

**UNIVERSIDADE PAULISTA
CIÊNCIA DA COMPUTAÇÃO**

FELIPE MATIAS MAXIMIANO DA SILVA

GIOVANNA ALCANTARA DO NASCIMENTO

HENRIQUE NASCIMENTO DE SOUZA

KAUAN ERON CARNEIRO ARAUJO

LEONARDO DE ARAÚJO MOURA

MATHEUS DE SOUZA LIMA COSTA

Calculadora de Carbono: Uma Ferramenta para a Neutralidade Climática

SÃO PAULO

2025

FELIPE MATIAS MAXIMIANO DA SILVA – R.A. F360GE5

GIOVANNA ALCANTARA DO NASCIMENTO – R.A. H7538A9

HENRIQUE NASCIMENTO DE SOUZA – R.A. R829FJ7

KAUAN ERON CARNEIRO ARAUJO – R.A. R8425J6

LEONARDO DE ARAÚJO MOURA – R.A. R939642

MATHEUS DE SOUZA LIMA COSTA – R.A. H7663B5

Calculadora de Carbono: Uma Ferramenta para a Neutralidade Climática

Trabalho apresentado a Universidade Paulista como requisito da elaboração da nota semestral da Atividade Pratica Supervisionada do curso de Ciência da Computação.

Orientador: Prof. Luiz Santos

SÃO PAULO

2025

RESUMO

Este artigo propõe um estudo desenvolvido para entender e calcular emissões de dióxido de carbono na atmosfera com base no consumo de energia, adjunto de uma calculadora desenvolvida em *python* para estimar as emissões de gases do efeito estufa, utilizando o conceito de créditos de carbono como ferramenta de análise e tomada de decisão. Avaliamos os impactos que as atividades causam no meio ambiente, levando em conta tanto as emissões diretas quanto as indiretas de gases poluentes, e expressamos esses resultados em toneladas equivalentes de dióxido de carbono (tCO₂e). A abordagem integra gestão ambiental e tecnologia aplicada à mitigação climática para a disseminação do conceito de créditos de carbono e a compreensão da importância da avaliação dos níveis de degradação ambiental, como meio para revelar o efeito das atividades humanas e corporativas sobre o meio ambiente.

Palavras-chave: crédito de carbono, dióxido de carbono, Python, energia, degradação ambiental.

ABSTRACT

This article presents a study aimed at understanding and calculating carbon dioxide emissions based on energy consumption, supported by a Python-based calculator designed to estimate greenhouse gas emissions. The concept of carbon credits is used as a tool for analysis and decision-making. We assess the environmental impacts of human and corporate activities, considering both direct and indirect pollutant emissions, and express the results in tons of carbon dioxide equivalent (tCO₂e). The approach integrates environmental management and technology applied to climate mitigation, promoting the dissemination of the carbon credit concept and awareness of environmental degradation to reveal the effects of human activities on the environment.

Keywords: carbon credit, carbon dioxide, Python, energy, environmental degradation.

SUMÁRIO

| | |
|--|-----------|
| 1. INTRODUÇÃO | 7 |
| 1.1 O QUE É CRÉDITO DE CARBONO? | 8 |
| 2. FUNDAMENTOS DAS PRINCIPAIS TECNOLOGIAS ENVOLVIDAS NO TRABALHO..... | 9 |
| 2.1 PYTHON | 9 |
| 2.2 CUSTOMTKINTER | 10 |
| 2.3 ESTRUTURA GERAL DO CÓDIGO | 12 |
| 3. PLANO DE DESENVOLVIMENTO DA APLICAÇÃO..... | 22 |
| 3.1 FASE 1 - BASE | 22 |
| 3.2 FASE 2 - NAVEGAÇÃO (SISTEMA DE FRAMES) | 23 |
| 3.3 FASE 3 – MENU..... | 23 |
| 3.4 FASE 4 - PERGUNTAS DINÂMICAS | 24 |
| 3.5 FASE 5 – CÁLCULO DE CARBONO | 24 |
| 3.6 FASE 6 - VISUALIZAÇÃO DE HISTÓRICO | 25 |
| 3.7 FASE 7 – COMPENSAÇÃO | 25 |
| 3.8 FASE 8 - REINICIALIZAÇÃO | 25 |
| 3.9 RESUMO..... | 26 |
| 4. PROJETO (ESTRUTURA) DO PROGRAMA..... | 26 |
| 4.1 VISÃO EXECUTIVA DA ARQUITETURA | 27 |
| 4.2 MAPA DO CÓDIGO-FONTE (VISÃO MACRO) | 28 |
| 4.3 ESPECIFICAÇÃO DAS TELAS E FLUXOS | 29 |
| 4.4 FUNÇÕES (PRE/PÓS/EFEITOS)..... | 30 |
| 4.5 MODELO DE DADOS E DICIONÁRIO | 31 |
| 4.6 REGRAS DE NEGÓCIO, FÓRMULAS E LIMITAÇÕES | 32 |
| 4.7 PERSISTÊNCIA, VERSÕES E MIGRAÇÃO DE DADOS | 33 |
| 4.8 INTERFACE: DESIGN SYSTEM, TEMA E ACESSIBILIDADE..... | 33 |
| 4.9 TRATAMENTO DE ERROS E MENSAGENS AO USUÁRIO | 34 |
| 4.10 REQUISITOS NÃO FUNCIONAIS (NFRs) | 35 |
| 4.11 TESTABILIDADE, PLANO E CASOS DE TESTE | 35 |
| 4.12 SEGURANÇA E PRIVACIDADE | 36 |
| 4.13 OBSERVABILIDADE E LOGS (PLANEJADO) | 36 |
| 4.14 OPERAÇÃO E SUPORTE..... | 36 |
| 4.15 DECISÕES ARQUITETURAIS (ADRs RESUMIDAS)..... | 37 |

| | |
|---|-----------|
| 4.16 RISCOS, IMPACTOS E MITIGAÇÕES | 37 |
| 4.17 DEPENDÊNCIAS E TERCEIROS | 38 |
| 4.18 ACESSIBILIDADE..... | 38 |
| 4.19 INTERNACIONALIZAÇÃO E LOCALIZAÇÃO (I18N/L10N)..... | 38 |
| 4.20 EMPACOTAMENTO E DISTRIBUIÇÃO | 38 |
| 4.21 CONTRIBUIÇÃO E PADRÕES DE CÓDIGO..... | 38 |
| 4.22 MANUTENÇÃO E GERÊNCIA DE MUDANÇA | 39 |
| 4.23 PSEUDOCÓDIGO DE CÁLCULO E COMPENSAÇÃO | 39 |
| 4.24 EXEMPLOS DE REGISTROS JSON..... | 39 |
| 4.25 CENÁRIOS NARRATIVOS DETALHADOS..... | 40 |
| 5. RELATÓRIO – CÓDIGO NO GITHUB..... | 41 |
| 6. BIBLIOGRAFIA..... | 42 |

1. INTRODUÇÃO

Um dos maiores desafios deste século são as mudanças climáticas, impactando ecossistemas, economias e sociedades. O aumento dos gases de efeito estufa na atmosfera, resultado direto das atividades humanas, vem agravando fenômenos como o aquecimento global, a elevação do nível do mar e a intensificação de eventos climáticos extremos. Diante desse cenário, é fundamental criar políticas e instrumentos que ajudem a diminuir as emissões e favoreçam uma mudança real em direção a um modelo de desenvolvimento mais sustentável. Entre as soluções propostas, os créditos de carbono destacam-se como uma estratégia eficiente de mitigação das mudanças climáticas. Esse mecanismo permite que países, empresas e indivíduos compensem suas emissões de gases financiando projetos sustentáveis. O sistema surgiu com o Protocolo de Kyoto (1997) e ganhou força após o Acordo de Paris (2015). Cada crédito corresponde à retirada ou à redução de uma tonelada de dióxido de carbono equivalente (tCO_2e) da atmosfera. Assim, o mercado de carbono passou a conectar economia e meio ambiente, atribuindo valor financeiro à sustentabilidade e incentivando práticas de baixo impacto ambiental.

O avanço da tecnologia tem contribuído muito para esse processo, permitindo medir e acompanhar com mais precisão as emissões geradas por diferentes atividades. A automação e a programação desempenham um papel central na criação de soluções que simplificam o cálculo, o controle e a compensação das emissões de carbono. Linguagens de programação como Python possibilitam o desenvolvimento de ferramentas acessíveis, interativas e adaptáveis, transformando dados em informações úteis para a tomada de decisão. Nesse contexto, este trabalho apresenta uma calculadora de carbono desenvolvida em Python para estimar as emissões de dióxido de carbono geradas pelo uso de energia elétrica, combustíveis fósseis e outras fontes ligadas às atividades humanas e empresariais. A proposta procura facilitar o processo de cálculo e torná-lo mais didático, ajudando os usuários a entender o impacto de suas ações e a buscar formas de reduzir ou compensar suas emissões.

Além da dimensão técnica, a pesquisa também destaca o valor educacional da ferramenta, que atua como um instrumento de conscientização sobre a crise climática. Mais do que apresentar números, a calculadora incentiva uma reflexão sobre o padrão

de consumo e o uso responsável dos recursos naturais. Assim, a tecnologia passa a conectar conhecimento e ação, incentivando mudanças de comportamento e a adoção de práticas mais sustentáveis, tanto no dia a dia quanto nas empresas. A implementação da calculadora baseou-se em fatores de emissão e dados de conversão energética amplamente utilizados, assegurando precisão científica e alinhamento com metodologias de avaliação ambiental. O projeto também incorpora conceitos do mercado de créditos de carbono, relacionando os resultados a aspectos econômicos e ambientais e ajudando o usuário a entender, na prática, a busca pela neutralidade climática. O projeto busca apresentar o desenvolvimento e a importância da calculadora de carbono, destacando como ela pode apoiar a gestão ambiental e a educação para a sustentabilidade. Espera-se que a calculadora ajude a difundir a prática de monitorar e compensar emissões, estimulando uma postura mais consciente diante dos desafios climáticos e reforçando a relação entre ciência, tecnologia e responsabilidade ambiental.

1.1 O QUE É CRÉDITO DE CARBONO?

O crédito de carbono é um instrumento criado a partir do Protocolo de Kyoto, firmado em 1997, com o propósito de promover a redução das emissões de gases de efeito estufa (GEE), responsáveis por intensificar as mudanças climáticas globais. Esse mecanismo faz parte de um sistema de flexibilização ambiental, concebido para auxiliar os países signatários que possuem metas obrigatórias de mitigação das emissões poluentes.

No contexto do mercado de carbono, os créditos funcionam como uma unidade de valor ambiental, representando a não emissão de uma tonelada de dióxido de carbono (CO_2) ou de gases equivalentes à atmosfera. Dessa forma, para cada tonelada de CO_2 que deixa de ser emitida, gera-se um crédito de carbono. Quando um país ou empresa consegue atingir essa redução, recebe uma certificação emitida pelo Mecanismo de Desenvolvimento Limpo (MDL), que possibilita a comercialização desses créditos com nações ou organizações que não alcançaram suas metas de redução.

De acordo com a organização Sustainable Carbon, a geração desses créditos ocorre por meio da implementação de projetos voltados ao desenvolvimento sustentável, cujo objetivo é evitar o aumento da concentração de gases de efeito estufa na atmosfera.

Entre esses projetos destacam-se as ações de reflorestamento e conservação florestal, o incentivo ao consumo consciente, a substituição de fontes fósseis por energias renováveis e outras iniciativas que contribuem diretamente para a mitigação das mudanças climáticas.

2. FUNDAMENTOS DAS PRINCIPAIS TECNOLOGIAS ENVOLVIDAS NO TRABALHO

Este tópico apresenta os fundamentos das tecnologias Python utilizadas no desenvolvimento do sistema de cálculo de carbono, com análise detalhada do arquivo e suas bibliotecas.

2.1 PYTHON

Python é uma linguagem de programação que foi utilizada em nosso projeto, interpretada, orientada a objetos e de propósito geral, conhecida por sua sintaxe clara e facilmente legível.

Características principais:

- Sintaxe simples e intuitiva
- Tipagem dinâmica
- Interpretada (não requer compilação)
- Grande biblioteca padrão
- Ampla comunidade e ecossistema

Vantagens do Python para o Projeto:

- Rápido desenvolvimento de protótipos
- Facilidade de integração com APIs
- Bibliotecas robustas para GUI (como o Tkinter/CustomTkinter)
- Excelente para processamento de dados
- Forte documentação e suporte da comunidade

2.2 CUSTOMTKINTER

CustomTkinter é uma biblioteca voltada para UI, moderna e construída sobre Tkinter (Outra biblioteca) que fornece widgets estilizados (CTk*) com suporte nativo a temas (claro/escuro), paletas de cores e escalonamento, mantendo a compatibilidade do loop de eventos e da base Tk. Ela reduz o esforço de estilização do ttk e entrega uma aparência atualizada em diversos sistemas operacionais.

Como Instalar:

- pip install customtkinter

Importação e inicialização padrão:

- import customtkinter as ctk

2.2.1 Arquitetura e Widgets (CTk*)

Janela raiz:

ctk.CTk(): janela principal (equivalente a Tk() na biblioteca tkinter)

Widgets principais:

- ctk.CTkFrame, ctk.CTkLabel, ctk.CTkButton, ctk.CTkEntry, ctk.CTkCheckBox, ctk.CTkSwitch
- ctk.CTkOptionMenu, ctk.CTkComboBox, ctk.CTkSlider, ctk.CTkProgressBar
- ctk.CTkTabview (abas), ctk.CTkScrollableFrame (conteúdo rolável), ctk.CTkTextbox (texto multilinha)

Gerenciamento de layout:

- pack(), grid() e place() continuam disponíveis (herdado de Tk)

2.2.2 Temas, Aparência e Escalonamento

- `ctk.set_appearance_mode("dark"/"light"/"system")` para modo escuro/claro
- `ctk.set_default_color_theme(...)` para paleta de cores consistente
- `ctk.set_widget_scaling(1.0..1.5)` e `ctk.set_window_scaling(...)` para acessibilidade e alta densidade de pixels

Benefício: visual consistente sem lidar com estilos ttk e temas externos.

2.2.3 Modelo de Eventos (compatível com Tkinter)

- `command=` em botões e outros widgets para utilizar funções.
- `bind("<Event>", callback)` permanece disponível

Vantagem: migração simples de callbacks existentes; nenhuma mudança no fluxo.

2.2.4 Comparação de Uso (Tkinter / CustomTkinter)

Antes (Tkinter/ttk):

```
# janela = tk.Tk()
# btnComprar = ttk.Button(janela, text="Comprar", command=comprarCredito)
```

Depois (CustomTkinter):

```
import customtkinter as ctk
ctk.set_appearance_mode("dark")
ctk.set_default_color_theme("green")
janela = ctk.CTk()
btnComprar = ttk.Button(janela, text="Comprar", command=comprarCredito)
```

Abas com CTkTabview:

```
tabs = ctk.CTkTabview(janela)
tabs.add("Marketplace")
tabs.add("Minha Carteira")
```

Área rolável e texto:

```
panel = ctk.CTkScrollableFrame(janela)
txt = ctk.CTkTextbox(panel)
```

Por que utilizamos o CustomTkinter

A biblioteca CustomTkinter nos permite uma UX moderna com modo escuro (dark mode) nativo para um sistema voltado a menor fadiga visual. Redução de código para estilização, simplificando-a e acelerando entrega e manutenção. Uma consistência visual entre plataformas sem muitos ajustes. Escalonamento e acessibilidade prontos, importante para dashboards e tabelas extensas.

2.3 ESTRUTURA GERAL DO CÓDIGO

O arquivo app.py implementa uma aplicação desktop para cálculo de créditos de carbono utilizando Tkinter para a interface gráfica e outras bibliotecas Python para funcionalidades específicas.

Organização do código:

1. Importação de bibliotecas
2. Definição de constantes
4. Métodos de interface
5. Funções
6. Inicialização da aplicação

2.3.1 Bibliotecas Utilizadas no Projeto

CustomTkinter:

```
import customtkinter as ctk
```

- Biblioteca de GUI moderna baseada em Tkinter (CTk* widgets, temas e dark mode)

Messagebox:

```
from tkinter import messagebox
```

- Caixas de diálogo (info/erro/confirmação), caso não use alternativa CTk de terceiros

JSON:

```
import json
```

- Manipulação de dados JSON

Datetime

```
import datetime
```

- Manipulação de datas e horas

OS:

```
import os
```

- Interação com sistema operacional

2.3.2 ESTRUTURA DO PROJETO APP.PY

Diferentemente de uma arquitetura orientada a objetos, o app.py utiliza uma abordagem com variáveis globais para gerenciar estado, combinada com funções para organizar a lógica do aplicativo.

Estrutura geral:

1. Importações e configurações iniciais
2. Carregamento de dados JSON (perguntas.json)
3. Definição de constantes (cores, fontes)
4. Configuração da janela principal e cabeçalho
5. Variáveis globais de estado
6. Funções auxiliares (carregar/salvar histórico)
7. Funções de interface (telas e navegação)

8. Funções de lógica de negócio (cálculos e validações)
9. Loop principal (mainloop)

Exemplo da estrutura inicial:

```
```python
import json
import customtkinter as ctk
import os
from datetime import datetime
import tkinter.messagebox as messagebox

Carregar dados de perguntas
with open('perguntas.json', 'r', encoding='utf-8') as arq:
 vListaPerguntas = json.load(arq)

Constantes de estilo
corFundo = "#ffffff"
corBtn = "#2e7d32"
...demais cores e fontes...

Configuração da janela principal
ctk.set_appearance_mode("light")
ctk.set_default_color_theme("green")

janela = ctk.CTk()
janela.title("Calculadora de Carbono")
janela.geometry("900x1000")

Variáveis globais de estado
usuario = None
entradas = []
escolhas = []
ultimo_calculo = None
selected_project_var = None
```

```

Variáveis Globais de Estado

O aplicativo gerencia seu estado através de variáveis globais declaradas no escopo do módulo:

usuario: str | None

- Armazena o nome do usuário logado

entradas: List[CTkEntry]

- Lista de campos de entrada para valores numéricos das perguntas

escolhas: List[StringVar]

- Lista de variáveis Tkinter para os radiobuttons (Sim/Não)

ultimo_calculo: Dict | None

- Dicionário com os resultados do último cálculo realizado
{"tipo": str, "total_co2": float, "creditos": float,
"valor_reais": float, "mudas": float}

selected_project_var: StringVar | None

- Variável para o projeto de compensação selecionado

project_desc_label: CTkLabel | None

- Widget que exibe a descrição do projeto

compensation_cost_label: CTkLabel | None

- Widget que exibe o custo de compensação

frame_menu: CTkFrame

- Frame principal para a tela de menu

frame_perguntas: CTkFrame

- Frame principal para a tela de perguntas/cálculo

Organização por Funções Modulares

O código é organizado em funções que representam diferentes responsabilidades:

a) Funções de Persistência:

```

```python
def carregar_historico() -> List[Dict]:
 """Carrega histórico de cálculos do arquivo JSON"""
 if os.path.exists(HIST_FILE):
 with open(HIST_FILE, 'r', encoding='utf-8') as f:
 return json.load(f)
 return []

def salvar_historico(entrada: Dict) -> None:
 """Adiciona nova entrada ao histórico e salva"""
 historico = carregar_historico()
 historico.append(entrada)
 with open(HIST_FILE, 'w', encoding='utf-8') as f:
 json.dump(historico, f, ensure_ascii=False, indent=2)
```

```

b) Funções de Navegação:

```

```python
def mostrar_login():
 """Exibe tela de login inicial"""
 # Cria interface de login com campo de nome
 # Valida entrada e chama mostrar_menu()

def mostrar_menu():
 """Exibe menu principal após login"""
 # Limpa frames anteriores
 # Cria botões para escolher tipo (pessoas/empresas)

def mostrar_perguntas(tipo: str):
 """Troca para tela de perguntas"""
 frame_menu.pack_forget()
 frame_perguntas.pack(fill="both", expand=True)
 carregar_perguntas(tipo)

def mostrar_historico(tipo=None):
 """Exibe histórico de cálculos"""
 # Carrega e exibe registros do histórico

```

```
Permite voltar para tela anterior
```

```
```
```

c) Funções de Interface Dinâmica:

```
```python
```

```
def carregar_perguntas(tipo: str):
 """Cria dinamicamente os widgets para as perguntas"""
 # Limpa frame_perguntas
 # Carrega perguntas do JSON baseado no tipo
 # Cria cards de perguntas com radiobuttons e entries
 # Adiciona botões de ação (calcular, voltar, histórico)
```

```
def alternar_campo(valor: str, entry_widget: CTkEntry):
 """Habilita/desabilita campo de entrada baseado em Sim/Não"""
 if valor == "Sim":
 entry_widget.configure(state="normal")
 else:
 entry_widget.delete(0, "end")
 entry_widget.configure(state="disabled")
````
```

d) Funções de Lógica de Negócio:

```
```python
```

```
def calcular_co2(tipo: str):
 """Calcula emissões totais de CO2"""
 global ultimo_calculo
 total_co2 = 0

 # Percorre todas as perguntas
 # Multiplica valores por fatores de emissão
 # Converte para créditos e valores monetários
 # Salva no histórico
 # Atualiza ultimo_calculo e interface
```

```
def compensar_emissao():
 """Registra compensação de emissões"""
 global ultimo_calculo
```

```
Valida se há cálculo prévio
Confirma com o usuário
Registra compensação no histórico
Atualiza interface
```
```

e) Função de Reinicialização:

```
```python  
def reiniciar_app():
 """Reinicia o aplicativo para o estado inicial"""
 global usuario, entradas, escolhas, ultimo_calculo
 # Confirma com usuário
 # Reseta variáveis globais
 # Destroi widgets (exceto cabeçalho)
 # Recria frames principais
 # Volta para tela de login
```
```

Fluxo de Dados e Estado

O fluxo de dados segue este padrão:

1. Entrada do Usuário
↓
2. Evento Tkinter (click, digitação)
↓
3. Callback de função invocado
↓
4. Leitura de variáveis globais
↓
5. Processamento de lógica
↓

6. Atualização de variáveis globais

↓

7. Atualização da interface (configure, pack/forget)

↓

8. Persistência em JSON (se necessário)

Exemplo prático:

```
```python
Usuário clica em "Calcular"
btn_calcular.configure(command=lambda t=tipo: calcular_co2(t))

Função calcular_co2 é invocada
def calcular_co2(tipo):
 global ultimo_calculo # Acessa estado global
 ↓
 # Lê valores dos widgets
 for i, pergunta in enumerate(perguntas_lista):
 escolha = escolhas[i].get() # StringVar global
 entrada_valor = entradas[i] # CTkEntry global
 ↓
 # Processa cálculos
 total_co2 += valor * pergunta["calculo_co2"]
 ↓
 # Atualiza estado global
 ultimo_calculo = {"tipo": tipo, "total_co2": total_co2, ...}
 ↓
 # Atualiza interface
 resultado_label.configure(text=f"Total: {total_co2:.2f} kg CO2")
 ↓
 # Persiste em JSON
 salvar_historico(entry)
```

```

Gerenciamento de Frames

A navegação entre telas é feita através de dois frames principais:

frame_menu: Tela de menu/escolha

frame_perguntas: Tela de cálculo/histórico

Padrão de troca de tela:

```
```python
def trocar_tela():
 frame_atual.pack_forget() # Oculta frame atual
 frame_destino.pack(fill="both", expand=True) # Mostra destino
 # Limpa e reconstrói widgets do destino
```

```

Exemplo real:

```
```python
def mostrar_perguntas(tipo):
 frame_menu.pack_forget() # Esconde menu
 frame_perguntas.pack(...) # Mostra perguntas
 carregar_perguntas(tipo) # Reconstrói interface

def mostrar_menu():
 frame_perguntas.pack_forget() # Esconde perguntas
 for widget in frame_menu.winfo_children():
 widget.destroy() # Limpa menu anterior
 # Reconstrói widgets do menu
 frame_menu.pack(fill="both", expand=True)
```

```

Integração com Dados JSON

O aplicativo utiliza dois arquivos JSON:

a) perguntas.json (leitura):

```
```python
with open('perguntas.json', 'r', encoding='utf-8') as arq:
 vListaPerguntas = json.load(arq)

Acesso aos dados
perguntas_pessoas = vListaPerguntas['pergunta_pessoas']
perguntas_empresas = vListaPerguntas['pergunta_empresas']
```

```

```

b) historico.json (leitura/escrita):
```python
HIST_FILE = 'historico.json'

def carregar_historico():
 """Carrega lista de cálculos anteriores"""
 # Retorna lista de dicionários

def salvar_historico(entrada):
 """Adiciona novo cálculo ao arquivo"""
 historico = carregar_historico()
 historico.append(entrada) # Adiciona ao final
 # Salva lista completa
```

```

Padrão de Callbacks e Eventos

Widgets Tkinter utilizam callbacks para responder a eventos:

```

```python
Botão com comando direto
btn_calcular = ctk.CTkButton(
 parent,
 text="Calcular",
 command=lambda: calcular_co2('pessoas') # Lambda para passar args
)

Radiobutton com callback que passa widget
btn_sim = ctk.CTkRadioButton(
 frame,
 text="Sim",
 variable=escolha_var,
 command=lambda v="Sim", i=i: alternar_campo(v, entradas[i])
)

Entry com bind de tecla
name_entry.bind("<Return>", lambda event: validar_e_continuar())
```

```

3. PLANO DE DESENVOLVIMENTO DA APLICAÇÃO

O desenvolvimento seguiu metodologia incremental onde cada fase produzia uma aplicação funcional. Esta abordagem reduz risco ao garantir que sempre há uma base operacional mesmo se desenvolvimento for interrompido. Cada incremento adiciona funcionalidade específica que pode ser testada independentemente antes de prosseguir para próxima fase. A ordem de implementação foi cuidadosamente planejada para que cada fase dependesse apenas de funcionalidades já implementadas. Começar com estruturas simples antes de complexas permitiu validar conceitos fundamentais cedo. Implementar interface antes de lógica complexa garantiu que dados podiam ser capturados adequadamente antes de processá-los.

3.1 FASE 1 - BASE

A primeira fase estabeleceu infraestrutura básica da aplicação. Importações de bibliotecas foram feitas primeiro para detectar dependências faltantes imediatamente. Arquivo perguntas.json foi carregado no início para validar existência e formato correto antes de qualquer interface ser criada.

Constantes visuais (cores, fontes) foram definidas centralizadamente facilitando manutenção futura. Configuração global do tema CustomTkinter garantiu consistência visual em todos os widgets. Janela principal foi criada com dimensões fixas apropriadas para monitores modernos. Cabeçalho fixo foi implementado primeiro como âncora visual permanente.

Esta parte do projeto provou-se fundamental pois estabeleceu padrões que guiariam todo desenvolvimento subsequente. Decisões de estilo tomadas aqui propagaram automaticamente para todas as telas futuras.

3.2 FASE 2 - NAVEGAÇÃO (SISTEMA DE FRAMES)

Sistema de navegação foi implementado antes de conteúdos específicos permitindo testar transições entre telas vazias. Duas variáveis globais de frames principais foram criadas: frame_menu e frame_perguntas. Padrão pack/pack_forget foi estabelecido para alternar visibilidade mantendo apenas um frame ativo por vez.

Variáveis globais de estado (usuario, entradas, escolhas, ultimo_calculo) foram declaradas permitindo que funções futuras soubessem onde encontrar estado compartilhado. Esta decisão de usar estado global, embora controversa em projetos grandes, mostrou-se perfeitamente adequada para escopo limitado desta aplicação.

Navegação robusta nesta fase facilitou enormemente fases posteriores pois adicionar novas telas tornou-se trivial seguindo padrão já estabelecido.

3.3 FASE 3 – MENU

Tela de login foi primeira funcionalidade real implementada por ser ponto de entrada natural do usuário. Interface minimalista com logo emoji, título e campo único de entrada direcionou foco para ação necessária. Validação multicamadas verifica comprimento mínimo e ausência de números no nome.

Feedback de erro através de label colorido é menos intrusivo que popups modais. Binding de tecla Enter permite submissão rápida seguindo convenções esperadas. Nome validado é armazenado em variável global para uso em saudações e histórico. Login estabeleceu padrão de qualidade que permeou toda aplicação: validação imediata, feedback claro, comportamentos esperados implementados.

Menu principal representa primeiro ponto de decisão onde fluxo bifurca entre tipos de usuário. Função mostrar_menu sempre reconstrói interface do zero

prevendo acúmulo de widgets ocultos. Saudação personalizada com nome do usuário cria conexão pessoal melhorando experiência percebida. Menu simples mas efetivo estabeleceu padrão de interfaces limpas focadas em ações claras do usuário.

3.4 FASE 4 - PERGUNTAS DINÂMICAS

Geração dinâmica de perguntas representa coração da aplicação. Expressão condicional seleciona conjunto apropriado de perguntas do JSON baseado em tipo escolhido. CTkScrollableFrame permite qualquer número de perguntas sem comprometer usabilidade.

Loop com enumerate cria cards individuais para cada pergunta mantendo estrutura visual consistente. StringVar gerencia estado de radiobuttons com binding bidirecional automático. Callback condicional habilita/desabilita entry baseado em resposta Sim/Não fornecendo feedback visual claro.

3.5 FASE 5 – CÁLCULO DE CARBONO

Função calcular_co2 transforma entrada do usuário em informação significativa. Três listas paralelas (perguntas, escolhas, entradas) são percorridas sincronizadamente garantindo correspondência correta. Bloco try-except protege contra entradas inválidas com feedback específico sobre qual pergunta tem problema.

Valor numérico multiplicado por fator de emissão da pergunta acumula em total. Conversões múltiplas (toneladas, créditos, reais, mudas) fornecem perspectivas variadas facilitando compreensão. Resultado exibido imediatamente em verde com formatação apropriada.

Cálculo robusto com validação adequada e feedback claro estabeleceu confiança na precisão do sistema.

Funções carregar_historico e salvar_historico implementam persistência com graceful degradation: retorna lista vazia se arquivo não existe ao invés de erro. Padrão

append carrega histórico existente, adiciona nova entrada, salva tudo de volta preservando dados anteriores.

Estrutura de entrada incluiu timestamp, tipo, usuário e todos resultados numéricos. Campo booleano 'compensado' prepara para funcionalidade futura. Salvamento automático após cálculo elimina necessidade de usuário lembrar de salvar.

Persistência transparente garantiu que dados nunca são perdidos acidentalmente.

3.6 FASE 6 - VISUALIZAÇÃO DE HISTÓRICO

Interface de histórico reutiliza frame_perguntas economizando recursos. CTkScrollableFrame é crítico pois histórico cresce indefinidamente. Função reversed() mostra cálculos recentes primeiro reconhecendo que são mais relevantes.

Cada entrada renderizada como card visual com informações completas. Botão de limpar histórico requer confirmação prevenindo exclusões acidentais. Navegação de volta condicional retorna à tela de origem apropriada. Histórico bem implementado forneceu valor adicional significativo permitindo usuários rastrearem progresso ao longo do tempo.

3.7 FASE 7 – COMPENSAÇÃO

Sistema de compensação foi uma das últimas funcionalidades feitas por depender de todas anteriores. Dicionário estruturado de projetos facilita adicionar novos sem modificar código. CTkOptionMenu integrado na tela de perguntas aparece naturalmente após resultado.

Descrição e custo atualizam dinamicamente via callback quando seleção muda.

Messagebox solicita confirmação final antes de ação irreversível. Nova entrada no histórico com flag compensado=True distingue de cálculos simples. Compensação completa o ciclo: calcular, compreender, agir.

3.8 FASE 8 - REINICIALIZAÇÃO

Botão no header sempre visível com ícone universal de reiniciar. Confirmação previne resets acidentais. Função reseta sistematicamente todas variáveis globais para valores iniciais. Loop destrói todos widgets exceto header preservando botão de reinicialização. Frames principais recriados vazios preparando para novo conteúdo.

Reinicialização permite múltiplas sessões sem reiniciar programa.

3.9 RESUMO

Simplicidade primeiro garantiu funcionalidade em cada estágio. Feedback imediato mantem o usuário informado constantemente. Validação no ponto de entrada (fail-fast) detectou problemas cedo. Persistência automática eliminou categoria de erros. Interface progressiva revelou elementos apenas quando relevantes prevenindo sobrecarga cognitiva.

Ordem provou-se eficiente, cada fase testável independentemente. Problemas isolados facilitaram depuração. Progresso tangível manteve motivação. Aproximadamente 700 linhas de código Python bem estruturado. Varias funções principais com auxiliares. Dois arquivos JSON externos. Três telas navegáveis.

Software efetivo não requer sempre arquiteturas complexas ou tecnologias de ponta. Escolhas pragmáticas baseadas em requisitos reais, implementação cuidadosa seguindo princípios sólidos e atenção à experiência do usuário produzem resultados excelentes com recursos modestos. Este projeto demonstra desenvolvimento ágil e eficiente de aplicações desktop através de abordagem incremental bem planejada.

4. PROJETO (ESTRUTURA) DO PROGRAMA

Este tópico documenta, de forma extensa e aprofundada, a estrutura do programa da aplicação “Calculadora de Carbono”, cujo núcleo encontra-se no arquivo app.py.

O objetivo é prover um material técnico completo, redigido no estilo de um capítulo de documentação (docx), para orientar compreensão, manutenção, evolução técnica e auditoria do sistema. A narrativa percorre arquitetura, módulos (atuais e planejados), fluxos de uso, UI, dados e regras, requisitos não funcionais, testes, riscos, decisões e roadmap. O foco é a versão atual (abordagem procedural) e o caminho recomendado para modularização em camadas.

A aplicação fornece uma calculadora de emissões de CO₂ para dois públicos: pessoas físicas e empresas. A lógica central é um questionário dinâmico carregado de perguntas.json, cuja resposta resulta em um total de emissões (kg de CO₂),

convertido para créditos de carbono (tCO_2), valor monetário estimado e equivalência em mudas. O sistema permite, opcionalmente, registrar uma compensação com base em projetos pré-definidos e persiste todo o histórico localmente em historico.json.

Metas de engenharia:

- Simplicidade: base procedural, estado global controlado, funções coesas.
- Usabilidade: navegação clara em janela única, UI consistente e responsiva.
- Portabilidade: dependências mínimas (CustomTkinter), JSON local, sem DB.
- Evolutividade: caminho de modularização em camadas (UI/serviços/dados/utilitários).
- Transparência: persistência legível por humanos (JSON), docstring e documentação.

Escopo funcional:

- Login com validação de nome.
- Menu com bifurcação de tipo (pessoas/empresas).
- Questionário dinâmico com habilitação condicional de entradas.
- Cálculo de emissões e conversões ($kg \rightarrow tCO_2 \rightarrow$ créditos $\rightarrow R\$$ e mudas).
- Compensação opcional por projeto (com preço e descrição).
- Histórico com listagem, limpeza e confirmação.
- Reinicialização completa do app preservando header.

4.1 VISÃO EXECUTIVA DA ARQUITETURA

A arquitetura atual é procedural, orientada a eventos (event-driven) e apoiada no loop principal do CustomTkinter. O app consolidado em app.py assume quatro camadas lógicas implícitas:

Apresentação (UI):

- Construção de janela (CTk), frames principais, widgets, layout visual.
- Padrão de navegação pack/pack_forget para alternância entre telas.
- Cards de perguntas em CTkScrollableFrame, áreas de resultado e

compensação.

Aplicação (Controle):

- Callbacks e orquestração de fluxo (mostrar_login, mostrar_menu, mostrar_perguntas, carregar_perguntas, mostrar_historico).
- Validações de entrada e encadeamento de ações (calcular_co2 → persistir → atualizar UI).

Domínio (Regras de Negócio):

- Fórmulas de cálculo de emissões: soma de (valor * fator de emissão).
- Conversões para tCO₂, créditos, R\$ e mudas.
- Cálculo de custo de compensação por projeto (créditos * preço).

Dados (Persistência):

- Leitura de perguntas.json no arranque.
- Append em historico.json para registrar cálculos/compensações (UTF-8, indentado).

Estado global no escopo do módulo:

- usuario, ultimo_calculo, entradas[], escolhas[] e variáveis de UI (labels e StringVars).
- HIST_FILE, dicionário vListaPerguntas, dicionário de project_types.

4.2 MAPA DO CÓDIGO-FONTE (VISÃO MACRO)

O app.py organiza o ciclo de vida completo:

Bootstrapping:

- Imports, leitura de perguntas.json, definição de tema, cores e fontes.
- Janela principal, header fixo, botão “↺ Reiniciar”.
- Criação dos frames centrais: frame_menu e frame_perguntas.

Telas/Navegação:

- mostrar_login(): entrada do usuário e validação.
- mostrar_menu(): saudação personalizada e bifurcação (pessoas/empresas).
- mostrar_perguntas(tipo): encaminha para carregar_perguntas.
- mostrar_historico(tipo?): reutiliza frame_perguntas para listagem.

UI Dinâmica:

- carregar_perguntas(tipo): constrói cards por pergunta, define radiobuttons Sim/Não (com alternar_campo) e entrada numérica condicional; cria bloco de resultado, seção de compensação e botões de ação.

Regras/Persistência:

- calcular_co2(tipo): valida entradas, aplica fatores, converte unidades, salva histórico (compensado=false), atualiza ultimo_calculo e UI.
- compensar_emissao(): calcula custo, confirma, salva histórico (compensado=true).
- carregar_historico/salvar_historico: utilitários de I/O JSON.

Sistema:

- alternar_campo(): habilitar/desabilitar entrada com base em Sim/Não.
- reiniciar_app(): confirmação, reset de estado, limpeza de UI, retorno ao login.
- mainloop(): mantém a aplicação responsiva a eventos.

4.3 ESPECIFICAÇÃO DAS TELAS E FLUXOS

Tela de Login:

- Elementos: logo emoji, título, subtítulo, entry de nome, erro contextual, botão “Começar”.
- Comportamentos: Enter para submeter; validações (mín. 2 chars; sem dígitos).
- Transição: sucesso → mostrar_menu(); erro → label vermelho e borda do entry.

Menu Principal:

- Saudação “Seja Bem-Vindo, {usuario}!”.
- Botões: “Perguntas para Pessoas” e “Perguntas para Empresas”.
- Ação: cada botão chama mostrar_perguntas(tipo).

Tela de Perguntas (dinâmica):

- Cards: número da pergunta, texto, contra-pergunta (itálico), radiobuttons Sim/Não,
CTkEntry inicialmente desabilitado (habilita com Sim).
- Resultado: bloco destacado com resumo (kg, tCO₂, créditos, R\$, mudas).
- Compensação: OptionMenu de projetos; descrição; custo calculado com base no último cálculo.
- Ações: “Calcular Emissão de CO₂”, “Voltar” (menu), “Ver Histórico” (no mesmo frame).

Tela de Histórico:

- Listagem em ordem reversa (mais recentes no topo).
- Card com timestamp, tipo, usuário, totais, equivalências e (se houver) compensação.
- Ações: “Limpar Histórico” (confirma e remove arquivo), “Voltar” (menu ou perguntas).

Reinicialização:

- Botão persistente no header; confirmação; reset de variáveis e reconstrução de frames.

4.4 FUNÇÕES (PRE/PÓS/EFEITOS)

mostrar_login():

- Pré: janela e frames criados.
- Pós: login exibido; ao validar, usuario definido; navega ao menu.
- Efeitos: define validar_e_continuar; foco no entry; bind de Enter.

mostrar_menu():

- Pré: usuario definido.
- Pós: frame_menu exibido; frame_perguntas ocultado; botões ativos.
- Efeitos: limpeza/recriação do frame para evitar resíduos de UI.

mostrar_perguntas(tipo):

- Pré: tipo ∈ {pessoas, empresas}; menu ocultado.
- Pós: frame_peruntas exibido; carregar_peruntas() invocado.
- Efeitos: nenhum adicional.

carregar_peruntas(tipo):

- Pré: perguntas.json carregado; tipo válido.
- Pós: listas entradas[] e escolhas[] alinhadas a perguntas do tipo; UI pronta.
- Efeitos: define update_project_info; instancia labels globais de resultado/compensação.

calcular_co2(tipo):

- Pré: listas entradas[]/escolhas[] alinhadas ao formulário exibido.
- Pós: ultimo_calculo atualizado; histórico persistido (compensado=false).
- Efeitos: mensagens de erro por pergunta (vermelho) ou sucesso (verde); atualização do custo.

compensar_emissao():

- Pré: ultimo_calculo válido; projeto selecionado.
- Pós: histórico persistido com compensado=true; UI atualizada com confirmação.
- Efeitos: diálogo de confirmação.

mostrar_historico(tipo?):

- Pré: leitura segura de historico.json (ou lista vazia).
- Pós: lista renderizada; limpeza possível; retorno ao ponto de origem.
- Efeitos: pode apagar o arquivo de histórico mediante confirmação.

reiniciar_app():

- Pré: janela ativa; header presente.
- Pós: estado global resetado; frames recriados; login exibido.
- Efeitos: destrói widgets exceto header_frame.

4.5 MODELO DE DADOS E DICIONÁRIO

perguntas.json:

- pergunta_pessoas/pergunta_empresas: lista de objetos com:
 - numero_pergunta (int): ordinal da questão no formulário.
 - pergunta (str): texto principal apresentado ao usuário.
 - contra_pergunta (str): texto auxiliar (unidade/escopo).
 - calculo_co2 (float): fator de emissão (kg CO₂ por unidade respondida).

historico.json (lista de registros, ordem cronológica na escrita):

Campos:

- timestamp (str, “YYYY-MM-DD HH:MM:SS”): data e hora do registro.
- tipo (str, “pessoas|empresas”): categoria do formulário.
- usuario (str): quem executou (conteúdo do login ou “—”).
- total_co2 (float, kg): soma ponderada das emissões.
- creditos (float, tCO₂): total_co2 / 1000.
- valor_reais (float, R\$): creditos * preço_ref (atual 78,46).
- mudas (float): equivalência 1 crédito = 1 muda (regra simplificada).
- compensado (bool): false para cálculo; true quando registro de compensação.
- projeto (str|null): nome do projeto escolhido, se compensado.
- preco_projeto (float|null): R\$/t do projeto.
- custo_compensacao (float|null): creditos * preco_projeto.

Estado em Memória:

- usuario: str|None; ultimo_calculo: dict|None;
- selected_project_var: StringVar; project_desc_label/compensation_cost_label: labels CTk.
- entradas: List[CTkEntry]; escolhas: List[StringVar].

4.6 REGRAS DE NEGÓCIO, FÓRMULAS E LIMITAÇÕES

Fórmulas principais:

- Emissão por pergunta: $e_i = \text{valor}_i * \text{fator}_i$ (kg).
- Emissão total: $E = \sum e_i$ (kg).
- Conversão para toneladas: $T = E / 1000$ (tCO₂).
- Créditos de carbono: $C = T$ (1 crédito ≡ 1 tCO₂).

- Valor monetário estimado: $R\$ = C * \text{preço_ref}$.
- Mudas: $M = C$ (regra 1:1 como métrica simplificada).

Regras de UI/Lógica:

- Entradas só são válidas quando a resposta for “Sim”.
- Entradas “Não”: entry desabilitado e limpo automaticamente.
- Erro de conversão (float) detém o cálculo e informa “Valor inválido” por pergunta.

Limitações atuais:

- preço_ref fixo (78,46) para estimativa monetária (aprimorável).
- Sem validação de limites de valor (ex.: negativos ou anômalos) – pode ser adicionado.
- Ausência de unidades explícitas no entry (implícitas na contra_pergunta).

4.7 PERSISTÊNCIA, VERSÕES E MIGRAÇÃO DE DADOS

Persistência:

- Esquema simples em JSON; leitura/escrita com encoding UTF-8 e indentação.
- Em caso de erro de leitura (I/O/corrupção), carregar_historico retorna lista vazia.

Migração de schema:

- Estratégia inicial: compatibilidade por adição de campos opcionais no final.
- Evoluções: versionar historico.json com um campo _schemaVersion quando necessário.
- Regras: novos campos devem ter defaults seguros ao ler registros antigos.

Integridade:

- Operações de append (ler → append → gravar) preservam histórico.
- Erros em escrita exibem messagebox (ponto de melhoria: fallback em arquivo .bak).

4.8 INTERFACE: DESIGN SYSTEM, TEMA E ACESSIBILIDADE

Paleta e fontes:

- Tema light; cores primárias verdes (corBtn, corHover) com fundo neutro (corFundo).
- Hierarquia tipográfica: títulos (24, bold), conteúdo (14-16), labels auxiliares (12-13).

Componentes:

- CTkFrame, CTkLabel, CTkButton, CTkEntry, CTkRadioButton,
- CTkOptionMenu,
- CTkScrollableFrame.

Diretrizes:

- Contraste suficiente (texto escuro em fundo claro, erro em vermelho).
- Espaçamento consistente; botões grandes; rolagem suave.
- Mensagens claras e curtas; evitar jargões.

Melhorias futuras:

- Aparência “dark” opcional com ctk.set_appearance_mode("dark").
- Escalonamento: ctk.set_widget_scaling/ctk.set_window_scaling para telas 4K.

4.9 TRATAMENTO DE ERROS E MENSAGENS AO USUÁRIO

Tipos de mensagens:

- Validação: erro vermelho contextual (login e perguntas).
- Confirmações: reiniciar, apagar histórico, compensar.
- Erros I/O: messagebox.showerror com conteúdo sucinto.
- Sucesso: messagebox.showinfo pós-compensação e limpeza.

Taxonomia de erros:

- Entrada inválida (ValueError).
- I/O (FileNotFoundException, PermissionError, JSONDecodeError – tratadas genericamente).
- Estado inválido (tentar compensar sem cálculo).

Princípios:

- Fail-fast na UI, sem travar aplicação.
- Manter o usuário no contexto de correção.
- Mensagens sem vazamento de stacktraces.

4.10 REQUISITOS NÃO FUNCIONAIS (NFRs)

Desempenho:

- Cálculo O(n) no número de perguntas exibidas.
- I/O local de baixa latência; arquivos modestos (milhares de linhas).

Confiabilidade:

- Try/except em pontos críticos de leitura/escrita e conversões.
- Confirmações para ações irreversíveis.

Portabilidade:

- Python 3.10+ e “pip install customtkinter”.
- Windows/macOS/Linux; sem serviços remotos.

Segurança e Privacidade:

- Sem PII sensível; histórico local; nenhuma transmissão de dados.
- Operação offline por padrão.

Manutenibilidade:

- Funções coesas; nomes descritivos; estruturas previsíveis.
- Roadmap de modularização para aumentar testabilidade.

4.11 TESTABILIDADE, PLANO E CASOS DE TESTE

Estratégia atual (manual):

- Login: nomes curtos, com dígitos e válidos; Enter; feedbacks.
- Menu: bifurcação; retorno.
- Perguntas: Sim/Não alternando entry; entradas válidas e inválidas; cálculo.

- Resultado: consistência das conversões e formatação.
- Compensação: projetos, custo, confirmação, registro no histórico.
- Histórico: crescimento, ordem reversa, limpeza e retorno.
- Reinício: confirmação, limpeza e volta ao login.

Casos notáveis:

- Vazio/whitespace no login.
- Valor textual em pergunta com “Sim”.
- Compensação sem cálculo prévio.
- Apagar histórico sem permissões (erro de I/O).

Futuro (automatizável):

- Services puros de cálculo/persistência após modularização.
- Testes de integração (calcular→persistir→listar).
- Smoke de UI com ferramentas de automação.

4.12 SEGURANÇA E PRIVACIDADE

- Dados armazenados localmente no diretório do app (historico.json).
- Sem autenticação ou rede; risco reduzido.
- Usuário pode limpar histórico a qualquer momento (com confirmação).
- Melhorias: alertar sobre backup antes de limpeza; padrão “lixeira”/restore opcional.

4.13 OBSERVABILIDADE E LOGS (PLANEJADO)

- Logging mínimo recomendado em arquivo .log com INFO/ERROR e timestamps.
- Registrar eventos: cálculo, falha de I/O, limpeza de histórico, compensação.
- Rotacionar logs e manter volume sob controle (política simples de retenção).

4.14 OPERAÇÃO E SUPORTE

Execução:

- Requisitos: Python 3.10+; instalar “pip install customtkinter”.

- Rodar: “python app.py” na raiz do projeto.

Diretórios e arquivos:

- perguntas.json (obrigatório).
- historico.json (criado sob demanda).

Suporte de campo:

- Verificar permissões de escrita no diretório do app.
- Conferir integridade dos JSONs (encoding UTF-8 e estrutura).

4.15 DECISÕES ARQUITETURAIS (ADRs RESUMIDAS)

ADR-001: Procedural vs OOP – Escolhido procedural por simplicidade e escopo curto.

ADR-002: CustomTkinter – Aparência moderna com mínima curva de aprendizado.

ADR-003: JSON local – Persistência simples, legível e portátil.

ADR-004: Frames + pack/pack_forget – Navegação simples em uma janela única.

ADR-005: Scroll nos containers – Escala com número de perguntas e histórico crescente.

4.16 RISCOS, IMPACTOS E MITIGAÇÕES

Risco 1: Crescimento do monolito → Mitigar com modularização planejada (Seção 18).

Risco 2: Estado global e side-effects → Documentar variáveis; reduzir escopos na refatoração.

Risco 3: Corrupção/erro de I/O → Try/except + mensagem; pensar em backup simples .bak.

Risco 4: Entradas inválidas silenciosas → Validação explícita e mensagens por pergunta.

Risco 5: Acessibilidade insuficiente → Reforçar contraste, tamanhos e navegação por teclado.

4.17 DEPENDÊNCIAS E TERCEIROS

- Python 3.10+.
- Biblioteca CustomTkinter (pip).
- Módulos padrão: json, os, datetime, tkinter.messagebox.

4.18 ACESSIBILIDADE

- Contraste preservado no tema light; reforçar com dark mode no futuro.
- Foco inicial no campo relevante (login); Enter para submeter.
- Tamanho mínimo de fonte e espaçamentos generosos.
- Evolução: rótulos acessíveis, mensagens textuais descritivas em todos os botões.

4.19 INTERNACIONALIZAÇÃO E LOCALIZAÇÃO (I18N/L10N)

- Textos em pt-BR; base pronta para externalização de strings.
- Futuro: arquivo i18n.json por idioma; chave→texto; OptionMenu de idioma.
- Datas e moedas: considerar locale-aware format (Brasil por padrão).

4.20 EMPACOTAMENTO E DISTRIBUIÇÃO

- Desenvolvimento: execução via Python local.
- Distribuição desktop (opcional): pyinstaller --noconsole --onefile --name CalculadoraCarbono app.py
- Incluir perguntas.json como recurso; garantir escrita para historico.json no diretório.

4.21 CONTRIBUIÇÃO E PADRÕES DE CÓDIGO

- Nomenclatura clara e coesa; comentários sucintos.
- Funções curtas com responsabilidade única.
- Commits descritivos; PRs pequenos e focados.
- Revisão: validar comportamentos de UI, persistência e mensagens.

4.22 MANUTENÇÃO E GERÊNCIA DE MUDANÇA

- Controle de versão semântico: MAJOR.MINOR.PATCH.
- Documentar mudanças no CHANGELOG (futuro).
- Para mudanças em perguntas.json: validar consistência das chaves e fatores.
- Para alterações em histórico: manter compatibilidade ou migrar schema com versionamento.

4.23 PSEUDOCÓDIGO DE CÁLCULO E COMPENSAÇÃO

Cálculo:

```
total_kg := 0
para i em [0..n-1]:
    se escolhas[i] == "Sim":
        valor := float(entradas[i])
        total_kg += valor * fator[i]
tco2 := total_kg / 1000
creditos := tco2
reais := creditos * preco_ref
mudas := creditos
persistir_registro(compensado=false, ...)
```

Compensação:

```
exigir ultimo_calculo
projeto := selected_project_var.get()
price := project_types[projeto].price
cost := ultimo_calculo.creditos * price
confirmar_usuario(cost)
persistir_registro(compensado=true, projeto, price, cost)
informar_sucesso()
```

4.24 EXEMPLOS DE REGISTROS JSON

Exemplo de cálculo (sem compensação):

```
{  
  "timestamp": "2025-01-06 14:52:31",  
  "tipo": "pessoas",  
  "usuario": "Maria",  
  "total_co2": 245.75,  
  "creditos": 0.24575,  
  "valor_reais": 19.29,  
  "mudas": 0.24575,  
  "compensado": false,  
  "projeto": null,  
  "preco_projeto": null,  
  "custo_compensacao": null  
}
```

Exemplo de compensação:

```
{  
  "timestamp": "2025-01-06 15:03:10",  
  "tipo": "pessoas",  
  "usuario": "Maria",  
  "total_co2": 245.75,  
  "creditos": 0.24575,  
  "valor_reais": 19.29,  
  "mudas": 0.24575,  
  "compensado": true,  
  "projeto": "Reflorestamento",  
  "preco_projeto": 90.0,  
  "custo_compensacao": 22.12  
}
```

4.25 CENÁRIOS NARRATIVOS DETALHADOS

Cenário Padrão (Pessoa):

- 1) Usuário “Ana” abre o app, insere o nome, valida e acessa o menu.
- 2) Escolhe “Pessoas”, marca “Sim” para eletricidade e “Não” para demais perguntas.

- 3) Informa 180 kWh/mês, calcula; vê total em kg, créditos, R\$ e mudas.
- 4) Seleciona “Energias Renováveis”, observa custo, e decide compensar; confirma diálogo.
- 5) Abre histórico, visualiza dois cards (cálculo e compensação), volta ao menu.

Cenário Empresa:

- 1) Gestor acessa, escolhe “Empresas”, preenche insumos variados.
- 2) Calcula, analisa resultado; opta por não compensar no momento.
- 3) Registro fica salvo; histórico cresce de forma ordenada.

Cenário Erro de Entrada:

- 1) Em “Pessoas”, marca “Sim” e digita “abc”.
- 2) Ao calcular, recebe “Valor inválido na pergunta X”; corrige e conclui.

5. RELATÓRIO – CÓDIGO NO GITHUB

<https://github.com/mtslipe/creditos-de-carbono>

6. BIBLIOGRAFIA

O que é e como funciona o mercado de carbono? Disponível em:
<<https://ipam.org.br/cartilhas-ipam/o-que-e-e-como-funciona-o-mercado-de-carbono>>. Acesso em: 11 nov. 2025.

NETLINKS. Crédito de carbono: o que é, como funciona e como vender. Disponível em: <<https://bydenergia.com/credito-de-carbono>>. Acesso em: 11 nov. 2025.

Créditos de carbono: o que são e para que servem? Disponível em:
<<https://mundoeducacao.uol.com.br/geografia/creditos-de-carbono.htm>>

Crédito de carbono: qual a importância para o futuro? Disponível em:
<<https://energiaenegocios.rn.sebrae.com.br/credito-de-carbono>>. Acesso em: 11 nov. 2025.

Saiba como fazer o cálculo da precificação de carbono | CredCarbo. Disponível em:
<<https://credcarbo.com/carbono/saiba-como-fazer-o-calculo-da-precificacao-de-carbono>>. Acesso em: 11 nov. 2025.

W3SCHOOLS. Python Tutorial. Disponível em:
<<https://www.w3schools.com/python/>>.

PYTHON SOFTWARE FOUNDATION. 3.7.3 Documentation. Disponível em:
<<https://docs.python.org/3/>>.

SCHIMANSKY, T. Official Documentation and Tutorial | CustomTkinter. Disponível em: <<https://customtkinter.tomschimansky.com/>>.



FICHA DAS ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

NOME: FELIPE MATIAS MAXIMIANO DA SILVA TURMA: CC2P13 RA: F360GE5

CURSO: CIÊNCIA DA COMPUTAÇÃO _____ **CAMPUS: MARQUÊS** _____ **SEMESTRE: 2º** _____ **TURNO: NOTURNO** _____

CÓDIGO DA ATIVIDADE: APS - IPE SEMESTRE: 2º SEMESTRE ANO GRADE: 2025/2

(1) Horas atribuídas de acordo com o regulamento das Atividades Práticas Supervisionadas do curso.

TOTAL DE HORAS ATRIBUÍDAS:

AVALIAÇÃO: _____

Aprovado ou Reprovado

NOTA: _____

DATA: _____ / _____ / _____

CARIMBO E ASSINATURA DO COORDENADOR DO CURSO



FICHA DAS ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

NOME: KAUAN ERON CARNEIRO ARAUJO **TURMA:** CC2P13 **RA:** R8425J6

CURSO: CIÊNCIA DA COMPUTAÇÃO **CAMPUS: MARQUES** **SEMESTRE: 2** **TURNO: NOTURNO**

CÓDIGO DA ATIVIDADE: APS – IPE _____ **SEMESTRE: 2º SEMESTRE** _____ **ANO GRADE: 2025/2**

(1) Horas atribuídas de acordo com o regulamento das Atividades Práticas Supervisionadas do curso.

TOTAL DE HORAS ATRIBUÍDAS:

AVALIAÇÃO:

Aprovado ou Reprovado

NOTA: _____

DATA: _____ / _____ / _____

CARIMBO E ASSINATURA DO COORDENADOR DO CURSO



FICHA DAS ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

NOME: GIOVANNA ALCANTARA DO NASCIMENTO **TURMA: CC2P13** **RA: H7538A9**

CURSO: CIÊNCIA DA COMPUTAÇÃO **CAMPUS: MARQUES** **SEMESTRE: 2** **TURNO: NOTURNO**

CÓDIGO DA ATIVIDADE: APS - IPE **SEMESTRE: 2° SEMESTRE** **ANO GRADE: 2025/2**

(1) Horas atribuídas de acordo com o regulamento das Atividades Práticas Supervisionadas do curso.

TOTAL DE HORAS ATRIBUÍDAS:

AVALIAÇÃO:

Aprovado ou Reprovado

NOTA: _____

DATA: _____ / _____ / _____

CARIMBO E ASSINATURA DO COORDENADOR DO CURSO



FICHA DAS ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

NOME: HENRIQUE NASCIMENTO DE SOUZA _____ **TURMA: CC2P13** _____ **RA: R829FJ7** _____

CURSO: CIENCIA DA COMPUTACAO CAMPUS: MARQUES SEMESTRE: 2 TURNO: NOTURNO

CÓDIGO DA ATIVIDADE: APS – IPE SEMESTRE: 2º SEMESTRE ANO GRADE: 2025/2

(1) Horas atribuídas de acordo com o regulamento das Atividades Práticas Supervisionadas do curso.

TOTAL DE HORAS ATRIBUÍDAS:

AVALIAÇÃO: _____

Aprovado ou Reprovado

NOTA: _____

DATA: _____ / _____ / _____

CARIMBO E ASSINATURA DO COORDENADOR DO CURSO



FICHA DAS ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

NOME: LEONARDO DE ARAÚJO MOURA **TURMA:** CC1P13 **RA:** R939642

CURSO: CIENCIA DA COMPUTAÇÃO _____ **CAMPUS: MARQUES** _____ **SEMESTRE: 1** _____ **TURNO: NOTURNO** _____

CÓDIGO DA ATIVIDADE: APS – IPE _____ SEMESTRE: 1º SEMESTRE _____ ANO GRADE: 2025/2_____

(1) Horas atribuídas de acordo com o regulamento das Atividades Práticas Supervisionadas do curso.

TOTAL DE HORAS ATRIBUÍDAS:

AVALIAÇÃO: _____

Aprovado ou Reprovado

NOTA: _____

DATA: _____ / _____ / _____

CARIMBO E ASSINATURA DO COORDENADOR DO CURSO



FICHA DAS ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

NOME: MATHEUS DE SOUZA LIMA COSTA TURMA: CC2P13 RA: H7663B5

TURMA: CC2P13 RA: H7663B5

CURSO: CIENCIA DA COMPUTACAO _____ **CAMPUS: MARQUES** _____ **SEMESTRE: 2** _____ **TURNO: NOTURNO** _____

CAMPUS: MARQUES _____ **SEMESTRE: 2** _____ **TURNO: NOTURNO** _____

SEMESTRE: 2 TURNO: NOTURNO

CÓDIGO DA ATIVIDADE: APS - IPE _____ **SEMESTRE: 2º SEMESTRE**

SEMESTRE: 2º SEMESTRE

(1) Horas atribuídas de acordo com o regulamento das Atividades Práticas Supervisionadas do curso.

TOTAL DE HORAS ATRIBUÍDAS:

AVALIAÇÃO:

Aprovado ou Reprovado

NOTA:

DATA: _____ / _____ / _____

For more information about the study, please contact the study team at 1-800-258-4263 or visit www.cancer.gov.

CARIMBO E ASSINATURA DO COORDENADOR DO CURSO