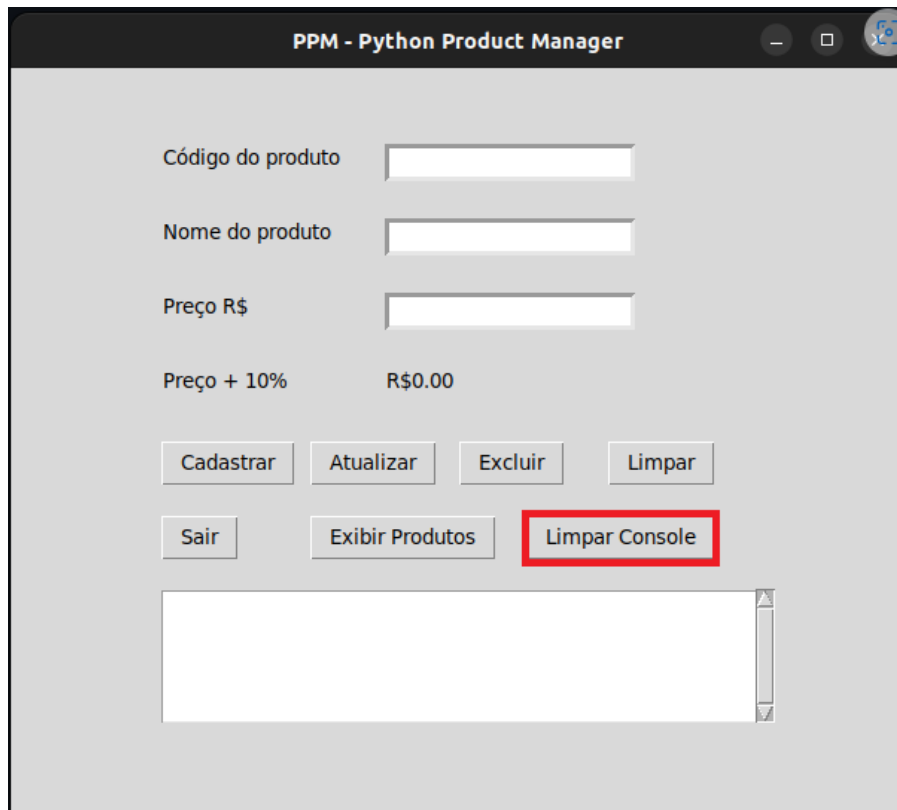
        lista_produtos = tk.Toplevel()
        lista_produtos.title("Lista de Produtos cadastrados")
        lista_text = tk.Text(lista_produtos, wrap=tk.WORD)
        lista_text.pack()

        for produto in produtos:
            lista_text.insert(
                tk.END,
                f"Código: {produto[2]}\t\tNome: {produto[1]}\t\tPreço: R${produto[3]:.2f}\n",
            )

    except Exception as e:
        print(f"Erro ao exibir os produtos: {e}")
```

10 - Botão “Limpar Console”

Limpa o campo do console na tela.



Método para limpar a console.

```
def limparConsole(self):  
    self.console_text.delete("1.0", tk.END)
```

11 - Código completo (main.py)

```
import tkinter as tk  
import sys  
import crud as crud
```

```
class PrincipalBD:  
    def __init__(self, win):
```

```
        self.objBD = crud.AppDB()
```

```
        # Cria rótulos (labels) e campos de entrada (entry) para código, nome, preço e porcentagem extra.
```

```
        self.lbCodigo = tk.Label(win, text="Código do produto")
```

```
        self.lbNome = tk.Label(win, text="Nome do produto")
```

```
        self.lbPreco = tk.Label(win, text="Preço R$")
```

```
        self.lbPorcentagemExtra = tk.Label(win, text="Preço + 10%")
```

```
        # Cria um campo de entrada para o código, nome e o preço.
```

```

self.txtCodigo = tk.Entry(bd=3)
self.txtNome = tk.Entry(bd=3)
self.txtPreco = tk.Entry(bd=3)
self.txtPorcentagemExtra = tk.Label(win, text="R$0.00", relief="flat")

# Cria botões para cada ação: cadastrar, atualizar, excluir, limpar, sair, exibir produtos e limpar console.
self.btnCadastrar = tk.Button(
    win, text="Cadastrar", command=self.cadastrarProduto
)
self.btnAtualizar = tk.Button(
    win, text="Atualizar", command=self.atualizarProduto
)
self.btnExcluir = tk.Button(win, text="Excluir", command=self.excluirProduto)
self.btnLimpar = tk.Button(win, text="Limpar", command=self.limparTela)
self.btnSair = tk.Button(win, text="Sair", command=self.sair)
self.btnExibirProdutos = tk.Button(
    win, text="Exibir Produtos", command=self.exibirProdutos
)
self.btnLimparConsole = tk.Button(
    win, text="Limpar Console", command=self.limparConsole
)

# Cria um campo de texto (Text) para exibir informações e um scrollbar para rolar o texto.
self.console_text = tk.Text(win, wrap=tk.WORD, height=5, width=50)
self.scrollbar = tk.Scrollbar(win, command=self.console_text.yview)
self.console_text.configure(yscrollcommand=self.scrollbar.set)

# Posiciona os rótulos, campos de entrada, botões e campo de texto na janela.
self.lbCodigo.place(x=100, y=50)
self.txtCodigo.place(x=250, y=50)

self.lbNome.place(x=100, y=100)
self.txtNome.place(x=250, y=100)

self.lbPreco.place(x=100, y=150)
self.txtPreco.place(x=250, y=150)

self.lbPorcentagemExtra.place(x=100, y=200)
self.txtPorcentagemExtra.place(x=250, y=200)

self.btnCadastrar.place(x=100, y=250)
self.btnAtualizar.place(x=200, y=250)
self.btnExcluir.place(x=300, y=250)
self.btnLimpar.place(x=400, y=250)
self.btnSair.place(x=100, y=300)
self.btnExibirProdutos.place(x=200, y=300)
self.btnLimparConsole.place(x=345, y=300)

```

```
self.console_text.place(x=100, y=350)
self.scrollbar.place(x=500, y=350, height=90)
```

```
# Redireciona a saída padrão para o campo de texto.
sys.stdout = TextRedirector(self.console_text, "stdout")
```

```
# Associa um evento de tecla (KeyRelease) ao campo de preço para calcular a porcentagem extra.
self.txtPreco.bind("<KeyRelease>", self.calcularPorcentagemExtra)
```

```
# Metodo para cadastrar o produto.
```

```
def cadastrarProduto(self):
    try:
        codigo, nome, preco = self.lerCampos()
        preco_com_extra = preco * 1.10
        self.objBD.inserirDados(codigo, nome, preco_com_extra)
        self.txtPorcentagemExtra.config(text=f"{preco_com_extra:.2f}")
        self.limparTela()
    except Exception as e:
        print(f"Erro ao cadastrar o produto: {e}")
```

```
# Metodo para atualizar o produto.
```

```
def atualizarProduto(self):
    try:
        codigo, nome, preco = self.lerCampos()
        preco_com_extra = preco * 1.10
        self.objBD.atualizarDados(codigo, nome, preco_com_extra)
        self.limparTela()
    except Exception as e:
        print(f"Erro ao atualizar o produto: {e}")
```

```
# Metodo para excluir o produto.
```

```
def excluirProduto(self):
    try:
        codigo = int(self.txtCodigo.get())
        if codigo is not None and codigo > 0:
            self.objBD.excluirDados(codigo)
            self.limparTela()
        else:
            print("Nenhum código de produto informado.")
    except ValueError as ve:
        print(f"Erro ao excluir o produto: Código inválido - {ve}")
    except Exception as e:
        print(f"Erro ao excluir o produto: {e}")
```

```
# Metodo para limpar os campos.
```

```
def limparTela(self):
    try:
        if (
```

```

        self.txtCodigo.get() == ""
        and self.txtNome.get() == ""
        and self.txtPreco.get() == ""
    ):
        print("Campos ja estão limpos")
    else:
        self.txtCodigo.delete(0, tk.END)
        self.txtNome.delete(0, tk.END)
        self.txtPreco.delete(0, tk.END)
        self.txtPorcentagemExtra.config(text="R$0.00")
        print("Campos limpos com sucesso")
except Exception as e:
    print(f"Erro ao limpar os campos: {e}")

```

Metodo para ler os campos.

```

def lerCampos(self):
    try:
        codigo = int(self.txtCodigo.get())
        if codigo is None or codigo < 0:
            raise ValueError("Código inválido")
        nome = self.txtNome.get()
        preco = float(self.txtPreco.get())
        if preco is None or preco < 0:
            raise ValueError("Preço inválido")
        print("Campos lidos com sucesso")
        return (codigo, nome, preco)
    except ValueError as e:
        self.txtCodigo.config(fg="red")
        self.txtPreco.config(fg="red")
        self.txtCodigo.insert(tk.END, "Código inválido")
        self.txtPreco.insert(tk.END, "Preço inválido")
        raise ValueError("Erro ao ler os campos: ", e)

```

Metodo para calcular a porcentagem extra.

```

def calcularPorcentagemExtra(self, event=None):
    try:
        preco = self.lerPreco()
        porcentagem_extra = preco * 1.1
        self.txtPorcentagemExtra.config(text=f"R${porcentagem_extra:.2f}")
    except ValueError:
        self.txtPorcentagemExtra.config(text="")

```

Metodo para ler o preço.

```

def lerPreco(self):
    return float(self.txtPreco.get())

```

Metodo para sair.

```

def sair(self):

```

```

janela.destroy()

# Metodo para exibir os produtos.
def exibirProdutos(self):
    try:
        produtos = self.objBD.getProdutos()
        for produto in produtos:
            print(
                f"Código: {produto[0]}, Nome: {produto[1]}, Preço: {produto[2]:.2f}"
            )

        lista_produtos = tk.Toplevel()
        lista_produtos.title("Lista de Produtos cadastrados")
        lista_text = tk.Text(lista_produtos, wrap=tk.WORD)
        lista_text.pack()

        for produto in produtos:
            lista_text.insert(
                tk.END,
                f"Código: {produto[0]}\t\t\tNome: {produto[1]}\t\t\tPreço: R${produto[2]:.2f}\n",
            )

    except Exception as e:
        print(f"Erro ao exibir os produtos: {e}")

# Metodo para limpar a console.
def limparConsole(self):
    self.console_text.delete("1.0", tk.END)

# Classe para redirecionar a saída padrão para o campo de texto (console).
class TextRedirector:
    def __init__(self, text_widget, tag):
        self.text_widget = (
            text_widget # O campo de texto onde a saída será redirecionada.
        )
        self.tag = tag # Um rótulo que pode ser usado para aplicar formatação ao texto inserido.

    def write(self, str):
        self.text_widget.insert(tk.END, str, (self.tag,)) # Insere o texto no console da janela do programa.
        self.text_widget.see(tk.END) # Move para o final do console, sempre exibindo a ultima atualização.

# Cria a janela principal do programa.
janela = tk.Tk()
principal = PrincipalBD(janela)
janela.title("PPM - Python Product Manager")
janela.geometry("600x500")

```

```
janela.mainloop()
```

12 - Código completo (crud.py)

```
import psycopg2
```

```
class AppDB:
```

```
    def __init__(self):
```

```
        print("Conectando ao banco de dados")
```

```
    def abrirConexao(self):
```

```
        try:
```

```
            self.connection = psycopg2.connect(
```

```
                user="postgres",
```

```
                password="alan1234",
```

```
                host="127.0.0.1",
```

```
                database="postgres",
```

```
            )
```

```
        except (Exception, psycopg2.Error) as error:
```

```
            if self.connection:
```

```
                print("Falha ao conectar ao banco de dados", error)
```

```
    def inserirDados(self, codigo, nome, preco):
```

```
        try:
```

```
            self.abrirConexao()
```

```
            cursor = self.connection.cursor()
```

```
            cursor.execute("SELECT * FROM public.produto WHERE codigo = %s", (codigo,))
```

```
            existing_product = cursor.fetchone() # Verifica se o registro existe.
```

```
            if existing_product:
```

```
                print(
```

```
                    "Produto ja existe"
```

```
                ) # Se o registro existir, Informa que o registro ja existe.
```

```
                return
```

```
            else:
```

```
                postgres_insert_query = """ INSERT INTO public."produto"("codigo","nome","preco") VALUES
(%s,%s,%s)"""
```

```
                record_to_insert = (codigo, nome, preco)
```

```
                cursor.execute(
```

```
                    postgres_insert_query, record_to_insert
```

```
                ) # Executa a query para inserir os dados com os metodos codigo, nome e preco passados.
```

```
                self.connection.commit()
```

```
                count = cursor.rowcount
```

```
                print(
```

```
                    count, "Registro inserido com sucesso"
```

```
                ) # Informa quantos registros foram inseridos com sucesso.
```

```
        except (Exception, psycopg2.Error) as error:
```



```

    if self.connection:
        print("Falha ao inserir dados", error) # Informa o erro.
finally:
    if self.connection:
        cursor.close()
        self.connection.close()
        print("Conexão fechada") # Encerra a conexão.

```

Metodo para atualizar dados

```
def atualizarDados(self, codigo, nome, preco):
```

```

    try:
        self.abrirConexao()
        cursor = self.connection.cursor()
        sql_update_query = """Update public."produto" set "nome" = %s,
        "preco" = %s where "codigo" = %s"""
        cursor.execute(
            sql_update_query, (nome, preco, codigo)
        ) # Executa a query para atualizar os dados com os parametros nome, preco e codigo passados.
        self.connection.commit() # Confirma a atualização.
        count = cursor.rowcount
        print(
            count, "Registro atualizado com sucesso! "
        ) # Informa quantos registros foram atualizados.
        print("Registro Depois da Atualização ")
        sql_select_query = """select * from public."produto"
        where "codigo" = %s"""
        cursor.execute(sql_select_query, (codigo,))
        record = cursor.fetchone()
        print(record) # Imprime o registro após a atualização.
    except (Exception, psycopg2.Error) as error:
        if self.connection:
            print("Falha ao atualizar dados", error)
finally:
    if self.connection:
        cursor.close()
        self.connection.close()
        print("Conexão fechada")

```

Metodo para excluir dados

```
def excluirDados(self, codigo):
```

```

    try:
        self.abrirConexao()
        cursor = self.connection.cursor()
        sql_delete_query = """Delete from public."produto" where "codigo" = %s"""
        cursor.execute(sql_delete_query, (codigo,))
        self.connection.commit()
        count = cursor.rowcount
        print(

```

```

        count, "Registro excluído com sucesso"
    ) # Informa quantos registros foram excluídos
    print(
        f"Produto com código {codigo} excluído com sucesso!"
    ) # Informa o código do produto que foi excluído
except (Exception, psycopg2.Error) as error:
    if self.connection:
        print("Falha ao excluir dados", error)
        # No caso de falha, informa o erro.
finally:
    if self.connection:
        cursor.close()
        self.connection.close()
        print("Conexão fechada")
        # Fecha a conexão com o banco de dados após a consulta.

```

Metodo para buscar os produtos no banco de dados

def getProdutos(self):

```

    try:
        self.abrirConexao()
        cursor = self.connection.cursor()
        cursor.execute("SELECT * FROM public.produto")
        produtos = cursor.fetchall()
        return produtos
        # Busca todos os produtos na tabela 'produto' e retorna como uma lista de registros.
    except (Exception, psycopg2.Error) as error:
        if self.connection:
            print("Falha ao buscar dados", error)
            # No caso de falha, informa o erro.
        finally:
            if self.connection:
                cursor.close()
                self.connection.close()
                print("Conexão fechada")
                # Fecha a conexão com o banco de dados após a consulta.

```