INTEGRAÇÃO ESTRATÉGICA: PYTHON E SISTEMAS DE PRODUÇÃO NA ERA DA INDÚSTRIA 4.0

Italo Mayacone Martins Barbosa (UFCG-CDSA) *italomayacone00@gmail.com*
Cecir Barbosa De Almeida Farias (UFCG-CDSA) *cecir.almeida@gmail.com*

Resumo

A pesquisa propôs um programa em *Python* para simplificar a coleta de dados empresariais, focando nos desafios atuais dos sistemas de produção. Destacou-se a integração de múltiplas fontes de dados, a segurança, escalabilidade e a agilidade na entrega de informações vitais para a produção. Enfatizou-se o papel essencial do *Python*, sua versatilidade e a alta demanda no mercado, reconhecendo-o como um pilar na evolução dos sistemas produtivos contemporâneos. A metodologia em *Python* visa simplificar a coleta e análise de dados empresariais, seguindo uma abordagem estruturada desde a identificação das necessidades até a implementação do Prodanalytica, priorizando não só a funcionalidade do software, mas também sua relevância na otimização dos processos de negócio. Os resultados, demonstrados nos trechos do código e suas respectivas GUIs, apresentam as funcionalidades do programa criado, Prodanalytica, como janelas de login, cadastro, questionários interativos e manipulação de respostas. Essas interfaces proporcionaram um ambiente amigável, permitindo a coleta e edição de dados, exibição de resultados e geração de arquivos. O programa Prodanalytica desenvolvido, visa fornecer soluções inovadoras, como otimização de fluxo de produção, identificação de gargalos, previsão de demanda e sugestões para melhorias na eficiência operacional, criando soluções práticas para modernos sistemas de produção.

Palavras-Chaves: *(Python; Sistemas de Produção; Coleta de Dados)*

1. Introdução

A programação em *Python* desempenha um papel essencial nos sistemas de produção modernos, consolidando-se como uma linguagem amplamente reconhecida por sua versatilidade e popularidade. Com uma sintaxe simples e poderosa, linguagem de programação *Python* é adotada em diversas indústrias, incluindo manufatura, logística e automação. Estatísticas recentes evidenciam um crescimento contínuo na demanda por profissionais

qualificados em *Python*, refletindo a crescente necessidade de proficiência nesta linguagem dinâmica.

Essa ascendência é impulsionada pela capacidade singular do *Python* em facilitar o desenvolvimento de sistemas eficientes, rápidos e confiáveis, características cruciais para operações de produção em larga escala. Nos sistemas de produção modernos, deparamo-nos com desafios complexos, como a necessidade imperativa de coletar e analisar grandes volumes de dados para otimização do desempenho, redução de custos e aprimoramento da qualidade dos produtos.

A programação em *Python* destaca-se nesse cenário, oferecendo ferramentas poderosas para a manipulação e análise de dados em tempo real. A linguagem sobressai-se pela sua notável capacidade de integração com sistemas existentes e dispositivos de coleta de dados, emergindo como a escolha natural para empresas que buscam aprimorar seus processos de produção (Manfrin Prado, 2021).

O objetivo deste projeto é desenvolver um programa em *Python* dedicado a simplificar e otimizar o processo de coleta de dados em empresas. Este programa foi meticulosamente elaborado para extrair dados de diversas fontes, como sensores industriais, sistemas de automação e bancos de dados, consolidando-os em um formato acessível e propício à análise detalhada. Isso permitirá uma análise aprofundada dos padrões de produção, eficiência operacional e desempenho, capacitando as empresas a tomar decisões estratégicas baseadas em informações precisas e atualizadas. A principal intenção é facilitar a extração de dados relevantes de múltiplas origens para oferecer uma visão abrangente do funcionamento operacional, permitindo que as empresas obtenham *insights* valiosos para melhorar processos e estratégias.

Adicionalmente, o programa emprega algoritmos avançados para identificar padrões, tendências e anomalias nos dados, proporcionando *insights* valiosos para a tomada de decisões estratégicas. Em síntese, este trabalho busca unir a potência da programação em *Python* aos desafios inerentes aos sistemas de produção modernos, criando uma solução inovadora e eficiente para a coleta e análise de dados.

Ao fazê-lo, almeja-se contribuir para a melhoria contínua dos processos industriais, impulsionando a eficiência, reduzindo custos e fomentando a inovação nas empresas que abraçam essa abordagem integrada. A sinergia entre a robustez do *Python* e a complexidade

dos ambientes de produção modernos cria uma oportunidade única para impulsionar avanços significativos na eficiência operacional e na qualidade dos produtos (Pinto, 2023).

2. Referencial teórico

2.1 Indústria 4.0

Ao longo da história as Revoluções Industriais (RIs) vem acontecendo mudanças em todo o ecossistema, mudando a forma em que as cadeias produtivas se comportam e marcando momentos de transição da humanidade (Huberman, 1986). Com o início da primeira RI, que ocorreu entre os anos de 1760 e 1840, foi possível notar o surgimento de novas fábricas e a utilização de máquinas a vapor, trocando o trabalho manual por mecânicos. Iniciando uma mudança do modo de produção das novas indústrias (Schwab, 2017).

No final do século XIX, teve início a Segunda Revolução Industrial, caracterizada pela introdução da eletricidade, que trouxe inovações nos sistemas de produção e aumentou a produtividade das fábricas. Além disso, essa revolução provocou mudanças culturais e comportamentais significativas na sociedade. A Terceira Revolução Industrial teve início na década de 1960 e marcou um ponto crucial para as gerações futuras com a introdução dos computadores nas décadas de 1970 e 1980, seguidos pelo surgimento da internet na década de 1990. Essas inovações possibilitaram avanços científicos significativos (Schwab, 2017).

Com a chegada da quarta RI ou Indústria 4.0, foi possível obter um avanço nas tecnologias e ferramentas de processamento de dados das empresas, visto que a competição por qualidade e produtividade dentro do mercado cresceu. A busca por melhorias no desempenho da produção, foi fundamental para que as indústrias otimizassem sua produção evitando desperdícios (Arbegaus, 2018).

2.2 Programação

Segundo Souza et al. (2021), a programação não se limita ao domínio de linguagens específicas, mas também inclui a compreensão de princípios algorítmicos, lógica de programação e

estruturas de dados que são essenciais para o desenvolvimento de soluções eficazes e eficientes. A programação é uma habilidade essencial no cenário tecnológico atual e desempenha um papel vital no desenvolvimento de softwares, aplicativos e sistemas que abrangem todas as esferas da sociedade globalizada. A capacidade de escrever código não é apenas uma habilidade técnica, mas também uma forma de pensamento crítico, onde se busca solucionar problemas dentro de um ambiente (Kahn, 2017).

Desse modo, a importância da programação na vida das pessoas vai além do âmbito profissional, acaba por influenciar a maneira com que os indivíduos interagem com as tecnologias no seu cotidiano. Estudos mostram que habilidades de programação são cruciais para promover a alfabetização digital e a participação ativa na sociedade digital (Prensky, 2019). Além do mais, a programação promove o pensamento computacional, que envolve a capacidade de pensar de forma estruturada e lógica, uma habilidade essencial em uma era dominada pela informação digital (Conforto, 2018).

2.3 Python

Python é uma linguagem de programação interpretada versátil, projetada para ser acessível e fácil de entender. Ao contrário de linguagens como *R* ou *Matlab*, Python é valorizado por sua legibilidade de código, tornando-o uma escolha popular para iniciantes. Sua natureza dinâmica permite o desenvolvimento iterativo e a prototipagem rápida, ao mesmo tempo que oferece suporte robusto para a construção de aplicativos complexos. *Python* também é amplamente reconhecida como a linguagem fundamental para aprendizado de máquina e ciência de dados devido à sua extensa biblioteca de ferramentas. Graças a esta qualidade, rapidamente se consolidou como a plataforma de eleição dos profissionais destas áreas, consolidando o seu estatuto como uma das línguas mais dominantes nestas áreas (Lopes et al. 2019).

Como apontado Formigoni (2021), *Python* é amplamente preferida em áreas como ciência de dados, aprendizado de máquina e inteligência artificial. A razão para esta preferência reside na abundância de bibliotecas especializadas como *numpy*, *Pandas* e *tensorflow*, que tornam o *Python* uma escolha ideal para analisar dados e criar algoritmos sofisticados. Um ponto forte notável da linguagem *Python* é sua comunidade ativa e colaborativa. Por meio de fóruns online,

iniciativas de código aberto e eventos como conferências e *hackathons*, os desenvolvedores *Python* estão constantemente inovando e aprimorando a linguagem (Araujo, 2018).

2.4 Sistemas de Produção

Os sistemas de manufatura têm sido objeto de diversas teorias de estudo ao longo do tempo. Uma abordagem proeminente é a teoria enxuta desenvolvida por Taiichi Ohno, que se baseia na minimização de desperdícios e na busca constante de eficiência. Segundo Ohno, "O principal objetivo de um sistema de produção é eliminar todos os tipos de desperdícios" (Ohno, 1988).

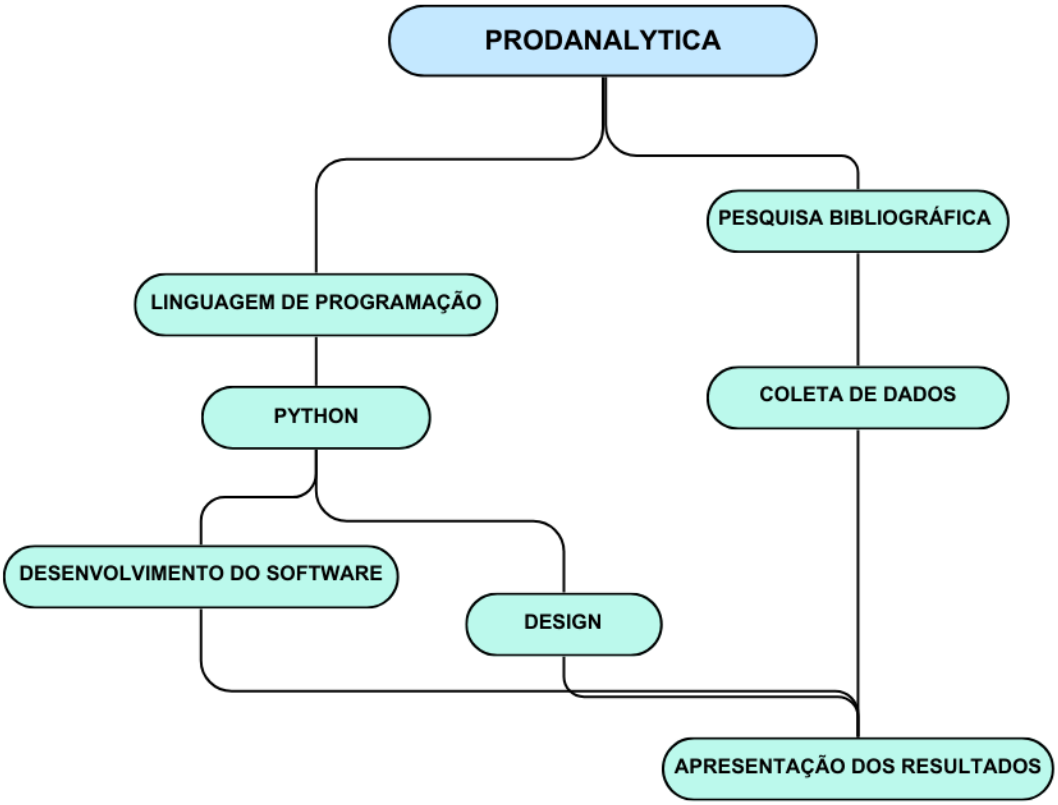
Dentro do campo dos sistemas de produção, uma abordagem contemporânea crucial é a Indústria 4.0. Esta teoria enfoca a integração de tecnologias digitais, como Internet das Coisas (IoT), inteligência artificial, big data e automação, para criar sistemas de produção altamente eficientes e adaptativos (Kagermann et al., 2013).

Outra perspectiva relevante é a abordagem do Sistema Toyota de Produção (TPS), também conhecido como Lean Manufacturing. Descrita por Jeffrey Liker, esta abordagem enfatiza a importância da melhoria contínua, do envolvimento dos funcionários e da eliminação de desperdícios para alcançar a máxima eficiência (Liker, 2004).

3. Metodologia

No presente trabalho é adotada uma metodologia de criação de software em *Python*, que visa otimizar e simplificar a coleta e análise de dados em ambientes empresariais, enfrentando os desafios inerentes aos sistemas de produção modernos. Esta metodologia está estruturada em fases que vão desde a identificação das necessidades e objetivos do programa até à implementação, teste e avaliação da sua eficácia. O desenvolvimento do programa Prodanalytica segue uma abordagem sistemática que inclui conhecimentos teóricos sobre as revoluções industriais, os sistemas de produção, a importância da programação e as especificidades da linguagem Python. Este enquadramento metodológico visa não só criar software funcional, mas também garantir a sua relevância e eficácia na melhoria dos processos de negócio.

Figura 1 – Fluxograma dos processos



Fonte: Autores (2023)

O fluxograma adotado nesta metodologia de desenvolvimento de software Prodanalytica representa uma estrutura sequencial de etapas interligadas, refletindo uma abordagem sistemática e organizada para resolver problemas de sistemas de produção. Cada etapa do fluxograma corresponde a uma etapa chave do processo de desenvolvimento, começando com a identificação de necessidades e objetivos e terminando com considerações finais e sugestões para futuras iterações. Esta estrutura de etapas fornece orientação clara e permite uma progressão lógica e ordenada durante o desenvolvimento de software.

A articulação entre as etapas do fluxograma reflete o caráter integrado da metodologia adotada. Cada fase está intimamente ligada às anteriores e subsequentes e forma um ciclo contínuo de planeamento, implementação, verificação e análise. Isto apoia um processo iterativo que permite ajustes e refinamentos ao longo do caminho, garantindo que o software Prodanalytica seja desenvolvido de forma eficiente e responda às necessidades identificadas no início do processo.

O fluxograma também enfatiza a importância do panorama teórico e da pesquisa bibliográfica como base para o desenvolvimento de software. Esta fase inicial fornece a base para decisões subsequentes e garante que a Prodanalytica se estruture com base no conhecimento consolidado das revoluções industriais, dos sistemas de produção e da relevância da linguagem *Python*. Essa abordagem contribui para a construção de um software sólido e alinhado às exigências do ambiente de negócios atual.

4. Resultados e Discussão

Com o propósito de facilitar a coleta de dados, foi desenvolvido um framework com o propósito específico de simplificar a coleta de dados para análise do sistema produtivo. Este framework foi projetado com o objetivo de oferecer gratuitamente um software que facilita e agiliza o processo de coleta de dados da empresa. A plataforma desenvolvida permite que os usuários preencham o questionário criado, de forma eficiente, proporcionando uma experiência amigável. Isso é alcançado através de um questionário intuitivo, desenvolvido para ser simples e fácil de utilizar.

Figura 2 – Interface Gráfica com Tkinter e PIL

File Edit Format Run Options Window Help

```
import tkinter as tk
from tkinter import messagebox
from PIL import Image, ImageTk
from tkinter import scrolledtext
from tkinter import ttk
from tkinter import filedialog
```

Fonte: Autores (2023)

O código criado com a linguagem *Python* na Figura 2 utiliza a biblioteca *Tkinter* para construir uma interface gráfica de usuário (GUI). Ele importa módulos como *tk* para a GUI básica, *messagebox* para caixas de diálogo, *image* e *imagetk* (do *Pillow*) para manipulação de imagens, *scrolledtext* para criar áreas de texto com rolagem, e *ttk* para widgets temáticos. O *filedialog* é importado para a seleção de arquivos. O código oferece funcionalidades para criar janelas, exibir mensagens, manipular imagens, criar áreas de texto roláveis e abrir caixas de diálogo para seleção de arquivos, comumente utilizadas em interfaces gráficas com *Python.*

Figura 3 – Interface de Login com Imagem de Fundo

File Edit Format Run Options Window Help

```
# Crie listas para armazenar os usuários e senhas cadastrados
usuarios_cadastrados = [""]
senhas_cadastradas = [""]

class InterfacePerguntas:
    def __init__(self, janela):
        self.janela = janela
        self.janela.title("Login")
        self.janela.geometry("800x600")  # Define o tamanho da janela

        # Carrega a imagem de fundo
        caminho_imagem = r"[redacted]"
        imagem_fundo = Image.open(caminho_imagem)
        imagem_fundo = imagem_fundo.resize((800, 600), Image.ANTIALIAS if hasattr(Image, 'ANTIALIAS') else 3)
        self.imagem_fundo = ImageTk.PhotoImage(imagem_fundo)

        # Crie um canvas para a imagem de fundo
        self.canvas = tk.Canvas(janela, width=800, height=600)
        self.canvas.pack(fill=tk.BOTH, expand=True)
        self.canvas.create_image(0, 0, anchor=tk.NW, image=self.imagem_fundo)

        # Rótulo e caixa de entrada para o nome de usuário
        self.label_usuario = tk.Label(self.canvas, text="Nome de Usuário:", fg="black", font=("Helvetica", 20))
        self.label_usuario.grid(row=0, column=0, padx=10, pady=10, sticky=tk.E)
        self.entry_usuario = tk.Entry(self.canvas, width=30, font=("Helvetica", 14))
        self.entry_usuario.grid(row=0, column=1, padx=10, pady=10)
        self.entry_usuario.bind("<Return>", self.mudar_para_senha)

        # Rótulo e caixa de entrada para a senha
        self.label_senha = tk.Label(self.canvas, text="Senha:", fg="black", font=("Helvetica", 20))
        self.label_senha.grid(row=1, column=0, padx=10, pady=10, sticky=tk.E)
        self.entry_senha = tk.Entry(self.canvas, show="*", width=30, font=("Helvetica", 14))
        self.entry_senha.grid(row=1, column=1, padx=10, pady=10)
        self.entry_senha.bind("<Return>", self.verificar_login)
```

Fonte: Autores (2023)

A imagem da Figura 3 mostra uma parte do código criado em *Python* que utiliza a biblioteca *Tkinter* para criar uma interface gráfica de login. A janela resultante é configurada com rótulos, caixas de entrada para nome de usuário e senha, e um *canvas* exibindo uma imagem de fundo. Essa interface oferece uma interação amigável, permitindo que o usuário pressione "Enter" para facilitar o processo de login. A imagem de fundo é carregada e redimensionada, e os dados inseridos são posteriormente utilizados para verificar o login. Em resumo, o código proporciona uma estrutura inicial para o desenvolvimento de um sistema visual de autenticação em *Python*.

Figura 4 – Interface de Login e Cadastro com Tkinter

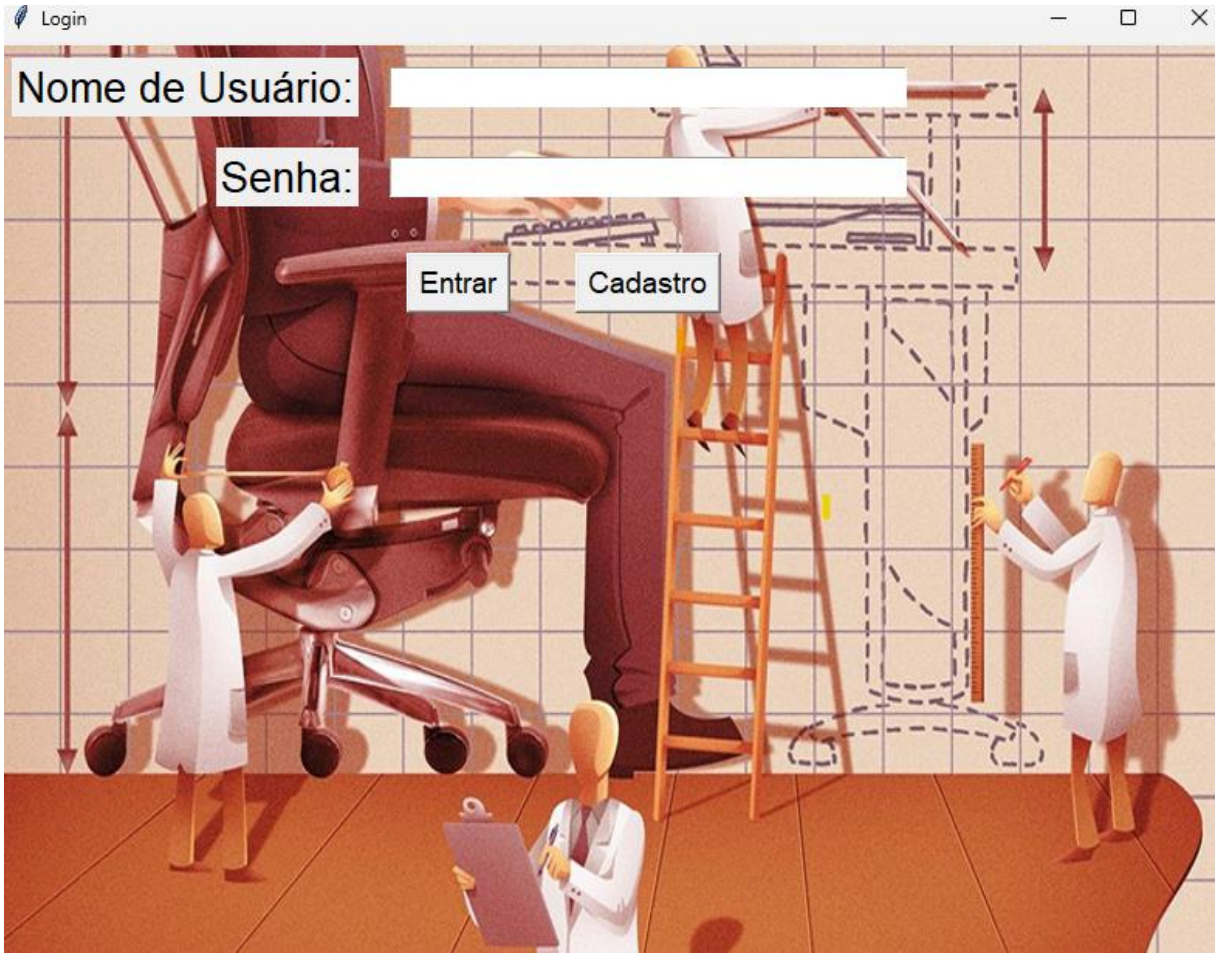
File Edit Format Run Options Window Help

```
        # Botão de entrar
        self.botao_entrar = tk.Button(self.canvas, text="Entrar", command=self.verificar_login, font=("Helvetica", 14))
        self.botao_entrar.grid(row=2, column=0, columnspan=2, pady=20)

        # Botão de cadastro
        self.botao_cadastro = tk.Button(self.canvas, text="Cadastro", command=self.abrir_janela_cadastro, font=("Helvetica", 14))
        self.botao_cadastro.grid(row=2, column=1, columnspan=2, pady=10)

    def verificar_login(self, event=None):  # Adicione event=None como argumento padrão
        usuario = self.entry_usuario.get().strip()  # Remover espaços em branco no início e no final
        senha = self.entry_senha.get().strip()  # Remover espaços em branco no início e no final

        # Verifique as credenciais
        if usuario in usuarios_cadastrados and senhas_cadastradas[usuarios_cadastrados.index(usuario)] == senha:
            messagebox.showinfo("Login", "Login bem-sucedido!")
            self.limpar_campos()
            self.abrir_janela_perguntas(usuario)  # Chame o método correto
        else:
            messagebox.showerror("Login", "Nome de usuário ou senha incorretos.")

    def mudar_para_senha(self, event):
        self.entry_senha.focus_set()

    def limpar_campos(self):
        # Limpa os campos de entrada
        self.entry_usuario.delete(0, tk.END)
        self.entry_senha.delete(0, tk.END)

    def abrir_janela_perguntas(self, usuario):
        janela_perguntas = tk.Toplevel(self.janela)  # Cria uma nova janela de perguntas
        app_perguntas = JanelaPerguntas(janela_perguntas, usuario)  # Instancia a classe da janela de perguntas

    def abrir_janela_cadastro(self):
        janela_cadastro = tk.Toplevel(self.janela)  # Cria uma nova janela de cadastro
        app_cadastro = JanelaCadastro(janela_cadastro)  # Instancia a classe da janela de cadastro
```

Fonte: Autores (2023)

O código da Figura 4 apresenta uma interface de login usando *Tkinter*, com botões para entrar e se cadastrar. Ao inserir nome de usuário e senha e pressionar "Entrar", verifica se as credenciais correspondem aos registros. Em caso de sucesso, abre uma janela de perguntas; caso contrário, exibe uma mensagem de erro. O botão "Cadastro" abre uma janela para novos registros, usando métodos para limpar campos, mudar para a caixa de senha e verificar o login, proporcionando uma interação fluída ao usuário.

Figura 5 – Janela de Cadastro de exibição



Fonte: Autores (2023)

A imagem exibe uma interface gráfica intuitiva, simplificando a experiência dos usuários no programa apresentado.

Figura 6 – Janela de Cadastro com Tkinter

File Edit Format Run Options Window Help

```
class JanelaCadastro:
    def __init__(self, janela_cadastro):
        self.janela_cadastro = janela_cadastro
        self.janela_cadastro.title("Cadastro")

        # Crie uma tela azul usando um widget de frame
        largura = 600
        altura = 400
        self.janela_cadastro.geometry(f"{largura}x{altura}")

        # Carregue a imagem de fundo do cadastro
        caminho_imagem_cadastro = r"[redacted]"
        imagem_fundo_cadastro = Image.open(caminho_imagem_cadastro)
        imagem_fundo_cadastro = imagem_fundo_cadastro.resize((largura, altura), Image.ANTIALIAS if hasattr(Image, 'ANTIALIAS') else 3)
        imagem_fundo_cadastro_tk = ImageTk.PhotoImage(imagem_fundo_cadastro)

        # Use um Label para exibir a imagem de fundo
        self.label_fundo = tk.Label(self.janela_cadastro, image=imagem_fundo_cadastro_tk)
        self.label_fundo.image = imagem_fundo_cadastro_tk
        self.label_fundo.place(x=0, y=0, relwidth=1, relheight=1)  # Estique para preencher toda a janela

        # Adicione elementos para a janela de cadastro aqui
        self.label_cadastro = tk.Label(self.janela_cadastro, text="Formulário de Cadastro", font=("Helvetica", 20))
        self.label_cadastro.pack(pady=10)

        # Rótulo e caixa de entrada para o nome de usuário para o cadastro
        self.label_usuario = tk.Label(self.janela_cadastro, text="Nome de Usuário:", font=("Helvetica", 14))
        self.label_usuario.pack(pady=10)
        self.entry_usuario = tk.Entry(self.janela_cadastro, width=30, font=("Helvetica", 14))
        self.entry_usuario.pack(pady=10)

        # Rótulo e caixa de entrada para a senha para o cadastro
        self.label_senha = tk.Label(self.janela_cadastro, text="Senha:", font=("Helvetica", 14))
        self.label_senha.pack(pady=10)
        self.entry_senha = tk.Entry(self.janela_cadastro, show="*", width=30, font=("Helvetica", 14))
        self.entry_senha.pack(pady=10)
```

Fonte: Autores (2023)

O código em *Python* da Figura 5 define uma classe chamada "janelacadastro" que cria uma interface gráfica de cadastro utilizando a biblioteca *Tkinter*. A janela apresenta um formulário de cadastro com rótulos e caixas de entrada para inserir um nome de usuário e senha. Utilizando a imagem de fundo carregada e redimensionada como plano de fundo, os elementos da interface são organizados de forma simples e intuitiva para o usuário. A interface fornece campos para inserir as informações necessárias para o registro de novos usuários, apresentando-os de maneira clara na janela criada.

Figura 6 – Verificação e Registro de Cadastro

File Edit Format Run Options Window Help

```
        # Botão do fim do cadastro
        self.botao_do_cadastro = tk.Button(self.janela_cadastro, text="Concluir Cadastro", command=self.verificar_cadastro, font=("Helvetica", 14))
        self.botao_do_cadastro.pack(pady=20)

    def verificar_cadastro(self):
        # Obtenha os dados do cadastro
        novo_usuario = self.entry_usuario.get().strip()
        nova_senha = self.entry_senha.get().strip()

        # Verifique se o usuário já existe
        if novo_usuario in usuarios_cadastrados:
            messagebox.showerror("Cadastro", "Nome de usuário já existe. Escolha outro nome de usuário.")
        # Verifique se a senha é igual ao nome de usuário
        elif novo_usuario == nova_senha:
            messagebox.showerror("Cadastro", "A senha não pode ser igual ao nome de usuário.")
        else:
            # Adicione o novo usuário e senha às listas
            usuarios_cadastrados.append(novo_usuario)
            senhas_cadastradas.append(nova_senha)
            messagebox.showinfo("Cadastro", "Cadastro concluído com sucesso!")
            self.janela_cadastro.destroy()  # Fecha a janela de cadastro
```

Fonte: Autores (2023)

O trecho de código em *Python* da Figura 6, parte de uma classe de interface gráfica, verifica e registra novos usuários. Ao clicar em um botão de confirmação, ele coleta os dados inseridos nos campos de nome de usuário e senha. Realiza verificações para evitar duplicatas ou casos em que a senha seja igual ao nome de usuário, exibindo mensagens de erro apropriadas nesses casos. Se os dados estiverem corretos, adiciona o novo usuário e senha às listas e exibe uma mensagem de sucesso, encerrando a janela de cadastro. Este código assegura a integridade dos dados e oferece um feedback claro ao usuário durante o processo de cadastro.

Figura 7 – Janela de Perguntas Empresariais

File Edit Format Run Options Window Help

```
class JanelaPerguntas(tk.Toplevel):

    def __init__(self, janela_perguntas, usuario):
        self.janelaperguntas = janela_perguntas
        self.janelaperguntas.title("Perguntas")
        self.perguntas = [
            "1-Qual é o nome da sua empresa?",
            "2-Qual é o setor de atuação da empresa?",
            "3-Há quanto tempo a empresa está no mercado?",
            "4-Quantos funcionários a empresa possui?",
            "5-Quais são os principais produtos ou serviços oferecidos pela empresa?",
            "6-Quem são seus clientes-alvo ou mercado-alvo?",
            "7-Existem parcerias estratégicas ou colaborações significativas que a empresa possui?",
            "8-A empresa recebeu algum reconhecimento ou prêmio por suas realizações no setor?",
            "9-Existem valores ou princípios fundamentais que a empresa segue?",
            "10-Quais são os principais desafios que sua empresa enfrenta atualmente?",
            "11-Quais são as metas de crescimento ou objetivos futuros da empresa?",
            "12-Qual é o processo principal de produção ou prestação de serviços da sua empresa?",
            "13-Como a empresa garante a eficiência e qualidade no processo de produção ou prestação de serviços?",
            "14-Como a empresa gerencia sua cadeia de suprimentos para garantir o fornecimento adequado de matéria-prima ou recursos necessários?",
            "15-Quais são os padrões de controle de qualidade adotados pela empresa para seus produtos ou serviços?",
            "16-Como a empresa lida com produtos ou serviços que não atendem aos padrões de qualidade?",
            "17-Como a empresa promove a inovação no processo produtivo ou na prestação de serviços?",
            "18-Existem programas de melhoria contínua implementados na empresa?",
            "19-A empresa utiliza tecnologias avançadas ou automação no processo de produção ou prestação de serviços?",
            "20-Como a tecnologia contribui para a eficiência operacional da empresa?",
            "21-Como a empresa gerencia seu estoque de produtos acabados, se aplicável?",
            "22-Existem estratégias para minimizar o excesso de estoque ou a falta de produtos disponíveis?",
            "23-Qual é o processo de logística e distribuição dos produtos ou serviços da empresa?",
            "24-Como a empresa garante a entrega oportuna aos clientes?",
            "25-Como a empresa determina sua capacidade produtiva em relação à demanda do mercado?",
            "26-Existem estratégias para lidar com picos inesperados na demanda?",
            "27-Quais são as práticas da empresa em relação à sustentabilidade e responsabilidade ambiental no processo produtivo?",
            "28-Existem iniciativas para reduzir o impacto ambiental da produção?",
            "29-Como a empresa treina e desenvolve seus funcionários para garantir que estejam atualizados com as práticas de produção ou prestação de serviços?",
            "30-Existem programas de capacitação para melhorar as habilidades da equipe?",
            "31-Como a empresa controla seus custos operacionais para manter a eficiência?",
            "32-Existem estratégias para reduzir desperdícios e otimizar recursos?",
            "33-A empresa possui integração vertical, produzindo seus próprios insumos ou realizando serviços relacionados?",
            "34-Pode descrever em detalhes como é o sistema produtivo da empresa?",
```

Fonte: Autores (2023)

A classe "janelaperguntas" do código da Figura 7 é uma janela secundária que apresenta uma série de 34 perguntas estratégicas relacionadas a uma empresa. Essas perguntas abrangem desde a identificação da empresa até questões específicas sobre produção, distribuição, sustentabilidade e gestão de recursos. Ao receber o nome de usuário como parâmetro, a janela é inicializada com o título "Perguntas" e exibe todas as questões no formato de lista. Esse código cria uma interface para coletar informações detalhadas sobre diversos aspectos de uma empresa, facilitando a análise e o entendimento de seu funcionamento.

Figura 8 – Continuação da Janela de Perguntas Empresariais

File Edit Format Run Options Window Help

```
            "34-Pode descrever em detalhes como é o sistema produtivo da empresa?",
            "35-Quais são os principais passos ou etapas envolvidos na produção de seus produtos ou prestação de serviços?",
            "36-Como são definidos e documentados os processos de produção ou prestação de serviços?",
            "37-Qual é o fluxo de trabalho típico desde a concepção do produto/serviço até a entrega final ao cliente?",
            "38-Existem pontos no processo produtivo onde há atrasos ou interrupções frequentes?",
            "39-Quais são as áreas ou processos que costumam enfrentar problemas de eficiência ou produtividade?",
            "40-Que sugestões você teria para melhorar a eficiência do sistema produtivo?",
            "41-Existem áreas específicas que você acredita que podem ser aprimoradas para aumentar a produtividade?"
        ]
        self.indice_pergunta_atual = 0
        self.respostas = []
        self.janelaperguntas.geometry("800x600")  # Define a largura para 800 pixels e a altura para 600 pixels
        self.usuario = usuario
        self.resultados_janela = None  # Referência para a janela de resultados
        self.scrollbar = None

        self.criar_interface()
```

Fonte: Autores (2023)

No trecho da Figura 8, a classe "janelaperguntas" continua a estrutura de perguntas iniciada anteriormente, agora com as questões de 35 a 41. Ela define variáveis para controlar o índice da pergunta atual, as respostas fornecidas pelo usuário e o tamanho da janela. Além disso, cria referências para a janela de resultados e uma barra de rolagem, preparando a interface para a inserção e manipulação das respostas do usuário. Este código estende a interface, incluindo mais questões e definindo parâmetros necessários para a interação e armazenamento das respostas.

Figura 9 – Interface de Perguntas Interativas

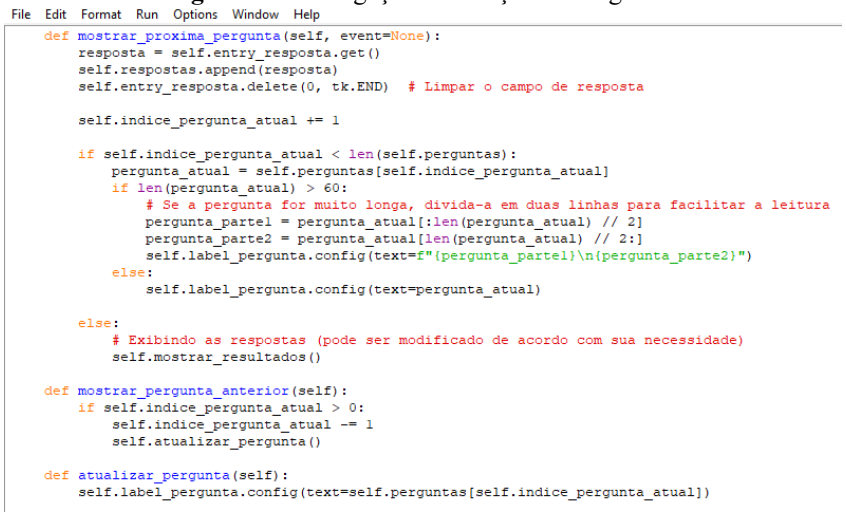
File Edit Format Run Options Window Help

```
    def criar_interface(self):
        self.label_pergunta = tk.Label(self.janelaperguntas, text=self.perguntas[self.indice_pergunta_atual], font=("Helvetica", 14))
        self.label_pergunta.pack()

        self.entry_resposta = tk.Entry(self.janelaperguntas, font=("Helvetica", 14))
        self.entry_resposta.pack()
        self.entry_resposta.bind("<Return>", self.mostrar_proxima_pergunta)

        self.botao_anterior = tk.Button(self.janelaperguntas, text="Pergunta Anterior", command=self.mostrar_pergunta_anterior, font=("Helvetica", 12))
        self.botao_anterior.pack()

        self.botao_proximo = tk.Button(self.janelaperguntas, text="Próxima Pergunta", command=self.mostrar_proxima_pergunta, font=("Helvetica", 12))
        self.botao_proximo.pack()

        self.botao_arquivo = tk.Button(self.janelaperguntas, text="Alterar Resposta", command=self.abrir_janela_alterar_resposta, font=("Helvetica", 12))
        self.botao_arquivo.pack()

        self.botao_arquivo = tk.Button(self.janelaperguntas, text="Mostrar Resultados", command=self.mostrar_resultados, font=("Helvetica", 12))
        self.botao_arquivo.pack()

        self.botao_gerar_arquivo = tk.Button(self.janelaperguntas, text="Gerar Arquivo de Resultados", command=self.gerar_arquivo_resultados, font=("Helvetica", 12))
        self.botao_gerar_arquivo.pack()
```

Fonte: Autores (2023)

O método "criar_interface" Figura 9, configura a interface para a janela de perguntas empresariais. Ele cria rótulos para exibir as perguntas e caixas de entrada para inserir as respostas do usuário. Além disso, implementa botões para navegar entre as perguntas, alterar respostas, exibir resultados e gerar arquivos com os resultados obtidos. Esses elementos proporcionam uma interação dinâmica, permitindo ao usuário inserir respostas, navegar entre as questões e acessar as funcionalidades oferecidas pela janela de perguntas.

Figura X – Navegação e Exibição de Perguntas



Fonte: Autores (2023)

Essas funções, "mostrar_proxima_pergunta" e "mostrar_pergunta_anterior", controlam a exibição das perguntas na interface. Ao avançar para a próxima pergunta, elas coletam a resposta do usuário, armazenam-na e atualizam a interface para exibir a próxima pergunta. Caso a pergunta seja muito longa, ela é dividida para facilitar a leitura. A função "mostrar_resultados" é chamada quando todas as perguntas foram respondidas, exibindo ou realizando ações com as respostas coletadas. A função "mostrar_pergunta_anterior" permite retroceder para a pergunta anterior, atualizando a interface para exibir a pergunta correspondente. Essas funções oferecem controle e exibição dinâmica das perguntas, proporcionando uma experiência interativa ao usuário durante o processo de resposta.

Figura X – Exibição de Resultados em Scrollbar

```
File  Edit  Format  Run  Options  Window  Help
    def mostrar_resultados(self):
        self.resultados_janela = tk.Toplevel(self.janelaperguntas)
        self.resultados_janela.title("Resultados")

        # Adicionando scrollbar
        self.scrollbar = tk.Scrollbar(self.resultados_janela)
        self.scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

        # Adicionando um widget Text para mostrar os resultados
        self.text_resultados = tk.Text(self.resultados_janela, wrap=tk.WORD, yscrollcommand=self.scrollbar.set, font=("Helvetica", 14))
        self.text_resultados.pack(expand=True, fill=tk.BOTH)

        # Configurando a scrollbar
        self.scrollbar.config(command=self.text_resultados.yview)

        # Adicionando os resultados ao widget Text
        for pergunta, resposta in zip(self.perguntas, self.respostas):
            resultado_texto = f"{pergunta}\nResposta: {resposta}\n{'-'*30}\n"
            self.text_resultados.insert(tk.END, resultado_texto)

        self.text_resultados.config(state=tk.DISABLED)  # Torna o widget Text somente leitura
```

Fonte: Autores (2023)

Essa função, "mostrar_resultados", cria uma nova janela para apresentar os resultados das perguntas respondidas. Dentro da janela, é adicionado um widget de texto que exibe as perguntas e suas respectivas respostas. Além disso, inclui uma barra de rolagem para facilitar a navegação nos resultados. O texto é formatado para exibir cada pergunta seguida de sua resposta, separadas por um marcador visual. Por fim, o widget de texto é configurado como somente leitura para garantir a visualização dos resultados sem a possibilidade de edição. Essa função proporciona uma visualização organizada e acessível dos dados coletados durante o processo de perguntas e respostas.

Figura X – Seleção e Modificação de Respostas

```
File  Edit  Format  Run  Options  Window  Help
    def abrir_janela_alterar_resposta(self):
        # Verificar se há respostas para modificar
        if not self.respostas:
            # Se não houver respostas, mostrar uma caixa de mensagem informativa
            tk.messagebox.showinfo("Aviso", "Não há respostas para modificar.")
            return

        self.janela_selecao_resposta = tk.Toplevel(self.janelaperguntas)
        self.janela_selecao_resposta.title("Selecionar Resposta para Alterar")
        self.janela_selecao_resposta.geometry("400x500")  # Defina as dimensões da janela

        # Rótulo explicativo
        label_instrucao = tk.Label(self.janela_selecao_resposta, text="Selecione uma resposta para modificar:", font=("Helvetica", 14))
        label_instrucao.pack(pady=10)

        # Lista de respostas
        self.lista_respostas = tk.Listbox(self.janela_selecao_resposta, font=("Helvetica", 14))
        self.lista_respostas.pack(expand=True, fill=tk.BOTH)

        # Preencher a lista de respostas com as respostas existentes
        for resposta in self.respostas:
            self.lista_respostas.insert(tk.END, resposta)

        # Botão para modificar resposta
        botao_modificar = tk.Button(self.janela_selecao_resposta, text="Modificar Resposta", command=self.modificar_resposta_selecionada,
                                    font=("Helvetica", 12), bg="#4CAF50", fg="white")
        botao_modificar.pack(pady=10)

        # Estilo da lista de respostas
        style = ttk.Style()
        style.configure("TListbox", font=("Helvetica", 14))

        # Cor de fundo da janela
        self.janela_selecao_resposta.configure(bg="#f2f2f2")
```

Fonte: Autores (2023)

A função "abrir_janela_alterar_resposta" abre uma nova janela que permite ao usuário selecionar e modificar uma resposta anteriormente inserida. Caso não haja respostas, exibe uma mensagem informativa. Na janela de seleção de resposta, são apresentadas as respostas em uma

lista. O usuário pode selecionar uma resposta específica e clicar no botão "Modificar Resposta" para realizar a alteração. O estilo da lista é configurado para melhor legibilidade, e o fundo da janela é definido para uma cor específica. Essa função oferece uma maneira simples e direta de editar respostas anteriores durante o processo de preenchimento das informações.

Figura X – Modificação de Respostas Selecionadas

```
File  Edit  Format  Run  Options  Window  Help
    def modificar_resposta_selecionada(self):
        # Obter o índice da resposta selecionada na lista
        indice_selecionado = self.lista_respostas.curselection()
        if indice_selecionado:
            indice_selecionado = indice_selecionado[0]
            # Criar uma nova janela para modificar a resposta selecionada
            janela_modificar_resposta = tk.Toplevel(self.janela_selecao_resposta)
            janela_modificar_resposta.title("Modificar Resposta")
            janela_modificar_resposta.geometry("400x200")  # Define as dimensões da janela

            # Adicionar um rótulo explicativo
            label_explicativo = tk.Label(janela_modificar_resposta, text="Digite a nova resposta abaixo:", font=("Helvetica", 14))
            label_explicativo.pack(pady=10)  # Adiciona algum espaço vertical

            # Adicionar uma caixa de entrada para a nova resposta
            nova_resposta_entry = tk.Entry(janela_modificar_resposta, font=("Helvetica", 14), width=40)
            nova_resposta_entry.pack(pady=10)  # Adiciona algum espaço vertical

            # Adicionar um botão para salvar a resposta modificada
            botao_salvar = tk.Button(janela_modificar_resposta, text="Salvar", command=lambda:
                                     self.salvar_resposta_modificada(indice_selecionado, nova_resposta_entry.get(), janela_modificar_resposta), font=("Helvetica", 12))
            botao_salvar.pack()

    def salvar_resposta_modificada(self, indice, nova_resposta, janela):
        # Modificar a resposta na lista de respostas
        self.respostas[indice] = nova_resposta
        # Fechar a janela de modificação de resposta
        janela.destroy()
        # Atualizar a lista de respostas na janela principal
        self.atualizar_lista_respostas()

    def atualizar_lista_respostas(self):
        # Limpar a lista de respostas
        self.lista_respostas.delete(0, tk.END)
        # Preencher a lista com as respostas atualizadas
        for resposta in self.respostas:
            self.lista_respostas.insert(tk.END, resposta)
```

Fonte: Autores (2023)

A função "modificar_resposta_selecionada" permite ao usuário escolher uma resposta na lista de respostas exibidas na janela de seleção. Ao selecionar uma resposta, é aberta uma nova janela que possibilita a modificação da resposta escolhida. Nessa nova janela, o usuário pode digitar a nova resposta desejada e salvá-la clicando no botão "Salvar". A função "salvar_resposta_modificada" atualiza a resposta na lista original, fecha a janela de modificação e atualiza a lista de respostas na janela principal. O método "atualizar_lista_respostas" atualiza a lista de respostas exibida na janela principal após a modificação. Essas funções oferecem uma maneira intuitiva e direta de editar respostas já inseridas durante o preenchimento do questionário.

Figura X – Geração de Arquivo de Resultados

```
    def gerar_arquivo_resultados(self):
        # Verifique se há respostas para gerar o arquivo
        if not self.respostas:
            messagebox.showinfo("Aviso", "Não há respostas para gerar o arquivo.")
            return

        # Solicite ao usuário o local para salvar o arquivo
        arquivo_nome = tk.filedialog.asksaveasfilename(defaultextension=".txt", filetypes=[("Arquivos de Texto", "*.txt")])
        if arquivo_nome:
            try:
                # Abre o arquivo para escrever as respostas
                with open(arquivo_nome, 'w') as arquivo:
                    for pergunta, resposta in zip(self.perguntas, self.respostas):
                        linha = f"{pergunta}\nResposta: {resposta}\n{'-'*30}\n"
                        arquivo.write(linha)
                messagebox.showinfo("Sucesso", "Arquivo de resultados criado com sucesso!")
            except Exception as e:
                messagebox.showerror("Erro", f"Erro ao criar o arquivo: {str(e)}")


if __name__ == "__main__":
    janela = tk.Tk()
    app = InterfacePerguntas(janela)
    janela.mainloop()
```

Fonte: Autores (2023)

A função "gerar_arquivo_resultados" verifica se há respostas para criar um arquivo de texto contendo as perguntas e respostas fornecidas. Solicita ao usuário um local para salvar o arquivo e, caso seja escolhido, cria um arquivo *.txt* com as perguntas, suas respostas e marcadores visuais. Caso ocorra algum erro durante o processo de criação do arquivo, uma mensagem de erro é exibida. O código final verifica se o script está sendo executado diretamente (não como um módulo importado) e inicializa a interface gráfica das perguntas em uma janela principal. Essa função oferece uma maneira conveniente de armazenar e salvar os resultados coletados durante a interação com o questionário.

5. Considerações finais

Este estudo investigou a interseção entre a linguagem de programação *Python* e sistemas de produção modernos e buscou otimizar a coleta e análise de dados empresariais. Desde a introdução, a crescente importância do *Python* nestes sistemas tem sido destacada, destacando a necessidade de soluções ágeis de aquisição de dados. O objetivo principal deste trabalho foi desenvolver um programa em *Python* customizado com interface gráfica que simplificasse a coleta e análise de dados em ambientes de produção complexos. Esse esforço foi direcionado para aproveitar a flexibilidade do *Python* na extração de dados de diversas fontes, oferecendo uma plataforma robusta para manipular e interpretar informações, beneficiando assim as operações e decisões das empresas.

Os resultados revelaram não só a eficácia do programa desenvolvido, mas também a sua capacidade de proporcionar aos utilizadores uma experiência interativa e intuitiva. Cada

elemento, desde o login até os questionários interativos, foi cuidadosamente projetado para facilitar a coleta de dados e a manipulação eficiente de dados empresariais.

Este trabalho contribuiu para a integração entre os desafios da codificação em linguagem *Python* e dos sistemas de produção além de estabelecer uma base sólida para futuras iterações. A solução apresentada oferece oportunidades para implementar algoritmos avançados para identificar padrões em dados. As direções futuras poderiam explorar técnicas de aprendizado de máquina para análise preditiva, melhorando a capacidade do programa de informar decisões estratégicas e ajudar as empresas nas questões tais como previsão de demanda, otimização de processos e personalização de experiências para os clientes.

REFERÊNCIAS

ARBEGAUS, A. A. Indústria 4.0 eo setor de alimentos e bebidas. **Tecnológica**, 2018. Disponível em: https://www.teclogica.com.br/industria-4-0-e-o-setor-de-alimentos-e-bebida/. Acesso em: 21 de outubro de 2023.

CONFORTO, Debora et al. Pensamento computacional na educação básica: interface tecnológica na construção de competências do século XXI. **Revista Brasileira de Ensino de Ciências e Matemática**, v. 1, n. 1, 2018.

DE ARAUJO, Rodrigo Ramos et al. Análise e classificação de Linguagens de Programação Musical. **Revista Vórtex**, v. 6, n. 2, 2018.

DE SOUZA, Franciely Alves; FALCÃO, Taciana Pontual; MELLO, Rafael Ferreira. O ensino de programação na Educação Básica: uma revisão da literatura. **Anais do XXXII Simpósio Brasileiro de Informática na Educação**, p. 1265-1275, 2021.

FORMIGONI, Philipe de Araújo Fernandes. Python na análise de dados: estudo de caso com dados de acidentes aéreos no Brasil. 2021.

HUBERMAN, Leo. História da riqueza do homem. 1986.

KAGERMANN, Henning; LUKAS, Wolf-Dieter; WAHLSTER, Wolfgang. Industrie 4.0: Mit dem Internet der Dinge auf dem Weg zur 4. industriellen Revolution. **VDI nachrichten**, v. 13, n. 1, p. 2-3, 2011.

KAHN, Ken. A half-century perspective on Computational Thinking. **Tecnologias, sociedade e conhecimento**, v. 4, n. 1, p. 23-42, 2017.

LIKER, Jeffrey K. **O modelo Toyota: 14 princípios de gestão do maior fabricante do mundo**. Bookman Editora, 2021.

LOPES, Gesiel Rios et al. Introdução à análise exploratória de dados com python. **Minicursos ERCAS ENUCMPI**, v. 2019, p. 160-176, 2019.

MANFRIN PRADO, Lara. Desenvolvimento de uma interface de Business Intelligence para análise de dados de afastamentos médicos em uma mineradora. 2021.

OHNO, Taiichi. **O sistema Toyota de produção além da produção**. Bookman, 1997.

PINTO, Bruno Augusto Oliveira. Desenvolvimento do gêmeo digital de um manipulador robótico e preparação para coleta em sistema PIMS. 2023.

PRENSKY, Marc. Nativos digitais, imigrantes digitais parte 2: Eles realmente pensam diferente?. **No horizonte** , v. 9, n. 6, pág. 1-6, 2001.

SCHWAB, Klaus. **A quarta revolução industrial** . Moeda, 2017.