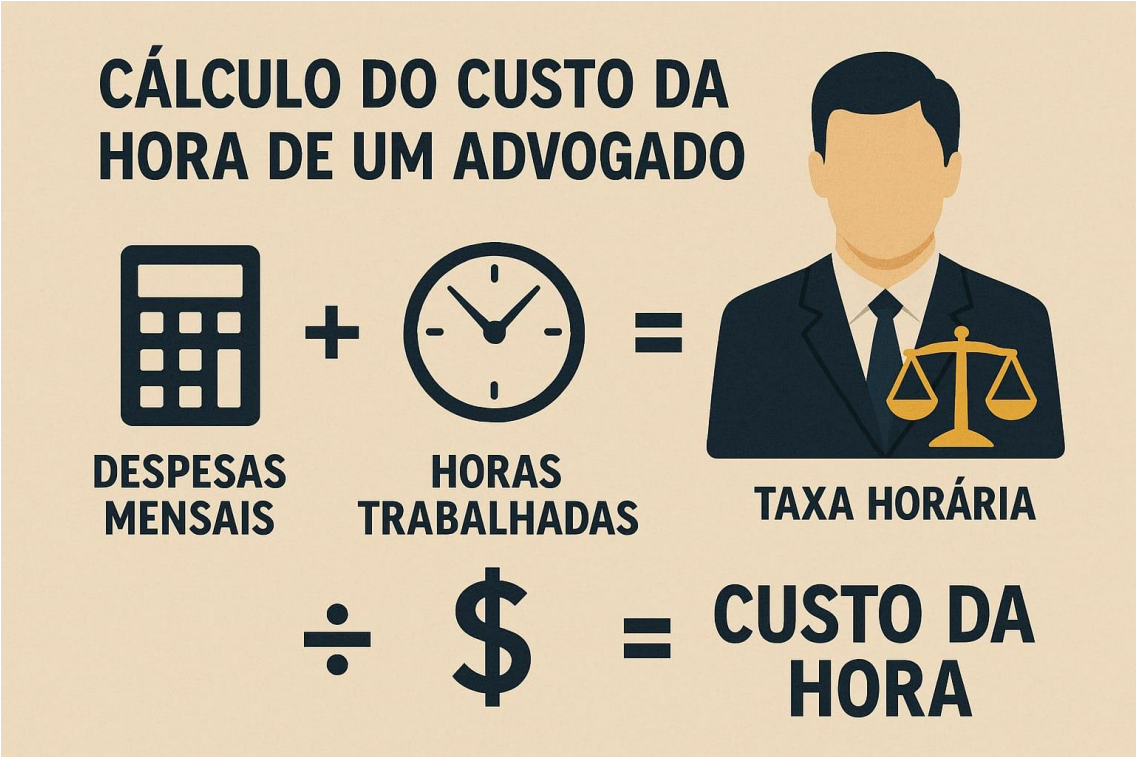
Do Zero à Interface Gráfica: Criando um Simulador de Honorários com Python e CustomTkinter

Izairton Oliveira de Vasconcelos

Repositório GitHub:
https://github.com/IOVASCON/simulador_honorarios.git



Você já pensou em criar uma aplicação desktop que resolve um problema real, mas talvez tenha se sentido intimidado pela complexidade ou por não saber por onde começar? Calcular honorários advocatícios, por exemplo, pode envolver diversas variáveis: tempo de experiência, especializações, sucesso em casos anteriores, horas dedicadas, e até mesmo o valor da causa. Como transformar essa lógica complexa em uma ferramenta prática e visualmente agradável?

Neste artigo, vamos embarcar juntos na jornada de construção de um Simulador de Honorários Advocatícios usando Python. Desmistificaremos o processo, desde a definição da lógica de cálculo até a criação de uma interface gráfica interativa com a biblioteca CustomTkinter e a geração de relatórios em PDF. Se você é iniciante ou já tem alguma experiência com Python e quer ver

como aplicar seus conhecimentos em um projeto concreto, este guia é para você! Vamos lá?

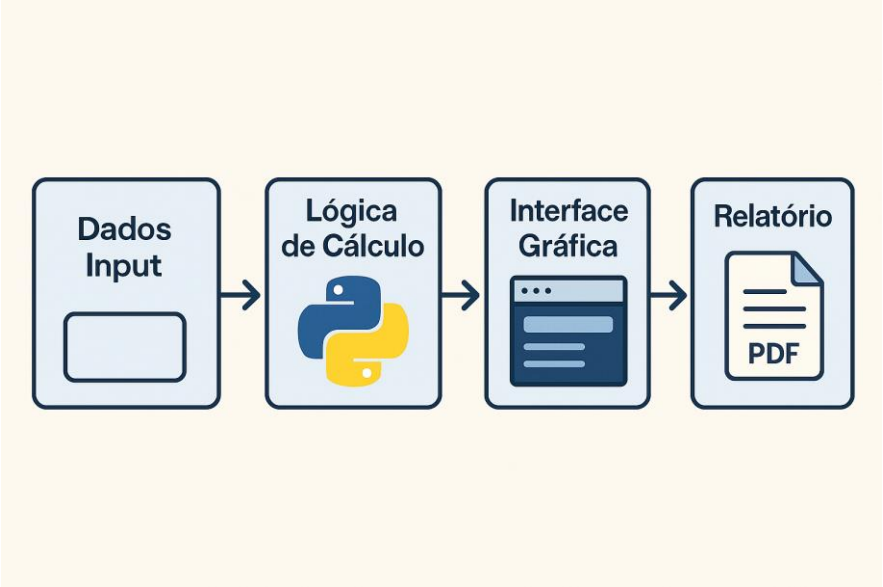


IMAGEM 1: Ilustração de um fluxograma simples: Dados Input -> Lógica de Cálculo (Python) -> Interface Gráfica (CustomTkinter) -> Relatório (PDF).

1. Desvendando o Desafio: Entendendo a Lógica por Trás dos Honorários

IMAGEM 2: Formação do Preço da Hora do Advogado

Antes de escrever qualquer linha de código de interface, o passo fundamental é entender e modelar a lógica de negócio. Afinal, o que a nossa calculadora precisa fazer? No nosso caso, o objetivo é sugerir um valor de taxa horária e um preço final para um

serviço, considerando diversos fatores que valorizam o profissional.

Decidimos que a taxa horária não seria fixa, mas sim calculada a partir de:

- Uma Taxa Base Mínima: Um valor inicial definido pelo advogado.
- Fatores Multiplicadores: Que aumentam a taxa base com base em:
  - Tempo de experiência (desde a graduação/OAB).
  - Número e "idade" das pós-graduações (especialização).
  - Volume e taxa de sucesso em casos anteriores (experiência prática).
  - Investimento em educação (custos com cursos, livros, etc.).
  - Dedicação extra (horas trabalhadas fora do expediente).

Quanto vale a hora de um advogado?

(E por que essa resposta é mais profunda do que parece)
Muitos já se fizeram essa pergunta:

"Por que a hora do advogado custa tanto?"

Mas poucos percebem que, por trás de cada consulta jurídica, existe **uma história de décadas de preparação, riscos assumidos, decisões difíceis e um capital invisível construído ao longo de anos.**

Neste artigo, vamos explorar de forma clara — mas com profundidade — os fatores que realmente compõem o preço da hora de um advogado. E mais: você verá como a tecnologia pode ajudar a precificar isso de forma mais justa e transparente, com um projeto em Python simples de aplicar.

A hora vendida não é a hora trabalhada

Vamos começar com um conceito importante do universo da precificação:

A hora cobrada ≠ hora cronológica trabalhada.

Isso significa que, quando um advogado cobra R$ 450 por hora, ele **não está vendendo 60 minutos do relógio**, mas sim:

- **Anos de estudo** (faculdade, pós, cursos, certificações);
- **Análise jurídica especializada** (cada caso exige decisões sérias e complexas);

- **Gestão de risco profissional** (uma má orientação pode ter grandes consequências);
- **Reputação e confiança** que ele construiu ao longo do tempo.

Intangíveis: o valor que não cabe na planilha

Aqui entramos nos **valores intangíveis**, aqueles que não aparecem no boleto do aluguel nem na fatura do cartão, mas pesam muito no preço final:

- **Credibilidade e autoridade na área**
- **Velocidade de raciocínio jurídico** (que vem da experiência, e não de pressa)
- **Atualização constante em leis, jurisprudência e ferramentas digitais**
- **Relacionamento com o meio jurídico** (que pode abrir portas, literalmente)

IMAGEM 3: Valoração do Intangível

Além disso, o preço final do serviço específico levaria em conta:

- A taxa horária calculada.
- As horas estimadas para o serviço.
- Fatores de complexidade e urgência do caso.

Separar essa lógica do restante do código (por exemplo, em um arquivo formulas.py) é crucial. Isso torna o sistema mais organizado, fácil de testar e de dar manutenção no futuro. Pense nisso como construir a fundação sólida da sua casa antes de se preocupar com a cor das paredes.

2. Mãos à Obra com Python: Implementando os Cálculos

Com a lógica definida, é hora de traduzi-la para Python. Criamos funções específicas dentro do nosso módulo core/formulas.py para cada parte do cálculo:

- calcular_taxa_horaria_sugerida(dados_input): Recebe um dicionário com todos os dados informados pelo usuário e aplica os fatores multiplicadores à taxa base. Ela retorna a taxa final e um dicionário com os detalhes de como cada fator contribuiu (ótimo para transparência no relatório!).
- calcular_preco_final_servico(taxa_calculada, dados_input): Usa a taxa horária calculada anteriormente, multiplica pelas horas estimadas e aplica os fatores de complexidade e urgência do serviço específico.

Também precisamos de funções auxiliares (talvez em core/utils.py) para tarefas como:

- Validar e converter datas (de string "DD/MM/AAAA" para objeto date).
- Formatar valores monetários (para exibir "R$ 1.234,56").

```
# --- Funções Principais de Cãlculo ---

def calcular_taxa_horaria_sugerida(dados:
   dict) → taple[float, dict]  ɔi'ffotaı_)

   ''Calcula a taxa horária final
   combinande todos os fatores.'

 fator_tempo_ dados['taxa_horaria_base_minima
 fatores_aplicados =()
 detalhes_fatores =detalhes_fatores

 fator_tempo, det_tempo=calcular_fator_tempo
 detalhes_fatores['Tempo Experiência'):
 fator_pos =detalhes_fatores'Espelalzação Pós'

 fatore_aplicados['Espcialização (ós)':
 detalhes_fatores['Especialização (Pós)'
```

IMAGEM 4: código da função calcular_taxa_horaria_sugerida.

Escrever testes para essas funções de cálculo, mesmo que simples, é uma excelente prática para garantir que a lógica está correta antes mesmo de pensar na interface.

3. Dando Vida à Aplicação: Construindo a Interface com CustomTkinter

Agora a parte visual! Escolhemos o CustomTkinter por ser uma biblioteca moderna que facilita a criação de interfaces com aparência agradável e temas customizáveis, baseada no conhecido Tkinter.

O coração da nossa interface é a classe App (em gui/app.py), que herda de ctk.CTk. Dentro dela, organizamos os elementos visuais usando o sistema de layout grid. Alguns componentes chave que utilizamos:

- **CTkScrollableFrame**: Essencial para acomodar todos os campos de entrada sem poluir a tela inicial. Se o conteúdo for maior que o espaço visível, a barra de rolagem aparece automaticamente. Mágico, não?
- **CTkLabel**: Para exibir textos descritivos (os rótulos dos campos).
- **CTkEntry**: Campos para o usuário digitar informações (nome, datas, números, valores).
- **CTkComboBox**: Caixas de seleção para opções pré-definidas (área de atuação, complexidade, urgência).
- **CTkButton**: O botão principal para acionar o cálculo.
- **CTkFrame**: Frames invisíveis (com fg_color="transparent") para agrupar widgets relacionados e organizar layouts mais complexos (como as "Ações Ganhas por Área" ou os campos de Pós-Graduação adicionados dinamicamente).

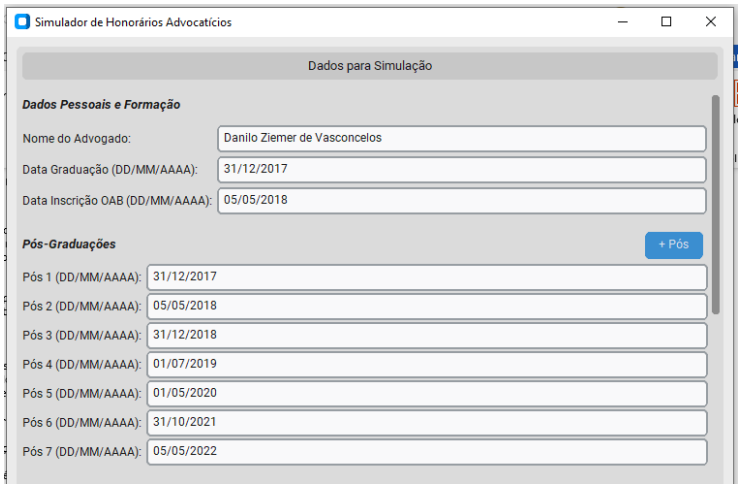


IMAGEM 5: Interface Gráfica do simulador

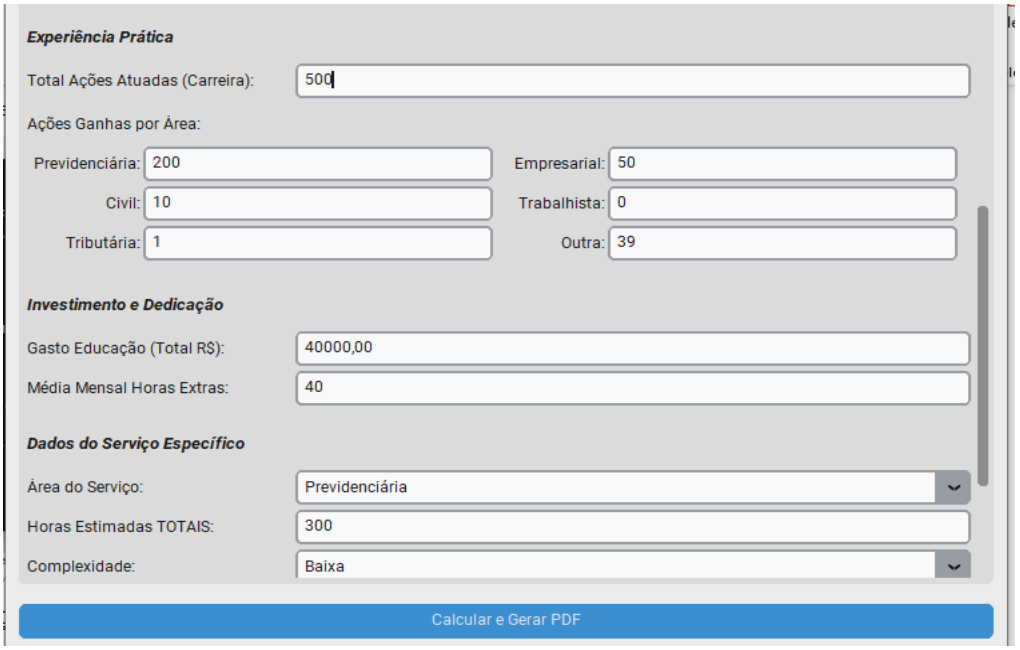


IMAGEM 6: Interface do simulador com alguns widgets

Organizar os widgets com grid exige um pouco de planejamento (definir row, column, padx, pady, sticky, columnspan), mas oferece grande flexibilidade. Usar columnconfigure e rowconfigure com weight=1 é o segredo para fazer os elementos se expandirem e se adaptarem ao redimensionamento da janela.

4. Conectando os Pontos: Integrando Lógica e Interface

Com a interface montada e a lógica de cálculo pronta, precisamos fazê-los conversar. É aqui que o método calculate() da nossa classe App entra em ação, disparado pelo clique no botão "Calcular e Gerar PDF". Veja o fluxo:

1. **Coleta de Dados**: O método lê o valor de cada CTkEntry e CTkComboBox relevante usando o método .get(). Guardamos essas entradas em um dicionário dados_input.
2. **Validação**: Antes de calcular, validamos tudo!
    - Formatos: Datas estão corretas? Números são realmente números? Usamos funções auxiliares e blocos try-except para isso.
    - Intervalos: Percentuais estão entre 0 e 100? Valores monetários não são negativos?
    - Lógica Cruzada: O total de ações ganhas não é maior que o total atuado?
    - Alertas: Valores como Horas Estimadas ou Taxa Base parecem altos demais? Usamos messagebox.askyesno para pedir confirmação ao usuário antes de prosseguir. Se algo estiver errado, exibimos um messagebox.showerror claro e interrompemos.

3. **Chamada da Lógica**: Se tudo estiver válido, passamos o dicionário dados_input para as funções calcular_taxa_horaria_sugerida() e calcular_preco_final_servico().
4. **Exibição**: Recebemos os resultados dessas funções e formatamos um texto de resumo. Atualizamos o CTkLabel na parte inferior da interface (self.results_label.configure(text=...)) para mostrar os valores calculados ao usuário.

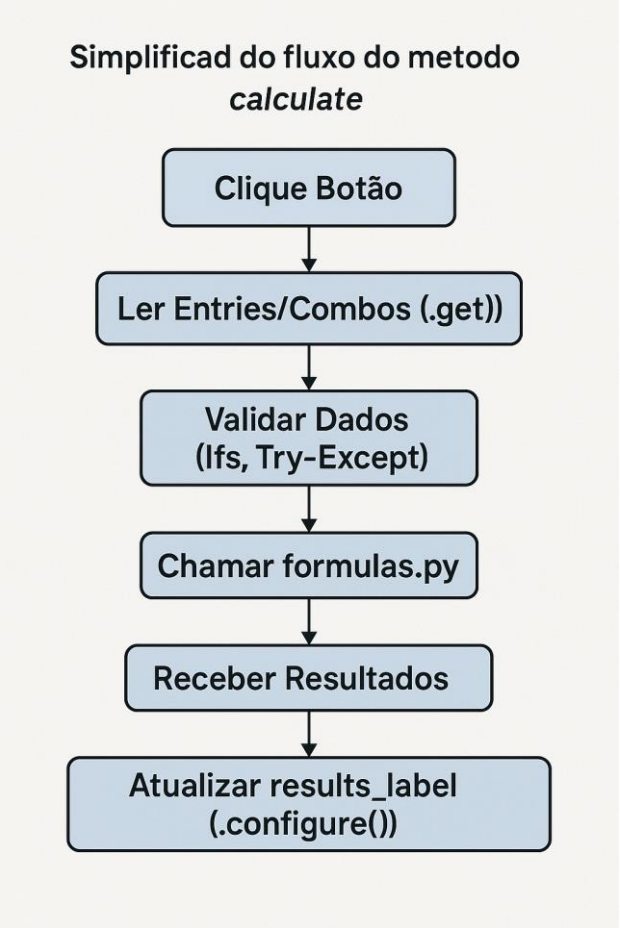


IMAGEM 7: Diagrama simplificado do fluxo do método calculate

Esse método calculate é o maestro que rege toda a operação, garantindo que os dados fluam corretamente da interface para a lógica e de volta para a interface.

5. Além da Tela: Gerando Relatórios em PDF

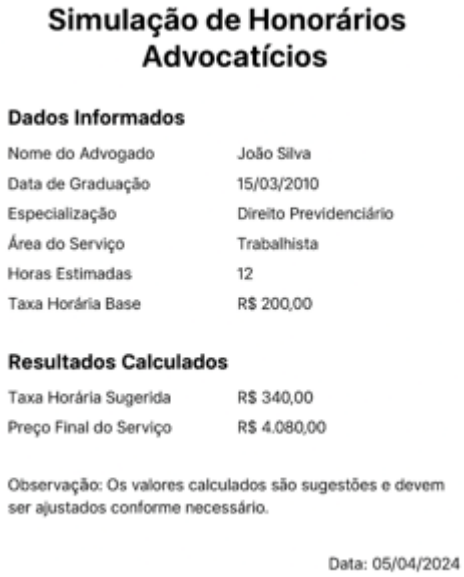
Uma calculadora na tela é útil, mas poder gerar um relatório formal em PDF para apresentar ao cliente ou arquivar é um grande diferencial. Para isso, utilizamos a biblioteca ReportLab.

Criamos um módulo dedicado reports/pdf_generator.py com uma função principal, por exemplo, gerar_pdf_simulacao(). Essa função recebe:

- O caminho completo onde o PDF será salvo.
- O dicionário dados_input com tudo que o usuário informou.
- O dicionário resultados_calculo com todos os valores e detalhes calculados.

Dentro dessa função, usamos os recursos do ReportLab para:

- Definir o layout da página (margens, etc.).
- Criar estilos de parágrafo (títulos, texto normal).
- Adicionar textos formatados, tabelas ou listas, buscando as informações nos dicionários recebidos.
- Incluir informações importantes como a data de geração, nome do advogado, e até mesmo as observações sobre como os valores são sugestões.
- Salvar o arquivo PDF final.



Simulação de Honorários Advocatícios

Dados Informados

Nome do Advogado	João Silva
Data de Graduação	15/03/2010
Especialização	Direito Previdenciário
Área do Serviço	Trabalhista
Horas Estimadas	12
Taxa Horária Base	R$ 200,00

Resultados Calculados

Taxa Horária Sugerida	R$ 340,00
Preço Final do Serviço	R$ 4.080,00

Observação: Os valores calculados são sugestões e devem ser ajustados conforme necessário.

Data: 05/04/2024

IMAGEM 8: Relatório PDF gerado

Integrar a geração do PDF foi o último passo para tornar o simulador uma ferramenta verdadeiramente completa.

6. Enfrentando o Inesperado: Depuração e a Solução do DPI

Nenhuma jornada de desenvolvimento é livre de obstáculos! Durante a criação da interface com CustomTkinter, nos deparamos com um erro misterioso (_tkinter.TclError: bad screen distance "XXX.0") que só ocorria em alguns ambientes Windows e impedia a renderização correta do CTkScrollableFrame.

IMAGEM 9: Programador confuso.

Após muita investigação e testes (tentando diferentes versões, isolando widgets, verificando configurações de localidade), descobrimos que o problema estava relacionado à forma como o CustomTkinter (ou o Tkinter subjacente) tentava lidar com o dimensionamento de DPI (pontos por polegada) da tela. A solução? Uma única linha de código no início do nosso script principal (main_gui.py):

customtkinter.deactivate_automatic_dpi_awareness()

Essa linha instrui a biblioteca a não tentar ajustar automaticamente o dimensionamento, evitando os valores mal formatados que causavam o erro. Foi um lembrete poderoso de que a depuração faz parte do processo e que a persistência em testar e pesquisar é fundamental para superar esses desafios inesperados.

7. Refinando e Expandindo: Dicas Finais e Próximos Passos

Nosso simulador está funcional, mas a jornada de um desenvolvedor nunca termina! Sempre há espaço para melhorias e novas funcionalidades. Algumas ideias para levar este projeto adiante:

- **Tooltips**: Adicionar dicas flutuantes (CTkToolTip ou similar) para explicar o que cada campo significa.
- **Melhor Feedback de Erro**: Destacar visualmente os campos com erros de validação.
- **Configuração Externa**: Mover parâmetros como limites de alerta ou pesos dos fatores para um arquivo de configuração (JSON, YAML), facilitando ajustes sem mexer no código Python.

- **Temas e Aparência**: Permitir que o usuário escolha entre temas Light/Dark.
- **Opção de Salvar/Carregar**: Implementar funcionalidade para salvar os dados de uma simulação e carregá-los posteriormente.
- **Empacotamento**: Usar ferramentas como PyInstaller ou cx_Freeze para criar um executável independente que possa ser distribuído facilmente.

Lembre-se, cada projeto é uma oportunidade de aprendizado. Não tenha medo de experimentar, quebrar coisas e, acima de tudo, persistir!

Conclusão

Construir o Simulador de Honorários Advocatícios foi uma jornada prática que nos levou desde a definição de regras de negócio complexas até a criação de uma interface gráfica funcional e a geração de relatórios profissionais. Vimos a importância de separar a lógica da interface, o poder de bibliotecas como CustomTkinter e ReportLab, e a realidade inevitável (e superável!) da depuração.

Esperamos que este passo a passo tenha sido útil e inspirador. O mais importante é começar, dividir o problema em partes menores e celebrar cada pequena vitória ao longo do caminho. A jornada da programação é feita de desafios, mas a satisfação de ver sua criação funcionando e resolvendo um problema real é imensa. Agora é com você: que tal tentar construir sua própria versão ou adaptar esses conceitos para outro projeto?

Apêndice: Código-Fonte Completo
A seguir, apresentamos os scripts Python utilizados no projeto.

a) Main_gui.py

```
import customtkinter
from gui.app import App # Importa a classe principal da GUI
# import locale # Mantido comentado, pois não era a causa e pode ser
adicionado se necessário

# --- DESATIVAR DPI AUTOMÁTICO (CORREÇÃO NECESSÁRIA) ---
# Esta linha é crucial para evitar o erro 'bad screen distance' no seu
ambiente.
customtkinter.deactivate_automatic_dpi_awareness()
# -------------------------------------------------

if __name__ == "__main__":
    # Configurações iniciais do CustomTkinter (Aparência e Tema)
    customtkinter.set_appearance_mode("Light") # Modes: "System"
(default), "Dark", "Light"
    customtkinter.set_default_color_theme("blue") # Themes: "blue"
(default), "green", "dark-blue"

    # Cria e executa a aplicação GUI
    app = App()
    app.mainloop()
```

b) app.py

```
import customtkinter as ctk
from tkinter import messagebox
from datetime import date, datetime
import os
import unicodedata # Para normalizar nomes de chave

# Importa a lógica de negócio, utilitários e constantes
from core import formulas, utils, constants
from reports import pdf_generator # Para gerar o PDF

# --- Função Auxiliar para Normalização de Chaves ---
def normalize_key(text):
    """Remove acentos, converte para minúsculas e substitui
espaços/hífens por underscores."""
    try:
        nfkd_form = unicodedata.normalize('NFKD', text)
        text_sem_acentos = "".join([c for c in nfkd_form if not
unicodedata.combining(c)])
```

```
        return text_sem_acentos.lower().replace("-", "_").replace(" ",
"_")
    except TypeError: # Caso receba algo que não seja string
        return str(text).lower().replace("-", "_").replace(" ", "_")

# --- Classe Principal da Aplicação GUI ---
class App(ctk.CTk):
    def __init__(self):
        """Inicializa a janela principal e todos os seus widgets."""
        super().__init__()

        self.title("Simulador de Honorários Advocatícios")
        # Ajuste fino da geometria pode ser necessário dependendo da
fonte/resolução
        self.geometry("750x700")

        # --- Definição de Fontes para consistência e compactação ---
        self.default_font = ctk.CTkFont(size=12)
        self.label_font = ctk.CTkFont(size=12)
        self.entry_font = ctk.CTkFont(size=12)
        self.title_font = ctk.CTkFont(size=13, weight="bold")

        # --- Dicionário para armazenar widgets de entrada para fácil
acesso ---
        self.entries = {}
        # --- Lista para armazenar widgets de entrada das Pós-Graduações
---
        self.pos_grad_entries = []

        # --- Configuração do Layout Principal da Janela ---
        # Faz a coluna 0 (onde o scroll frame estará) expandir com a
janela
        self.grid_columnconfigure(0, weight=1)
        # Faz a linha 0 (onde o scroll frame estará) expandir com a
janela
        self.grid_rowconfigure(0, weight=1)

        # --- Frame Principal com Barra de Rolagem ---
        # Contém a maioria dos campos de entrada
        self.scrollable_frame = ctk.CTkScrollableFrame(self,
label_text="Dados para Simulação")
        self.scrollable_frame.grid(row=0, column=0, padx=10, pady=10,
sticky="nsew")
        # Configura a coluna 1 DENTRO do scroll frame (onde ficam as
Entries) para expandir
        self.scrollable_frame.grid_columnconfigure(1, weight=1)
        # Configura a coluna 3 DENTRO do scroll frame para expandir
(usada na linha Valor Causa/Percentual)
        self.scrollable_frame.grid_columnconfigure(3, weight=1)
```

```
        # --- Variável para controlar a próxima linha disponível no grid
do scroll_frame ---
        row_idx = 0

        # --- Seção: Dados Pessoais e Formação ---
        ctk.CTkLabel(self.scrollable_frame, text="Dados Pessoais e
Formação", font=self.title_font).grid(
            row=row_idx, column=0, columnspan=4, pady=(0, 5), sticky="w")
# Span 4 colunas
        row_idx += 1
        self.entries['nome_advogado'] = self._create_entry_row("Nome do
Advogado:", row_idx)
        row_idx += 1
        self.entries['data_graduacao'] = self._create_entry_row("Data
Graduação (DD/MM/AAAA):", row_idx)
        row_idx += 1
        self.entries['data_oab'] = self._create_entry_row("Data Inscrição
OAB (DD/MM/AAAA):", row_idx)
        row_idx += 1

        # --- Seção: Pós-Graduações ---
        ctk.CTkLabel(self.scrollable_frame, text="Pós-Graduações",
font=self.title_font).grid(
            row=row_idx, column=0, columnspan=3, pady=(15, 5),
sticky="w") # Span 3 colunas
        # Botão para adicionar mais campos de pós
        add_pos_button = ctk.CTkButton(self.scrollable_frame, text="+
Pós", command=self._add_pos_grad_entry, width=60, font=self.default_font)
        add_pos_button.grid(row=row_idx, column=3, pady=(15,2),
padx=(0,5), sticky="e") # Coluna 3
        row_idx += 1
        # Frame interno para agrupar os campos de pós adicionados
dinamicamente
        self.pos_grad_frame = ctk.CTkFrame(self.scrollable_frame,
fg_color="transparent")
        self.pos_grad_frame.grid(row=row_idx, column=0, columnspan=4,
sticky="ew", pady=(0, 5)) # Span 4 colunas
        self.pos_grad_frame.grid_columnconfigure(1, weight=1) # Coluna da
entry dentro deste frame expande
        self.pos_grad_next_row = 0 # Contador de linha interno para o
pos_grad_frame
        self._add_pos_grad_entry() # Adiciona o primeiro campo
        row_idx += 1

        # --- Seção: Experiência Prática ---
        ctk.CTkLabel(self.scrollable_frame, text="Experiência Prática",
font=self.title_font).grid(
```

```
            row=row_idx, column=0, columnspan=4, pady=(15, 5),
sticky="w") # Span 4 colunas
        row_idx += 1
        # Ajuste o rótulo para ser claro sobre o que o valor representa
        self.entries['total_acoes_defendidas'] =
self._create_entry_row("Total Ações Atuadas (Carreira):", row_idx)
        row_idx += 1
        ctk.CTkLabel(self.scrollable_frame, text="Ações Ganhas por
Área:", font=self.label_font).grid(
            row=row_idx, column=0, columnspan=4, padx=(0,5), pady=(5,2),
sticky="w") # Span 4 colunas
        row_idx += 1
        # Frame interno para layout das ações ganhas (pode quebrar em
linhas)
        acoes_frame = ctk.CTkFrame(self.scrollable_frame,
fg_color="transparent")
        acoes_frame.grid(row=row_idx, column=0, columnspan=4,
sticky="ew", pady=(0, 5)) # Span 4 colunas
        # Configura colunas internas do acoes_frame para 2 pares
Label/Entry por linha
        acoes_frame.grid_columnconfigure(1, weight=1) # Coluna Entry 1
expande
        acoes_frame.grid_columnconfigure(3, weight=1) # Coluna Entry 2
expande
        acoes_row = 0
        acoes_col = 0
        for area in constants.AREAS_ATUACAO: # Itera sobre as áreas
definidas nas constantes
            key = f'acoes_ganhas_{normalize_key(area)}' # Gera chave
normalizada
            lab = ctk.CTkLabel(acoes_frame, text=f"{area}:",
font=self.label_font)
            ent = ctk.CTkEntry(acoes_frame, font=self.entry_font)
            lab.grid(row=acoes_row, column=acoes_col, padx=(5,2), pady=2,
sticky="e")
            ent.grid(row=acoes_row, column=acoes_col + 1, padx=(0, 10),
pady=2, sticky="ew")
            self.entries[key] = ent # Armazena a entry no dicionário
            acoes_col += 2 # Próximo par Label/Entry
            if acoes_col >= 4: # Se preencheu as 4 colunas, vai para a
próxima linha
                acoes_col = 0
                acoes_row += 1
        row_idx += 1

        # --- Seção: Investimento e Dedicação ---
        ctk.CTkLabel(self.scrollable_frame, text="Investimento e
Dedicação", font=self.title_font).grid(
```

```
            row=row_idx, column=0, columnspan=4, pady=(15, 5),
sticky="w") # Span 4 colunas
        row_idx += 1
        self.entries['gastos_educacao'] = self._create_entry_row("Gasto
Educação (Total R$):", row_idx)
        row_idx += 1
        # **IMPORTANTE**: Ajuste o rótulo para refletir o que as horas
extras significam (mensal, anual, etc.)
        self.entries['horas_trabalhadas_fds_total'] =
self._create_entry_row("Média Mensal Horas Extras:", row_idx)
        row_idx += 1

        # --- Seção: Dados do Serviço Específico ---
        ctk.CTkLabel(self.scrollable_frame, text="Dados do Serviço
Específico", font=self.title_font).grid(
            row=row_idx, column=0, columnspan=4, pady=(15, 5),
sticky="w") # Span 4 colunas
        row_idx += 1
        self.entries['area_servico_atual'] =
self._create_combobox_row("Área do Serviço:", constants.AREAS_ATUACAO,
row_idx)
        row_idx += 1
        self.entries['horas_estimadas_servico'] =
self._create_entry_row("Horas Estimadas TOTAIS:", row_idx)
        row_idx += 1
        self.entries['nivel_complexidade_servico'] =
self._create_combobox_row("Complexidade:", constants.NIVEIS_COMPLEXIDADE,
row_idx)
        row_idx += 1
        self.entries['nivel_urgencia_servico'] =
self._create_combobox_row("Urgência:", constants.NIVEIS_URGENCIA,
row_idx)
        row_idx += 1

        # --- Linha Combinada: Valor Causa e Percentual Êxito ---
        # Label Valor Causa
        ctk.CTkLabel(self.scrollable_frame, text="Valor Causa (Cliente -
R$, 0 se N/A):", font=self.label_font).grid(
            row=row_idx, column=0, padx=(0, 5), pady=2, sticky="w")
        # Entry Valor Causa
        entry_valor_causa = ctk.CTkEntry(self.scrollable_frame,
font=self.entry_font)
        entry_valor_causa.grid(row=row_idx, column=1, padx=0, pady=2,
sticky="ew")
        self.entries['valor_estimado_causa_ganha'] = entry_valor_causa
        # Label Percentual Êxito
        ctk.CTkLabel(self.scrollable_frame, text="Percentual Êxito (%):",
font=self.label_font).grid(
```

```
            row=row_idx, column=2, padx=(10, 5), pady=2, sticky="w") #
Padding à esquerda para separar
        # Entry Percentual Êxito
        entry_perc_exito = ctk.CTkEntry(self.scrollable_frame,
font=self.entry_font)
        entry_perc_exito.grid(row=row_idx, column=3, padx=0, pady=2,
sticky="ew")
        self.entries['percentual_exito'] = entry_perc_exito
        row_idx += 1

        # --- Seção: Parâmetros Base ---
        ctk.CTkLabel(self.scrollable_frame, text="Parâmetros Base
(Cálculo Horário)", font=self.title_font).grid(
            row=row_idx, column=0, columnspan=4, pady=(15, 5),
sticky="w") # Span 4 colunas
        row_idx += 1
        self.entries['taxa_horaria_base_minima'] =
self._create_entry_row("Taxa Horária Mínima (R$):", row_idx)
        row_idx += 1

        # --- Botão Calcular (Fora do Scroll Frame) ---
        # Posicionado na linha 1 da janela principal (self)
        calculate_button = ctk.CTkButton(self, text="Calcular e Gerar
PDF", command=self.calculate, font=self.default_font)
        calculate_button.grid(row=1, column=0, padx=10, pady=5,
sticky="ew")

        # --- Área de Resultados (Fora do Scroll Frame) ---
        # Posicionado na linha 2 da janela principal (self)
        self.results_label = ctk.CTkLabel(self, text="Resultados
aparecerão aqui...", wraplength=700, anchor="w", justify="left",
font=self.default_font)
        self.results_label.grid(row=2, column=0, padx=10, pady=(5, 10),
sticky="ew")

    # --- Métodos Auxiliares para Criação de Widgets ---
    def _create_entry_row(self, label_text, row_index):
        """Cria uma linha padrão com Label na coluna 0 e Entry na coluna
1."""
        label = ctk.CTkLabel(self.scrollable_frame, text=label_text,
font=self.label_font)
        # Coloca o label na coluna 0 da linha especificada
        label.grid(row=row_index, column=0, padx=(0, 5), pady=2,
sticky="w")
        entry = ctk.CTkEntry(self.scrollable_frame, font=self.entry_font)
        # Coloca a entry na coluna 1 da linha especificada, fazendo-a
expandir horizontalmente
        entry.grid(row=row_index, column=1, columnspan=3, padx=0, pady=2,
sticky="ew") # Span 3 colunas restantes
```

```
        return entry

    def _create_combobox_row(self, label_text, values, row_index):
        """Cria uma linha padrão com Label na coluna 0 e ComboBox na
coluna 1."""
        label = ctk.CTkLabel(self.scrollable_frame, text=label_text,
font=self.label_font)
        label.grid(row=row_index, column=0, padx=(0, 5), pady=2,
sticky="w")
        combobox = ctk.CTkComboBox(self.scrollable_frame, values=values,
state="readonly", font=self.entry_font, dropdown_font=self.entry_font)
        combobox.grid(row=row_index, column=1, columnspan=3, padx=0,
pady=2, sticky="ew") # Span 3 colunas restantes
        if values: # Define o primeiro valor como padrão, se houver
valores
            combobox.set(values[0])
        return combobox

    def _add_pos_grad_entry(self):
        """Adiciona dinamicamente uma nova linha para data de Pós-
Graduação."""
        row = self.pos_grad_next_row # Linha dentro do pos_grad_frame
        # Label curto para economizar espaço
        label = ctk.CTkLabel(self.pos_grad_frame, text=f"Pós
{len(self.pos_grad_entries) + 1} (DD/MM/AAAA):", font=self.label_font)
        label.grid(row=row, column=0, padx=(0, 5), pady=1, sticky="w") #
Coluna 0 do frame interno
        entry = ctk.CTkEntry(self.pos_grad_frame, font=self.entry_font)
        entry.grid(row=row, column=1, padx=0, pady=1, sticky="ew") #
Coluna 1 do frame interno
        self.pos_grad_entries.append(entry) # Adiciona a entry à lista
        self.pos_grad_next_row += 1 # Incrementa o contador de linha
interna

    # --- Método Principal de Cálculo e Geração de Relatório ---
    def calculate(self):
        """Coleta todos os dados da GUI, valida, chama os cálculos e
exibe/salva os resultados."""
        dados_input = {} # Dicionário para armazenar os dados coletados
        self.results_label.configure(text="Calculando...",
font=self.default_font) # Feedback

        try:
            # --- 1. Coleta e Validação dos Dados de Entrada ---

            # Coleta Nome (simples validação de não vazio)
            dados_input['nome_advogado'] =
self.entries['nome_advogado'].get().strip()
            if not dados_input['nome_advogado']:
```

```
                raise ValueError("Nome do Advogado não pode estar
vazio.")

            # Coleta e Validação de Datas
            data_graduacao_str =
self.entries['data_graduacao'].get().strip()
            dados_input['data_graduacao'] =
utils.parse_data(data_graduacao_str) # Usa helper de utils.py
            if not dados_input['data_graduacao'] or
dados_input['data_graduacao'] > date.today():
                 raise ValueError("Data de Graduação inválida ou futura.
Use DD/MM/AAAA.")

            data_oab_str = self.entries['data_oab'].get().strip()
            dados_input['data_oab'] = utils.parse_data(data_oab_str)
            if not dados_input['data_oab'] or dados_input['data_oab'] >
date.today():
                 raise ValueError("Data da OAB inválida ou futura. Use
DD/MM/AAAA.")
            if dados_input['data_oab'] < dados_input['data_graduacao']:
                 raise ValueError("Data da OAB não pode ser anterior à
data de graduação.")

            # Coleta e Validação das Datas de Pós-Graduação (pode haver
várias)
            dados_input['datas_pos_graduacao'] = []
            for i, entry in enumerate(self.pos_grad_entries):
                data_str = entry.get().strip()
                if data_str: # Processa apenas se o campo não estiver
vazio
                    data_obj = utils.parse_data(data_str)
                    if not data_obj or data_obj > date.today():
                         raise ValueError(f"Data de Pós {i+1}
('{data_str}') inválida/futura.")
                    if data_obj < dados_input['data_graduacao']:
                         raise ValueError(f"Data de Pós {i+1}
('{data_str}') anterior à graduação.")
                    if data_obj not in
dados_input['datas_pos_graduacao']: # Evita datas duplicadas
                         dados_input['datas_pos_graduacao'].append(data_o
bj)
            dados_input['datas_pos_graduacao'].sort() # Ordena as datas

            # Coleta e Validação de Campos Inteiros
            int_fields = { # Mapeia chave interna para rótulo amigável
                'total_acoes_defendidas': "Total Ações Atuadas
(Carreira)",
                'horas_trabalhadas_fds_total': "Média Mensal Horas
Extras", # Ajuste conforme o significado real
```

```
            }
            # Adiciona dinamicamente as chaves das ações ganhas por área
            for area in constants.AREAS_ATUACAO:
                key = f'acoes_ganhas_{normalize_key(area)}'
                int_fields[key] = f"Ações Ganhas ({area})"

            for key, label in int_fields.items():
                 if key not in self.entries: # Verificação de segurança
                      print(f"Aviso: Chave inteira '{key}' esperada não
encontrada em self.entries.")
                      dados_input[key] = 0 # Assume 0 se não encontrar
                      continue
                 try:
                     value_str = self.entries[key].get().strip()
                     dados_input[key] = int(value_str) if value_str else
0 # Converte para int, 0 se vazio
                     if dados_input[key] < 0: raise ValueError() # Não
permite negativos
                 except ValueError:
                     raise ValueError(f"Valor inválido para '{label}'.
Insira um número inteiro >= 0.")

            # Coleta e Validação de Campos Numéricos (float)
            float_fields = {
                 'gastos_educacao': "Gasto Educação (Total R$)",
                 'horas_estimadas_servico': "Horas Estimadas TOTAIS",
                 'valor_estimado_causa_ganha': "Valor Causa Cliente",
                 'taxa_horaria_base_minima': "Taxa Horária Mínima",
                 'percentual_exito': "Percentual Êxito (%)", # Nome
mantido para compatibilidade da chave
            }
            for key, label in float_fields.items():
                 if key not in self.entries: # Verificação de segurança
                      print(f"Aviso: Chave float '{key}' esperada não
encontrada em self.entries.")
                      dados_input[key] = 0.0 # Assume 0.0 se não
encontrar
                      continue
                 try:
                     # Trata vírgula como decimal e remove pontos de
milhar
                     value_str =
self.entries[key].get().strip().replace('.', '').replace(',', '.')
                     if not value_str: # Se vazio, considera 0.0
                          dados_input[key] = 0.0
                     else:
                          dados_input[key] = float(value_str) # Converte
para float
                          # Validações específicas de intervalo/mínimo
```

```
                        if key == 'horas_estimadas_servico' and
dados_input[key] < 0.1:
                            raise ValueError("Mínimo de 0.1 hora.")
                        elif key == 'taxa_horaria_base_minima' and
dados_input[key] < 1.0:
                            raise ValueError("Mínimo de R$ 1.00.")
                        elif key == 'percentual_exito' and not (0 <=
dados_input[key] <= 100):
                            raise ValueError("Percentual deve ser entre
0 e 100.") # Ajuste da mensagem
                        elif dados_input[key] < 0: # Outros campos
float não podem ser negativos
                             raise ValueError("Valor não pode ser
negativo.")
                except ValueError as e: # Captura erros de conversão ou
das validações acima
                    msg_base = f"Valor inválido para '{label}'. Insira
número >= 0"
                    if key == 'percentual_exito': msg_base += " (entre 0
e 100)"
                    # Adiciona a mensagem específica do erro se for uma
das nossas validações
                    msg_extra = str(e) if str(e).startswith(("Mínimo",
"Percentual deve", "Valor não pode")) else "" # Ajuste do check
                    raise ValueError(f"{msg_base}. {msg_extra}".strip())

            # Coleta dos valores dos ComboBoxes (seleção já garante valor
válido da lista)
            dados_input['area_servico_atual'] =
self.entries['area_servico_atual'].get()
            dados_input['nivel_complexidade_servico'] =
self.entries['nivel_complexidade_servico'].get()
            dados_input['nivel_urgencia_servico'] =
self.entries['nivel_urgencia_servico'].get()

            # --- 2. Validação Lógica Cruzada e Alertas de Confirmação --
-

            # Verifica se o total de ações ganhas não excede o total
atuado
            total_ganhas =
sum(dados_input.get(f'acoes_ganhas_{normalize_key(area)}', 0) for area in
constants.AREAS_ATUACAO)
            if total_ganhas > dados_input.get('total_acoes_defendidas',
0):
                raise ValueError(f"Total Ações Ganhas ({total_ganhas}) >
Total Atuadas ({dados_input.get('total_acoes_defendidas', 0)}).")
```

```
            # Alertas opcionais para valores altos (permite ao usuário
confirmar ou cancelar)
            # Utiliza limites definidos em core/constants.py
            if dados_input.get('horas_estimadas_servico', 0) >
constants.HORAS_SERVICO_ALERTA_LIMITE:
                if not messagebox.askyesno("Alerta de Esforço Alto",
f"Estimativa de {dados_input['horas_estimadas_servico']:.1f} horas parece
alta.\nDeseja prosseguir?"):
                    self.results_label.configure(text="Cálculo
cancelado. Revise as horas estimadas.")
                    return # Interrompe se o usuário clicar "Não"

            if dados_input.get('valor_estimado_causa_ganha', 0) >
constants.VALOR_CAUSA_ALERTA_LIMITE:
                if not messagebox.askyesno("Alerta de Valor de Causa
Alto", f"Valor da causa
({utils.formatar_moeda(dados_input['valor_estimado_causa_ganha'])})
parece alto.\nDeseja prosseguir?"):
                    self.results_label.configure(text="Cálculo
cancelado. Revise o valor da causa.")
                    return # Interrompe se o usuário clicar "Não"

            if dados_input.get('taxa_horaria_base_minima', 0) >
constants.TAXA_BASE_ALERTA_LIMITE:
                if not messagebox.askyesno("Alerta de Taxa Base Alta",
f"Taxa base
({utils.formatar_moeda(dados_input['taxa_horaria_base_minima'])}) parece
alta como MÍNIMO.\nDeseja prosseguir?"):
                    self.results_label.configure(text="Cálculo
cancelado. Revise a taxa base mínima.")
                    return # Interrompe se o usuário clicar "Não"

            # --- 3. Execução dos Cálculos Principais ---
            # Chama as funções do módulo 'core.formulas' para obter os
resultados

            # Calcula a taxa horária sugerida com base nos fatores de
experiência, etc.
            taxa_horaria_sugerida, detalhes_taxa =
formulas.calcular_taxa_horaria_sugerida(dados_input)

            # Calcula o preço final do serviço com base na taxa horária e
outros fatores do serviço atual
            preco_horario_sugerido, detalhes_preco_horario =
formulas.calcular_preco_final_servico(
                taxa_horaria_sugerida, dados_input
            )

            # --- INÍCIO DA LÓGICA MODIFICADA ---
```

```
            # Calcula um valor de referência aplicando o percentual
informado sobre o PREÇO HORÁRIO calculado
            valor_ref_percentual = None # Valor de referência calculado
com o percentual
            detalhes_valor_percentual = {} # Detalhes deste cálculo para
o PDF
            percentual_informado = dados_input.get('percentual_exito',
0.0) # Pega o % informado
            valor_causa_cliente =
dados_input.get('valor_estimado_causa_ganha', 0.0) # Pega valor original
(contexto)

            # Calcula o valor de referência se um percentual foi
informado
            if percentual_informado > 0:
                percentual_decimal = percentual_informado / 100.0
                # *** AQUI A MUDANÇA PRINCIPAL: Base do cálculo é
preco_horario_sugerido ***
                valor_ref_percentual = preco_horario_sugerido *
percentual_decimal
                # Armazena detalhes para o relatório PDF
                detalhes_valor_percentual = {
                    "Valor Estimado Causa Cliente": valor_causa_cliente,
# Para contexto no PDF
                    "Preço Base Horária (Usado p/ Cálculo %)":
preco_horario_sugerido, # Indica a base usada
                    "Percentual Informado (%)": percentual_informado, # O
percentual usado
                    "Valor Calculado (Ref. Percentual)":
valor_ref_percentual # O resultado do cálculo
                }
            # --- FIM DA LÓGICA MODIFICADA ---

            # Agrupa todos os resultados e detalhes em um dicionário para
passar ao gerador de PDF
            # Mantém as chaves originais ("preco_exito_sugerido",
"detalhes_preco_exito")
            # por consistência interna e para o gerador de PDF, mas os
valores são os novos.
            resultados_calculo = {
                "taxa_horaria_sugerida": taxa_horaria_sugerida,
                "preco_horario_sugerido": preco_horario_sugerido,
                "detalhes_taxa_horaria": detalhes_taxa,
                "detalhes_preco_horario": detalhes_preco_horario,
                "preco_exito_sugerido": valor_ref_percentual, # Usa o
novo valor calculado (ou None)
                "detalhes_preco_exito": detalhes_valor_percentual  # Usa
os novos detalhes (ou {})
            }
```

```
            # --- 4. Geração do Relatório PDF ---
            # Cria um nome de arquivo único com timestamp e nome
sanitizado
            timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
            nome_adv_sanit = "".join(c if c.isalnum() else "_" for c in
dados_input.get('nome_advogado', 'Advogado'))
            nome_arquivo_pdf =
f"Simulacao_Honorarios_{nome_adv_sanit}_{timestamp}.pdf"

            # Define a pasta de saída e a cria se não existir
            output_dir = "output"
            if not os.path.exists(output_dir):
                 os.makedirs(output_dir)

            # Monta o caminho completo para o arquivo PDF
            caminho_pdf = os.path.join(output_dir, nome_arquivo_pdf)

            # Chama a função do módulo 'reports.pdf_generator' para criar
o PDF
            pdf_generator.gerar_pdf_simulacao(caminho_pdf, dados_input,
resultados_calculo)

            # --- 5. Exibição dos Resultados na GUI ---
            # Formata o texto de resumo para mostrar na interface
            resultado_txt = f"Cálculo Concluído!\n\n"
            resultado_txt += f"Taxa Horária Sugerida:
{utils.formatar_moeda(taxa_horaria_sugerida)}\n"
            resultado_txt += f"Preço Sugerido (Base Horária):
{utils.formatar_moeda(preco_horario_sugerido)}\n"

            # --- INÍCIO DA EXIBIÇÃO MODIFICADA ---
            # Adiciona a linha do valor de referência percentual, se
calculado
            if valor_ref_percentual is not None:
                # Atualiza o texto para refletir que o % foi sobre a Base
Horária
                resultado_txt += f"Valor Ref.
({percentual_informado:.1f}% de Base Horária):
{utils.formatar_moeda(valor_ref_percentual)}\n"
            # --- FIM DA EXIBIÇÃO MODIFICADA ---

            resultado_txt += f"\nRelatório PDF gerado em: {caminho_pdf}"

            # Atualiza o label na interface
            self.results_label.configure(text=resultado_txt)
            # Mostra um pop-up de sucesso
            messagebox.showinfo("Sucesso", f"Simulação concluída e PDF
gerado:\n{caminho_pdf}")
```

```
        # --- Tratamento de Erros Esperados e Inesperados ---
        except ValueError as ve: # Erros de validação específicos
(formato, intervalo)
            messagebox.showerror("Erro de Validação", str(ve))
            self.results_label.configure(text=f"Erro: {str(ve)}",
font=self.default_font)
        except ImportError as ie: # Erros ao importar módulos (problema
de setup/instalação)
             messagebox.showerror("Erro de Importação", f"Erro ao
importar módulos: {ie}\nVerifique a instalação e a estrutura do
projeto.")
             self.results_label.configure(text="Erro interno
(importação).")
        except FileNotFoundError as fnfe: # Erro ao tentar criar/salvar o
PDF (permissão, caminho inválido)
             messagebox.showerror("Erro de Arquivo", f"Erro ao tentar
salvar o PDF: {fnfe}\nVerifique as permissões na pasta '{output_dir}'.")
             self.results_label.configure(text="Erro ao salvar PDF.")
        except KeyError as ke: # Erros se alguma chave esperada faltar
nos dicionários (erro de programação)
             messagebox.showerror("Erro Interno (Chave)", f"Erro ao
acessar dados internos: Chave '{ke}' não encontrada.\nContate o
desenvolvedor.")
             self.results_label.configure(text=f"Erro interno (chave
{ke}).")
             import traceback
             traceback.print_exc() # Ajuda a debugar no console
        except Exception as e: # Captura qualquer outro erro inesperado
            import traceback
            traceback.print_exc() # Imprime detalhes do erro no console
para depuração
            messagebox.showerror("Erro Inesperado", f"Ocorreu um
erro:\n{type(e).__name__}: {e}")
            self.results_label.configure(text="Ocorreu um erro
inesperado.", font=self.default_font)

# --- Fim da classe App ---
```

c) constants.py

```
# -*- coding: utf-8 -*-

"""
Arquivo de Constantes e Parâmetros Configuráveis
------------------------------------------------
Este arquivo centraliza todos os pesos, multiplicadores e limiares
utilizados nas fórmulas de cálculo de honorários, bem como listas
```

```
de opções usadas na GUI e validações. Ajuste estes
valores para calibrar o simulador à sua realidade e mercado.
"""

# --- Fator Experiência Temporal ---
PESO_ANO_POS_OAB: float = 0.035      # % adicional por ano de prática
efetiva (pós-OAB)
PESO_ANO_POS_GRAD: float = 0.015    # % adicional por ano médio desde a
conclusão das pós

# --- Fator Especialização e Pós-Graduação ---
VALOR_BASE_POS_GRAD: float = 0.08   # Bônus base por CADA pós-graduação
concluída

# --- Fator Experiência Prática (Casos) ---
PESO_VOLUME_ACOES: float = 0.05    # Peso para o logaritmo do volume
total de ações
PESO_TAXA_SUCESSO_GERAL: float = 0.40 # Peso da taxa de sucesso geral (0
a 1)
PESO_TAXA_SUCESSO_ESPECIFICA: float = 0.60 # Peso da taxa de sucesso na
área do serviço (0 a 1) - Maior peso (usado como proxy da geral)
MIN_ACOES_PARA_SUCESSO_GERAL: int = 10 # Mínimo de ações totais para
considerar a taxa de sucesso geral significativa

# --- Fator Investimento Educacional ---
PESO_GASTO_EDUCACAO_SOBRE_TAXA_MIN_ANUAL: float = 0.10 # Peso relativo ao
gasto anual com taxa mínima
HORAS_NORMAIS_POR_ANO: int = 1800 # Base para cálculo da receita anual
mínima

# --- Fator Dedicação (Horas Extras) ---
PESO_DEDICACAO_HORAS_FDS: float = 0.20 # Peso para a proporção de horas
FDS sobre horas normais

# --- Multiplicadores do Serviço Específico (Base Horária) ---
MULTIPLICADOR_COMPLEXIDADE: dict[str, float] = {
    "Baixa": 1.0,
    "Média": 1.2,
    "Alta": 1.5,
    "Muito Alta": 1.8,
    "Excepcional": 2.2
}
MULTIPLICADOR_URGENCIA: dict[str, float] = {
    "Normal": 1.0,
    "Moderada": 1.15,
    "Alta": 1.35,
    "Imediata": 1.6
}
```

```
# --- Parâmetros para Cálculo de Êxito ---
PERCENTUAL_EXITO_PADRAO: float = 0.30 # Percentual padrão aplicado sobre
o valor ganho pelo cliente (0.30 = 30%)

# --- Limites para Alertas/Validações na GUI ---
HORAS_SERVICO_ALERTA_LIMITE: float = 100.0 # Acima disso, pede
confirmação na GUI
TAXA_BASE_ALERTA_LIMITE: float = 300.0 # Acima disso, pede confirmação na
GUI
VALOR_CAUSA_ALERTA_LIMITE: float = 1000000.0 # Acima disso, pede
confirmação na GUI

# --- Listas de Opções para GUI (ComboBoxes) ---
AREAS_ATUACAO: list[str] = ["Previdenciária", "Empresarial", "Civil",
"Trabalhista", "Tributária", "Outra"]
NIVEIS_COMPLEXIDADE: list[str] = list(MULTIPLICADOR_COMPLEXIDADE.keys())
NIVEIS_URGENCIA: list[str] = list(MULTIPLICADOR_URGENCIA.keys())
```

d) formulas.py

```
import math
from datetime import date
# Importa o módulo de constantes e as funções utilitárias
import core.constants as constants
from core.utils import calcular_anos_desde, formatar_numero,
formatar_moeda

# (As definições de constantes foram MOVIDAS para constants.py)

# --- Funções de Cálculo dos Fatores (agora usam constants.) ---

def calcular_fator_tempo_experiencia(data_graduacao: date, data_oab:
date, datas_pos: list[date]) -> tuple[float, dict]:
    """Calcula o fator baseado no tempo de formado, prática (OAB) e pós-
graduações."""
    anos_desde_grad = calcular_anos_desde(data_graduacao)
    anos_desde_oab = calcular_anos_desde(data_oab) # Prática efetiva

    anos_desde_pos_lista = [calcular_anos_desde(dt) for dt in datas_pos
if dt]
    anos_desde_pos_media = sum(anos_desde_pos_lista) /
len(anos_desde_pos_lista) if anos_desde_pos_lista else 0

    # Fórmula: Base 1 + Bônus por ano de OAB + Bônus por ano médio de pós
    fator = 1.0 + (constants.PESO_ANO_POS_OAB * anos_desde_oab) \
                + (constants.PESO_ANO_POS_GRAD * anos_desde_pos_media)

    detalhes = {
        "Anos desde Graduação": formatar_numero(anos_desde_grad, 1),
```

```
        "Anos desde Inscrição OAB": formatar_numero(anos_desde_oab, 1),
        "Anos Médios desde Pós-Graduações":
formatar_numero(anos_desde_pos_media, 1) if anos_desde_pos_lista else
"N/A",
        "Peso Ano Pós-OAB (const)": constants.PESO_ANO_POS_OAB,
        "Peso Ano Médio Pós (const)": constants.PESO_ANO_POS_GRAD
    }
    return max(1.0, fator), detalhes

def calcular_fator_especializacao_pos(datas_pos: list[date]) ->
tuple[float, dict]:
    """Calcula o fator baseado na quantidade de pós-graduações."""
    num_pos = len(datas_pos)
    # Fórmula: Base 1 + Bônus por cada pós
    fator = 1.0 + (constants.VALOR_BASE_POS_GRAD * num_pos)

    detalhes = {
        "Número de Pós-Graduações": num_pos,
        "Bônus por Pós-Graduação (const)": constants.VALOR_BASE_POS_GRAD
    }
    return max(1.0, fator), detalhes

def calcular_fator_experiencia_pratica(
    total_acoes: int,
    ganhas_prev: int, ganhas_emp: int, ganhas_civil: int, ganhas_trab:
int, ganhas_trib: int, ganhas_outras: int, # Adicionado Trab e Trib
    area_servico_atual: str
) -> tuple[float, dict]:
    """Calcula o fator baseado no volume e sucesso em ações judiciais."""
    total_ganhas = ganhas_prev + ganhas_emp + ganhas_civil + ganhas_trab
+ ganhas_trib + ganhas_outras
    taxa_sucesso_geral = (total_ganhas / total_acoes) if total_acoes >=
constants.MIN_ACOES_PARA_SUCESSO_GERAL else 0.0

    # --- Taxa de sucesso específica ---
    # Idealmente, o input coletaria o TOTAL de ações atuadas POR ÁREA.
    # Sem isso, a taxa específica fica prejudicada.
    # Simplificação: Usaremos a taxa geral, mas o peso dela será maior.
    # (Mantendo a lógica anterior para simplificar, mas ciente da
limitação)
    taxa_sucesso_especifica_proxy = taxa_sucesso_geral # Usando a geral
como proxy
    ganhas_na_area_atual = 0
    if area_servico_atual == "Previdenciária": ganhas_na_area_atual =
ganhas_prev
    elif area_servico_atual == "Empresarial": ganhas_na_area_atual =
ganhas_emp
    elif area_servico_atual == "Civil": ganhas_na_area_atual =
ganhas_civil
```

```
    elif area_servico_atual == "Trabalhista": ganhas_na_area_atual =
ganhas_trab
    elif area_servico_atual == "Tributária": ganhas_na_area_atual =
ganhas_trib
    elif area_servico_atual == "Outra": ganhas_na_area_atual =
ganhas_outras
    # Nota: Uma taxa específica real seria:
    # total_acoes_na_area = obter_input_int(f"Total de ações atuadas na
área {area_servico_atual}:")
    # taxa_sucesso_especifica = (ganhas_na_area_atual /
total_acoes_na_area) if total_acoes_na_area >=
constants.MIN_ACOES_PARA_SUCESSO_ESPECIFICO else 0.0

    # Usar logaritmo natural + 1 para o volume, suaviza o impacto de
números muito grandes
    log_volume = math.log(total_acoes + 1) if total_acoes > 0 else 0

    # Fórmula: Base 1 + Bônus Volume (Log) + Bônus Sucesso Geral + Bônus
Sucesso Específico (Proxy)
    fator = 1.0 + (constants.PESO_VOLUME_ACOES * log_volume) \
                + (constants.PESO_TAXA_SUCESSO_GERAL *
taxa_sucesso_geral) \
                + (constants.PESO_TAXA_SUCESSO_ESPECIFICA *
taxa_sucesso_especifica_proxy)

    detalhes = {
        "Total de Ações Atuadas": total_acoes,
        "Total de Ações Ganhas": total_ganhas,
        f"Ações Ganhas em {area_servico_atual}": ganhas_na_area_atual,
        "Taxa de Sucesso Geral Estimada": f"{taxa_sucesso_geral:.1%}" if
total_acoes >= constants.MIN_ACOES_PARA_SUCESSO_GERAL else f"<
{constants.MIN_ACOES_PARA_SUCESSO_GERAL} ações",
        "Taxa Sucesso Específica Usada (Proxy)":
f"{taxa_sucesso_especifica_proxy:.1%}",
        "Log(Volume + 1)": formatar_numero(log_volume, 3),
        "Peso Volume (const)": constants.PESO_VOLUME_ACOES,
        "Peso Sucesso Geral (const)": constants.PESO_TAXA_SUCESSO_GERAL,
        "Peso Sucesso Específico (const)":
constants.PESO_TAXA_SUCESSO_ESPECIFICA,
        "Mínimo Ações Sucesso Geral (const)":
constants.MIN_ACOES_PARA_SUCESSO_GERAL,
    }
    return max(1.0, fator), detalhes


def calcular_fator_investimento_educacional(gastos_educacao: float,
taxa_horaria_minima: float) -> tuple[float, dict]:
    """Calcula o fator baseado no investimento em educação."""
```

```
    if taxa_horaria_minima <= 0: return 1.0, {"Detalhe": "Taxa horária
mínima inválida."}

    # Estimativa de receita anual baseada na taxa mínima
    receita_anual_minima_estimada = taxa_horaria_minima *
constants.HORAS_NORMAIS_POR_ANO

    if receita_anual_minima_estimada <= 0: return 1.0, {"Detalhe":
"Receita anual mínima estimada inválida."}

    # Proporção do gasto educacional sobre a receita anual mínima
    proporcao_gasto_receita = gastos_educacao /
receita_anual_minima_estimada

    # Fórmula: Base 1 + Peso * Proporção
    fator = 1.0 + (constants.PESO_GASTO_EDUCACAO_SOBRE_TAXA_MIN_ANUAL *
proporcao_gasto_receita)

    detalhes = {
        "Gasto Total com Educação": formatar_moeda(gastos_educacao),
        "Taxa Horária Mínima Informada":
formatar_moeda(taxa_horaria_minima),
        "Receita Anual Mínima Estimada":
formatar_moeda(receita_anual_minima_estimada),
        "Proporção Gasto/Receita Anual Min.":
f"{proporcao_gasto_receita:.2%}",
        "Peso Gasto Educação (const)":
constants.PESO_GASTO_EDUCACAO_SOBRE_TAXA_MIN_ANUAL,
        "Horas Normais/Ano (const)": constants.HORAS_NORMAIS_POR_ANO,
    }
    return max(1.0, fator), detalhes


def calcular_fator_dedicacao(horas_fds: int, data_oab: date) ->
tuple[float, dict]:
    """Calcula o fator baseado nas horas trabalhadas em fins de
semana/feriados."""
    anos_desde_oab = calcular_anos_desde(data_oab)
    if anos_desde_oab <= 0:
        return 1.0, {"Detalhe": "Menos de um ano de prática (OAB)."}

    total_horas_normais_estimadas_carreira = anos_desde_oab *
constants.HORAS_NORMAIS_POR_ANO
    if total_horas_normais_estimadas_carreira <= 0:
         return 1.0, {"Detalhe": "Horas normais estimadas inválidas."}

    proporcao_horas_fds = horas_fds /
total_horas_normais_estimadas_carreira if
total_horas_normais_estimadas_carreira > 0 else 0
```

```
    # Fórmula: Base 1 + Peso * Proporção
    fator = 1.0 + (constants.PESO_DEDICACAO_HORAS_FDS *
proporcao_horas_fds)

    detalhes = {
        "Total Horas Estimadas FDS/Feriados": horas_fds,
        "Anos de Prática (OAB)": formatar_numero(anos_desde_oab, 1),
        "Total Horas Normais Estimadas (Carreira)":
formatar_numero(total_horas_normais_estimadas_carreira, 0),
        "Proporção Horas FDS / Normais": f"{proporcao_horas_fds:.2%}",
        "Peso Dedicação (Horas FDS) (const)":
constants.PESO_DEDICACAO_HORAS_FDS,
        "Horas Normais/Ano (const)": constants.HORAS_NORMAIS_POR_ANO,
    }
    return max(1.0, fator), detalhes


# --- Funções Principais de Cálculo ---

def calcular_taxa_horaria_sugerida(dados: dict) -> tuple[float, dict]:
    """Calcula a taxa horária final combinando todos os fatores."""
    taxa_base = dados['taxa_horaria_base_minima']
    fatores_aplicados = {}
    detalhes_fatores = {}

    fator_tempo, det_tempo = calcular_fator_tempo_experiencia(
        dados['data_graduacao'], dados['data_oab'],
dados['datas_pos_graduacao']
    )
    fatores_aplicados['Tempo Experiência'] = fator_tempo
    detalhes_fatores['Tempo Experiência'] = det_tempo

    fator_pos, det_pos =
calcular_fator_especializacao_pos(dados['datas_pos_graduacao'])
    fatores_aplicados['Especialização (Pós)'] = fator_pos
    detalhes_fatores['Especialização (Pós)'] = det_pos

    fator_pratica, det_pratica = calcular_fator_experiencia_pratica(
        dados['total_acoes_defendidas'],
        dados['acoes_ganhas_previdenciaria'],
dados['acoes_ganhas_empresarial'],
        dados['acoes_ganhas_civil'], dados['acoes_ganhas_trabalhista'], #
Adicionado
        dados['acoes_ganhas_tributaria'], dados['acoes_ganhas_outra'], #
Adicionado 'acoes_ganhas_outra' 'outra' no sigular
        dados['area_servico_atual']
    )
    fatores_aplicados['Experiência Prática (Casos)'] = fator_pratica
```

```
    detalhes_fatores['Experiência Prática (Casos)'] = det_pratica

    fator_invest_edu, det_invest =
calcular_fator_investimento_educacional(
        dados['gastos_educacao'], dados['taxa_horaria_base_minima']
    )
    fatores_aplicados['Investimento Educacional'] = fator_invest_edu
    detalhes_fatores['Investimento Educacional'] = det_invest

    fator_dedic, det_dedic = calcular_fator_dedicacao(
        dados['horas_trabalhadas_fds_total'], dados['data_oab']
    )
    fatores_aplicados['Dedicação (Horas Extras)'] = fator_dedic
    detalhes_fatores['Dedicação (Horas Extras)'] = det_dedic

    taxa_calculada = taxa_base
    print("\nAplicando Fatores à Taxa Base:")
    print(f"- Taxa Base Informada: {formatar_moeda(taxa_base)}")
    for nome, fator in fatores_aplicados.items():
        taxa_calculada *= fator
        print(f"- Fator {nome}: {formatar_numero(fator, 3)}x")

    detalhes_calculo_taxa = {
        "Taxa Horária Base Informada": taxa_base,
        "Fatores Multiplicadores": fatores_aplicados,
        "Detalhes dos Fatores": detalhes_fatores,
        "Taxa Horária Calculada Bruta": taxa_calculada
    }

    # Aplica um arredondamento ou ajuste final se desejado
    # taxa_final_ajustada = round(taxa_calculada / 5) * 5 # Ex: Múltiplo
de 5
    taxa_final_ajustada = taxa_calculada # Sem ajuste por enquanto

    detalhes_calculo_taxa["Taxa Horária Sugerida Final"] =
taxa_final_ajustada

    print(f"\n=> Taxa Horária Sugerida:
{formatar_moeda(taxa_final_ajustada)}")

    return taxa_final_ajustada, detalhes_calculo_taxa


def calcular_preco_final_servico(taxa_horaria_sugerida: float,
dados_servico: dict) -> tuple[float, dict]:
    """Calcula o preço final do serviço aplicando complexidade e
urgência."""
    horas = dados_servico['horas_estimadas_servico']
    complexidade_str = dados_servico['nivel_complexidade_servico']
```

```
    urgencia_str = dados_servico['nivel_urgencia_servico']

    fator_complexidade =
constants.MULTIPLICADOR_COMPLEXIDADE.get(complexidade_str, 1.0)
    fator_urgencia = constants.MULTIPLICADOR_URGENCIA.get(urgencia_str,
1.0)

    preco_base = taxa_horaria_sugerida * horas
    preco_final = preco_base * fator_complexidade * fator_urgencia

    print(f"\nCalculando Preço do Serviço Específico:")
    print(f"- Horas Estimadas: {formatar_numero(horas, 1)} h")
    print(f"- Nível Complexidade: {complexidade_str}
({fator_complexidade:.2f}x)")
    print(f"- Nível Urgência: {urgencia_str} ({fator_urgencia:.2f}x)")
    print(f"- Preço Base (Taxa * Horas): {formatar_moeda(preco_base)}")
    print(f"=> Preço Final Sugerido (Base * Complexidade * Urgência):
{formatar_moeda(preco_final)}")

    detalhes = {
        "Taxa Horária Utilizada": taxa_horaria_sugerida,
        "Horas Estimadas": horas,
        "Nível de Complexidade": complexidade_str,
        "Fator Complexidade": fator_complexidade,
        "Nível de Urgência": urgencia_str,
        "Fator Urgência": fator_urgencia,
        "Preço Base (Taxa * Horas)": preco_base,
        "Preço Final Sugerido": preco_final
    }
    return preco_final, detalhes
```

e) inputs.py

```
from datetime import date
# Importa utils E constants agora
from core.utils import parse_data, formatar_moeda # Importar
formatar_moeda para o alerta
from core import constants # Para usar as listas de opções

# (Funções obter_input_* e obter_input_opcao permanecem as mesmas)
# ... (Inclua as funções obter_input_float, obter_input_int, etc. aqui)
...
def obter_input_float(mensagem: str, minimo: float | None = 0.0) ->
float:
    """Obtém um input numérico (float) do usuário com tratamento de
erro."""
    while True:
        try:
```

```
            # Use replace('.', '') para remover separador de milhar se
existir
            valor_str = input(mensagem).replace('.', '').replace(',',
'.')
            valor = float(valor_str)
            if minimo is None or valor >= minimo:
                return valor
            else:
                print(f"Valor inválido. O valor mínimo é {minimo}.")
        except ValueError:
            print("Entrada inválida. Por favor, insira um número
válido.")

def obter_input_int(mensagem: str, minimo: int | None = 0) -> int:
    """Obtém um input numérico (int) do usuário com tratamento de
erro."""
    while True:
        try:
            valor = int(input(mensagem))
            if minimo is None or valor >= minimo:
                return valor
            else:
                print(f"Valor inválido. O valor mínimo é {minimo}.")
        except ValueError:
            print("Entrada inválida. Por favor, insira um número
inteiro.")

def obter_input_data(mensagem: str) -> date | None:
    """Obtém uma data do usuário no formato dd/mm/aaaa."""
    while True:
        data_str = input(mensagem + " (formato DD/MM/AAAA): ").strip()
        if not data_str:
            print("Data inválida. Não pode ser vazia.") # Mensagem mais
clara
            continue
        data_obj = parse_data(data_str)
        if data_obj:
            if data_obj > date.today():
                print("Data inválida. A data não pode ser no futuro.")
            else:
                return data_obj
        else:
            # Tenta dar uma dica se o formato estiver próximo mas errado
            if len(data_str) == 8 and data_str.isdigit():
                 hint = f"{data_str[:2]}/{data_str[2:4]}/{data_str[4:]}"
                 print(f"Formato de data inválido. Use DD/MM/AAAA (ex:
{hint}).")
            elif "/" not in data_str and "-" not in data_str and
len(data_str) > 5:
```

```
                print("Formato de data inválido. Use DD/MM/AAAA
(separado por /).")
            else:
                print("Formato de data inválido. Use DD/MM/AAAA.")

def obter_input_sim_nao(mensagem: str) -> bool:
    """Obtém uma resposta Sim/Não do usuário."""
    while True:
        resposta = input(mensagem + " (S/N): ").strip().upper()
        if resposta == 'S':
            return True
        elif resposta == 'N':
            return False
        else:
            print("Resposta inválida. Por favor, digite S ou N.")

def obter_input_opcao(mensagem: str, opcoes: list) -> str: # Simplificado
para aceitar apenas lista agora
    """Obtém uma escolha de uma lista de opções."""
    opcoes_dict = {}
    print(mensagem)
    for i, opcao in enumerate(opcoes):
        print(f"  {i+1}. {opcao}")
        opcoes_dict[str(i+1)] = opcao
        opcoes_dict[opcao.lower()] = opcao # Para aceitar digitação do
nome

    while True:
        escolha = input("Sua escolha (número ou nome): ").strip().lower()
        if escolha in opcoes_dict:
            return opcoes_dict[escolha] # Retorna o nome original da
opção
        else:
             print(f"Opção inválida. Escolha um número ou um dos nomes
listados.")


# --- Função Principal de Coleta ---

def coletar_dados_simulacao() -> dict:
    """Coleta todos os dados necessários para a simulação, com
validações."""
    print("\n--- Simulador de Honorários Advocatícios ---")
    print("Por favor, forneça as informações abaixo.")

    dados = {}

    print("\n--- Dados Pessoais e de Formação ---")
```

```
    dados['nome_advogado'] = input("Nome do(a) Advogado(a) (Opcional,
para o relatório): ").strip()
    dados['data_graduacao'] = obter_input_data("Data de Conclusão do
Bacharelado em Direito")
    dados['data_oab'] = obter_input_data("Data de Aprovação na OAB
(Inscrição Definitiva)")

    dados['datas_pos_graduacao'] = []
    print("\n--- Cursos de Pós-Graduação (Lato Sensu ou Stricto Sensu) --
-")
    while obter_input_sim_nao("Deseja adicionar uma data de certificação
de pós-graduação?"):
        data_pos = obter_input_data("  Data de Certificação da Pós-
Graduação")
        if data_pos:
             if data_pos not in dados['datas_pos_graduacao']:
                dados['datas_pos_graduacao'].append(data_pos)
             else:
                print("  Data já adicionada.")

    print("\n--- Experiência Prática ---")
    # Loop de validação para ações totais vs ganhas
    while True:
        dados['total_acoes_defendidas'] = obter_input_int("Quantidade
TOTAL de ações defendidas/atuadas na justiça: ")

        print("Quantidade de ações GANHAS (com êxito para o cliente):")
        dados['acoes_ganhas_previdenciaria'] = obter_input_int("  - Área
Previdenciária: ")
        dados['acoes_ganhas_empresarial'] = obter_input_int("  - Área
Empresarial: ")
        dados['acoes_ganhas_civil'] = obter_input_int("  - Área Civil
(inclui família, consumidor, etc.): ")
        dados['acoes_ganhas_trabalhista'] = obter_input_int("  - Área
Trabalhista: ")
        dados['acoes_ganhas_tributaria'] = obter_input_int("  - Área
Tributária: ")
        dados['acoes_ganhas_outras'] = obter_input_int("  - Outras Áreas:
")

        total_ganhas = (
            dados['acoes_ganhas_previdenciaria'] +
dados['acoes_ganhas_empresarial'] +
            dados['acoes_ganhas_civil'] +
dados['acoes_ganhas_trabalhista'] +
            dados['acoes_ganhas_tributaria'] +
dados['acoes_ganhas_outras']
        )
```

```
        if total_ganhas > dados['total_acoes_defendidas']:
            print("\n" + "="*25 + " ERRO DE VALIDAÇÃO " + "="*25)
            print(f"  O total de ações ganhas informado ({total_ganhas})
é MAIOR que")
            print(f"  o total de ações atuadas informado
({dados['total_acoes_defendidas']}).")
            print("  Isto é inconsistente. Por favor, revise e insira
novamente os dados de experiência.")
            print("="*70 + "\n")
            # Continue no loop para pedir os dados de experiência
novamente
        else:
            break # Dados consistentes, sair do loop de validação

    print("\n--- Investimento e Dedicação ---")
    dados['gastos_educacao'] = obter_input_float("Gasto TOTAL estimado
com educação formal (Graduação, Pós, Cursos, Livros) (R$): ")
    dados['horas_trabalhadas_fds_total'] = obter_input_int("Estimativa de
TOTAL de horas trabalhadas em Finais de Semana/Feriados ao longo da
carreira: ")

    print("\n--- Dados do Serviço a Precificar ---")
    dados['area_servico_atual'] = obter_input_opcao("Área Principal do
Serviço ATUAL:", constants.AREAS_ATUACAO)

    # Alerta para horas estimadas altas
    while True:
        dados['horas_estimadas_servico'] = obter_input_float("Horas
Estimadas para este Serviço Específico: ", minimo=0.1)
        if dados['horas_estimadas_servico'] >
constants.HORAS_SERVICO_ALERTA_LIMITE:
            horas_estimadas_fmt = dados['horas_estimadas_servico']
            limite_fmt = constants.HORAS_SERVICO_ALERTA_LIMITE
            print(f"\n--- ALERTA DE VALOR ALTO ---")
            # Mensagem mais clara:
            print(f"  A estimativa de {horas_estimadas_fmt:.1f} horas
parece ALTA como o **esforço TOTAL**")
            print(f"  necessário para concluir este serviço específico.")
            print(f"  (O limite configurado para alerta é
{limite_fmt:.1f} horas totais para um serviço).")
            print(f"  Valores muito altos aqui (representando todo o
trabalho para este caso/serviço)")
            print(f"  podem levar a um preço final exagerado.")
            print(f"  Certifique-se de que esta estimativa cobre todo o
trabalho esperado para ESTE serviço.")
            if not obter_input_sim_nao("  Deseja prosseguir com este
número TOTAL de horas?"):
                continue # Volta para pedir as horas novamente
```

```
        break # Sai do loop se o valor for baixo ou se o usuário
confirmar

    dados['nivel_complexidade_servico'] = obter_input_opcao("Nível de
Complexidade deste Serviço:", constants.NIVEIS_COMPLEXIDADE)
    dados['nivel_urgencia_servico'] = obter_input_opcao("Nível de
Urgência deste Serviço:", constants.NIVEIS_URGENCIA)

    print("\n--- Parâmetros Base ---")
    # Alerta para taxa base alta
    while True:
        dados['taxa_horaria_base_minima'] = obter_input_float("Sua Taxa
Horária MÍNIMA (R$) (cobre custos básicos e retirada mínima desejada): ",
minimo=1.0)
        if dados['taxa_horaria_base_minima'] >
constants.TAXA_BASE_ALERTA_LIMITE:
            taxa_formatada =
formatar_moeda(dados['taxa_horaria_base_minima'])
            limite_formatado =
formatar_moeda(constants.TAXA_BASE_ALERTA_LIMITE)
            print(f"\n--- ALERTA DE VALOR ALTO ---")
            print(f"  A taxa horária base de {taxa_formatada} parece ALTA
como valor MÍNIMO inicial.")
            print(f"  (O limite configurado para alerta é
{limite_formatado}).")
            print("  Lembre-se: este valor deve cobrir custos essenciais
+ retirada mínima.")
            print("  A valorização pela experiência, especialização,
etc., será adicionada pelos fatores.")
            print("  Uma base muito alta pode inflacionar o resultado
final.")
            if not obter_input_sim_nao("  Deseja prosseguir com esta taxa
base mínima?"):
                continue # Volta para pedir a taxa base novamente
        break # Sai do loop se o valor for baixo ou se o usuário
confirmar

    print("\n--- Coleta de Dados Concluída ---")
    return dados
```

f) utils.py

```
# -*- coding: utf-8 -*-

import locale
from datetime import datetime, date

# Flag para garantir que o locale seja configurado apenas uma vez
_locale_configurado = False
```

```
def configurar_locale_brasileiro():
    """Tenta configurar o locale para pt_BR.UTF-8 ou similar."""
    global _locale_configurado
    if _locale_configurado:
        return

    locales_tentativa = ['pt_BR.UTF-8', 'pt_BR.utf8',
'Portuguese_Brazil.1252', 'pt_BR', '']
    for loc in locales_tentativa:
        try:
            locale.setlocale(locale.LC_ALL, loc)
            print(f"Locale configurado para
'{locale.getlocale(locale.LC_ALL)[0]}'.")
            _locale_configurado = True
            return # Sucesso, sair da função
        except locale.Error:
            continue # Tentar próximo locale
    print(f"Aviso: Não foi possível configurar um locale pt_BR. Usando
locale padrão do sistema: '{locale.getlocale(locale.LC_ALL)[0]}'. A
formatação de moeda pode variar.")
    _locale_configurado = True # Marcar como configurado mesmo com
fallback

def formatar_moeda(valor: float | int | None) -> str:
    """Formata um valor numérico como moeda brasileira (R$)."""
    if valor is None:
        return "N/A"
    configurar_locale_brasileiro() # Garante que o locale está
configurado
    try:
        # Usa a formatação do locale se possível
        return locale.currency(valor, grouping=True, symbol=True)
    except (ValueError, locale.Error):
        # Fallback manual simples se o locale falhar
        try:
            return f"R$ {valor:,.2f}".replace(",", "X").replace(".",
",").replace("X", ".")
        except (TypeError, ValueError):
            return "Inválido" # Se o valor não for numérico

def formatar_numero(valor: float | int | None, casas_decimais: int = 2) -
> str:
    """Formata um número com separador de milhar brasileiro."""
    if valor is None:
        return "N/A"
    configurar_locale_brasileiro() # Garante que o locale está
configurado
    try:
```

```
        # Usa a formatação do locale se possível
        return locale.format_string(f"%.{casas_decimais}f", valor,
grouping=True)
    except (ValueError, locale.Error, OverflowError):
         # Fallback manual simples
        try:
            return f"{valor:,.{casas_decimais}f}".replace(",",
"X").replace(".", ",").replace("X", ".")
        except (TypeError, ValueError):
            return "Inválido"

def calcular_anos_desde(data_evento: date | None) -> float:
    """Calcula a diferença em anos fracionados entre uma data e hoje."""
    if not data_evento or not isinstance(data_evento, date):
        return 0.0
    hoje = date.today()
    if data_evento > hoje: # Não calcula para datas futuras
        return 0.0
    delta = hoje - data_evento
    # Usar 365.2425 para média mais precisa (ano tropical médio)
    return delta.days / 365.2425

def parse_data(data_str: str | None) -> date | None:
    """
    Converte string (DD/MM/AAAA, DD-MM-AAAA, YYYY-MM-DD, etc.) para
objeto date.
    Retorna None se a string for vazia, None ou o formato for inválido.
    """
    if not data_str:
        return None

    # Formatos comuns a tentar
    formatos_tentativa = ["%d/%m/%Y", "%d-%m-%Y", "%Y-%m-%d", "%d%m%Y"]

    for fmt in formatos_tentativa:
        try:
            # Tenta fazer o parse
            dt_obj = datetime.strptime(data_str.strip(), fmt)
            # Validação extra: verifica se a data é razoável (ex: não ano
10000)
            if dt_obj.year > date.today().year + 1 or dt_obj.year < 1900:
                 continue # Ignora datas muito no futuro ou muito antigas
            return dt_obj.date()
        except (ValueError, TypeError):
            continue # Tenta o próximo formato

    # Se nenhum formato funcionou
    return None
```

```
# Chama a configuração do locale na primeira importação deste módulo
configurar_locale_brasileiro()
```

g) pdf_generator.py

```
# -*- coding: utf-8 -*-

from reportlab.platypus import SimpleDocTemplate, Paragraph, Spacer,
PageBreak, KeepTogether
from reportlab.lib.styles import getSampleStyleSheet, ParagraphStyle
from reportlab.lib.enums import TA_CENTER, TA_LEFT, TA_JUSTIFY
from reportlab.lib.units import cm
from reportlab.lib.colors import navy, gray, black, darkblue
from datetime import datetime, date

# Importa utils e constants do core
from core.utils import formatar_moeda, formatar_numero
from core import constants # Necessário para exibir o percentual padrão
de êxito

def gerar_pdf_simulacao(nome_arquivo: str, dados_input: dict, calculos:
dict):
    """Gera um PDF com os resultados da simulação."""

    try:
        doc = SimpleDocTemplate(nome_arquivo, pagesize=(21*cm, 29.7*cm),
# A4
                                leftMargin=2*cm, rightMargin=2*cm,
                                topMargin=2*cm, bottomMargin=2*cm,
                                title=f"Simulação Honorários -
{dados_input.get('nome_advogado', 'Advogado')}",
                                author="Simulador de Honorários")
        styles = getSampleStyleSheet()
        story = []

        # --- Estilos Customizados ---
        style_titulo = ParagraphStyle(name='TituloPrincipal',
parent=styles['h1'], alignment=TA_CENTER, textColor=darkblue,
fontSize=18, spaceAfter=0.6*cm)
        style_subtitulo = ParagraphStyle(name='Subtitulo',
parent=styles['h2'], textColor=navy, fontSize=14, spaceBefore=0.8*cm,
spaceAfter=0.4*cm)
        style_normal = ParagraphStyle(name='NormalPDF',
parent=styles['Normal'], fontSize=10, leading=14, alignment=TA_JUSTIFY)
        style_label_valor = ParagraphStyle(name='LabelValor',
parent=styles['Normal'], fontSize=10, leading=14) # Para Label: Valor
        style_destaque = ParagraphStyle(name='Destaque',
parent=styles['h3'], fontSize=11, textColor=black, spaceBefore=0.4*cm,
spaceAfter=0.2*cm, alignment=TA_LEFT, fontName='Helvetica-Bold')
```

```
        style_final_preco = ParagraphStyle(name='PrecoFinal',
parent=styles['h2'], alignment=TA_CENTER, fontSize=15,
textColor=darkblue, spaceBefore=0.8*cm, spaceAfter=0.5*cm,
fontName='Helvetica-Bold')
        style_obs = ParagraphStyle(name='Observacoes',
parent=styles['Italic'], fontSize=9, textColor=gray, spaceBefore=1.5*cm,
alignment=TA_JUSTIFY, leading=12)

        # --- Conteúdo do PDF ---
        story.append(Paragraph("Simulador de Honorários Advocatícios",
style_titulo))
        story.append(Paragraph(f"Relatório Gerado em:
{datetime.now().strftime('%d/%m/%Y %H:%M:%S')}", styles['Normal']))
        nome_adv = dados_input.get('nome_advogado')
        if nome_adv:
            story.append(Paragraph(f"Advogado(a): {nome_adv}",
styles['Normal']))
        story.append(Spacer(1, 0.5*cm))

        # --- Função Auxiliar para adicionar linhas Label: Valor ---
        def add_info(label, valor, format_func=None, unit=""):
             valor_str = "N/A"
             if isinstance(valor, date):
                 valor_str = valor.strftime('%d/%m/%Y')
             elif isinstance(valor, list) and all(isinstance(item, date)
for item in valor):
                 valor_str = ", ".join([d.strftime('%d/%m/%Y') for d in
valor]) if valor else "Nenhuma"
             elif format_func:
                 valor_str = format_func(valor)
             elif valor is not None:
                  valor_str = str(valor)

             story.append(Paragraph(f"<b>{label}:</b> {valor_str}{unit}",
style_label_valor))


        # --- Dados de Entrada Consolidados ---
        story.append(Paragraph("Dados Informados na Simulação",
style_subtitulo))
        # Usar KeepTogether para tentar manter seções juntas na página
        entrada_section = []
        entrada_section.append(Paragraph("<u>Formação e
Experiência:</u>", style_destaque))
        add_info("Data Graduação Direito",
dados_input.get('data_graduacao'))
        add_info("Data Inscrição OAB", dados_input.get('data_oab'))
        add_info("Datas Pós-Graduações",
dados_input.get('datas_pos_graduacao', []))
```

```
        entrada_section.append(Spacer(1, 0.2*cm))
        add_info("Total Ações Atuadas",
dados_input.get('total_acoes_defendidas'), lambda v: formatar_numero(v,
0))
        total_ganhas = sum(dados_input.get(key, 0) for key in dados_input
if key.startswith('acoes_ganhas_'))
        add_info("Total Ações Ganhas (Informado)", total_ganhas, lambda
v: formatar_numero(v, 0))
        # Opcional: Detalhar ações ganhas por área
        # for area in constants.AREAS_ATUACAO:
        #    key = f'acoes_ganhas_{area.lower().replace("-","").replace("
","_")}'
        #    if key in dados_input and dados_input[key] > 0:
        #         add_info(f"  - Ganhas {area}", dados_input[key], lambda
v: formatar_numero(v, 0))
        entrada_section.append(Spacer(1, 0.2*cm))
        add_info("Gasto Estimado Educação",
dados_input.get('gastos_educacao'), formatar_moeda)
        add_info("Horas Estimadas FDS/Feriados (Carreira)",
dados_input.get('horas_trabalhadas_fds_total'), lambda v:
formatar_numero(v, 0))
        entrada_section.append(Spacer(1, 0.4*cm))

        entrada_section.append(Paragraph("<u>Dados do Serviço
Específico:</u>", style_destaque))
        add_info("Área Serviço Atual",
dados_input.get('area_servico_atual'))
        add_info("Horas Estimadas Totais (Esforço)",
dados_input.get('horas_estimadas_servico'), lambda v: formatar_numero(v,
1), unit=" h")
        add_info("Complexidade Serviço",
dados_input.get('nivel_complexidade_servico'))
        add_info("Urgência Serviço",
dados_input.get('nivel_urgencia_servico'))
        entrada_section.append(Spacer(1, 0.2*cm))
        add_info("Valor Estimado da Causa (Recebimento Cliente)",
dados_input.get('valor_estimado_causa_ganha', 0.0), formatar_moeda)
        add_info("Taxa Horária Base Mínima (p/ Cálculo Horário)",
dados_input.get('taxa_horaria_base_minima'), formatar_moeda)

        story.append(KeepTogether(entrada_section))

        # --- Detalhes do Cálculo da Taxa Horária (Modelo Horário) ---
        story.append(PageBreak())
        story.append(Paragraph("Cálculo da Taxa Horária Sugerida (Base
Horária)", style_subtitulo))
        detalhes_taxa = calculos.get('detalhes_taxa_horaria', {})
        taxa_section = []
```

```
        taxa_section.append(Paragraph("<u>Componentes do Cálculo:</u>",
style_destaque))
        add_info("Taxa Horária Base Informada", detalhes_taxa.get('Taxa
Horária Base Informada'), formatar_moeda)
        taxa_section.append(Spacer(1, 0.3*cm))
        taxa_section.append(Paragraph("Fatores Multiplicadores
Aplicados:", style_label_valor))
        fatores = detalhes_taxa.get('Fatores Multiplicadores', {})
        for nome, valor in fatores.items():
            # Indentar os fatores
            p = Paragraph(f"    - {nome}: {formatar_numero(valor, 3)}x",
style_label_valor)
            taxa_section.append(p)

        taxa_section.append(Spacer(1, 0.6*cm))
        taxa_horaria_final_fmt =
formatar_moeda(calculos.get('taxa_horaria_sugerida'))
        taxa_section.append(Paragraph(f"<b>Taxa Horária Sugerida
Calculada: {taxa_horaria_final_fmt}</b>", style_destaque))
        story.append(KeepTogether(taxa_section))

        # --- Cálculo do Preço Final (Modelo Horário) ---
        story.append(Paragraph("Cálculo do Preço Final (Base Horária)",
style_subtitulo))
        detalhes_preco_horario = calculos.get('detalhes_preco_horario',
{})
        preco_h_section = []
        preco_h_section.append(Paragraph("<u>Componentes do
Cálculo:</u>", style_destaque))
        add_info("Taxa Horária Utilizada",
detalhes_preco_horario.get('Taxa Horária Utilizada'), formatar_moeda)
        add_info("Horas Estimadas (Esforço)",
detalhes_preco_horario.get('Horas Estimadas'), lambda v:
formatar_numero(v, 1), unit=" h")
        add_info("Fator Complexidade",
f"{detalhes_preco_horario.get('Nível de Complexidade')}
({formatar_numero(detalhes_preco_horario.get('Fator
Complexidade'),2)}x)")
        add_info("Fator Urgência", f"{detalhes_preco_horario.get('Nível
de Urgência')} ({formatar_numero(detalhes_preco_horario.get('Fator
Urgência'),2)}x)")
        preco_h_section.append(Spacer(1, 0.3*cm))
        add_info("Preço Base (Taxa * Horas)",
detalhes_preco_horario.get('Preço Base (Taxa * Horas)'), formatar_moeda)

        preco_h_section.append(Spacer(1, 0.5*cm))
        preco_final_horario_fmt =
formatar_moeda(calculos.get('preco_horario_sugerido'))
```

```
        preco_h_section.append(Paragraph(f"Preço Final Sugerido (Base
Horária): {preco_final_horario_fmt}", style_final_preco))
        story.append(KeepTogether(preco_h_section))


        # --- Cálculo por Êxito (Se aplicável) ---
        detalhes_preco_exito = calculos.get('detalhes_preco_exito', {})
        preco_exito_sugerido = calculos.get('preco_exito_sugerido')

        if preco_exito_sugerido is not None and preco_exito_sugerido > 0:
            story.append(PageBreak())
            story.append(Paragraph("Cálculo do Preço (Base Êxito -
Estimativa)", style_subtitulo))
            preco_e_section = []
            preco_e_section.append(Paragraph("<u>Componentes do
Cálculo:</u>", style_destaque))

            valor_causa_fmt =
formatar_moeda(detalhes_preco_exito.get('Valor Estimado Causa Cliente'))
            percentual_aplicado = detalhes_preco_exito.get('Percentual
Êxito Aplicado', constants.PERCENTUAL_EXITO_PADRAO) # Pega o aplicado ou
o padrão
            percentual_fmt = f"{percentual_aplicado:.1%}"
            percentual_const_fmt =
f"{constants.PERCENTUAL_EXITO_PADRAO:.0%}" # Para mostrar o padrão usado

            add_info("Valor Estimado da Causa (Recebimento Cliente)",
valor_causa_fmt)
            add_info(f"Percentual de Êxito Aplicado
({percentual_const_fmt} padrão)", percentual_fmt)

            preco_e_section.append(Spacer(1, 0.5*cm))
            preco_final_exito_fmt = formatar_moeda(preco_exito_sugerido)
            preco_e_section.append(Paragraph(f"Preço Sugerido (Base
Êxito): {preco_final_exito_fmt}", style_final_preco))
            story.append(KeepTogether(preco_e_section))


        # --- Observações Finais ---
        # Adiciona espaço antes das observações, a menos que seja a
última coisa na página anterior
        if preco_exito_sugerido is None or preco_exito_sugerido <= 0:
             story.append(Spacer(1, 3*cm)) # Mais espaço se não houve
cálculo de êxito
        else:
             story.append(Spacer(1, 1*cm))

        story.append(Paragraph("Observações Importantes:", style_obs))
        obs_text = """
```

```
        - Os valores apresentados são <b>SUGESTÕES</b> calculadas com
base nos dados fornecidos e nos parâmetros definidos em
<i>core/constants.py</i>.
        - O modelo de 'Base Horária' reflete o esforço estimado (horas)
multiplicado por uma taxa horária valorizada pela experiência,
especialização e outros fatores.
        - O modelo de 'Base Êxito' (se aplicável) reflete um percentual
padrão sobre o ganho estimado do cliente, comum em certas áreas (ex:
previdenciária, trabalhista, cível).
        - A escolha final do modelo de cobrança (horário, êxito, misto,
fixo) e o valor dependem da análise de mercado, do tipo de serviço, do
valor percebido pelo cliente, do acordo contratual e da estratégia do
escritório.
        - A valoração de fatores intangíveis (experiência, sucesso) é
inerentemente subjetiva. Ajuste os pesos e parâmetros no código para
refletir sua realidade.
        - Este simulador é uma ferramenta de apoio à decisão e não
substitui o julgamento profissional, a análise de risco e a negociação
com o cliente.
        """
        story.append(Paragraph(obs_text, style_obs))

        # --- Construção do PDF ---
        doc.build(story)
        print(f"\nPDF gerado com sucesso: {nome_arquivo}")

    except ImportError as ie:
         print(f"\nErro de Importação ao gerar PDF: {ie}. Verifique as
dependências (reportlab) e a estrutura do projeto.")
         # Poderia lançar a exceção para ser pega pela GUI, ou retornar
False
         raise # Re-lança a exceção para a GUI tratar

    except Exception as e:
        print(f"\nErro inesperado ao gerar PDF: {type(e).__name__}: {e}")
        # Re-lança a exceção para a GUI tratar
        raise
```

Siga-me no LinkedIn: www.linkedin.com/comm/mynetwork/discovery-see-all?usecase=PEOPLE_FOLLOWS&followMember=izairton-oliveira-de-vasconcelos-a1916351

Minha Newsletter, o link para assinar:https://www.linkedin.com/build-relation/newsletter-follow?entityUrn=7287106727202742273

https://www.linkedin.com/pulse/do-zero-%25C3%25A0-interface-gr%25C3%25A1fica-criando-um-simulador-de-e-izairton-nkaof

https://github.com/IOVASCON/simulador_honorarios.git