Ramo Estudantil IEEE - UEL

Guilherme Possari (guilherme.possari@uel.br)
Danilo Kotaka Marana (danilo.kotaka.marana@uel.br)
Henry Eiji Maeda Takemoto (henry.maedatakemoto@uel.br)
Livia Kouketsu da Silva (livia.kouketsu@uel.br)
Fabio Jun Ido Nakagawa (fabio.nakagawa@uel.br)

RELATÓRIO FINAL:

Calculadora de

Resistores

Diretor de Projetos: Nathan Andreani Netzel

Londrina
2023

Contato do Ramo: sb.uel@ieee.org
Institute of Electrical and Electronics Engineers – IEEE
Universidade Estadual de Londrina - UEL • Paraná - Brasil

Ramo Estudantil IEEE - UEL

Guilherme Possari
Danilo Kotaka Marana
Henry Eiji Maeda Takemoto
Livia Kouketsu da Silva
Fabio Jun Ido Nakagawa

RELATÓRIO FINAL:

CALCULADORA DE RESISTORES

Relatório apresentado ao Ramo Estudantil IEEE da Universidade Estadual de Londrina.

Londrina
2023

Contato do Ramo: sb.uel@ieee.org
Institute of Electrical and Electronics Engineers – IEEE
Universidade Estadual de Londrina - UEL • Paraná - Brasil

POSSARI, Guilherme. MARANA, Danilo Kotaka. TAKEMOTO, Henry Eiji Maeda. SILVA, Livia Kouketsu da. NAKAGAWA, Fábio Jun Ido. **RELATÓRIO FINAL:** CALCULADORA DE RESISTORES. 2023. Número total de folhas: 12. Relatório apresentado ao Ramo Estudantil IEEE da Universidade Estadual de Londrina, Londrina, 2023.

RESUMO

O projeto da Calculadora de Resistores consiste na utilização de Python para a criação de um programa que possibilita o cálculo do valor da resistência e da taxa de tolerância de um resistor de 4 ou 5 faixas através de uma interface própria em que esses dados são inseridos pelo usuário.

Palavras-chave: Resistor. Resistência. Taxa de Tolerância. Python

OBJETIVOS

Realizar a programação de um software que possibilite a o cálculo da resistência e da taxa de resistência de um resistor através das cores de suas faixas. Criar uma interface para que o usuário insira esses dados.

SUMÁRIO

1 FERRAMENTAS UTILIZADAS 5

1.1 Python 5

1.2 Tkinter 6

1.3 Visual Studio Code 6

2 METODOLOGIA 7

2.1 Calculadora 7

2.2 Interface 8

3 RESULTADOS E DISCUSSÕES 10

CONCLUSÕES 11

1 FERRAMENTAS UTILIZADAS

1.1 Python:

O Python é uma linguagem de programação amplamente usada em aplicações da Web, desenvolvimento de software, ciência de dados e machine learning (ML). Os desenvolvedores usam o Python porque é eficiente e fácil de aprender e pode ser executada em muitas plataformas diferentes. O software Python pode ser baixado gratuitamente, integra-se bem a todos os tipos de sistema e agiliza o desenvolvimento. Os benefícios do Python incluem, mas não se restringem a:

- Os desenvolvedores podem ler e entender facilmente um programa Python, porque tem uma sintaxe básica semelhante à do inglês.
- O Python tem uma grande biblioteca-padrão que contém códigos reutilizáveis para quase todas as tarefas. Como resultado, os desenvolvedores não precisam escrever códigos do zero;
- Os desenvolvedores podem usar o Python facilmente com outras linguagens de programação populares, como Java, C e C++.
- Python é uma linguagem de programação Open Source, e com isso conta com a colaboração de diversos profissionais espalhados nos quatro cantos do mundo, realizando correções e melhorias contínuas;
- Python é uma linguagem multiplataforma, o que significa que os programas escritos em Python podem ser executados em diferentes sistemas operacionais, como Windows, macOS e Linux.

Figura 1 - Python

Fonte: <https://apexensino.com.br/wp-content/uploads/2020/05/python-1280x640.jpg

1.2 Tkinter (Tkinter == Tk Interface.)

De uma maneira geral, podemos afirmar que o Tkinter se trata de uma “biblioteca” que pode ser acessada pelo Python. O mesmo utiliza essa biblioteca apenas com o chamado: import tkinter ou from tkinter import *.

A biblioteca Tkinter é usada em multiplataformas (Unix, Microsoft Windows), para desenvolver interface gráfica, ressaltando que ela não faz parte do Python e sim o Python faz uso dela.

Figura 2 - Tkinter

```
import tkinter as tk

class Application(tk.Frame):
    def __init__(self, master=None):
        super().__init__(master)
        self.master = master
        self.pack()
        self.create_widgets()
```

Fonte: <https://deinfo.uepg.br/~alunoso/2019/AEP/TKINTER/TKINTER_arquivos/image001.png>

1.3 Visual Studio Code

O Visual Studio Code é um editor de código-fonte gratuito e de código aberto desenvolvido pela Microsoft. Ele é amplamente utilizado por desenvolvedores de software para escrever, editar e depurar código em diversas linguagens de programação. O VS Code é conhecido por sua leveza, extensibilidade e suporte a uma ampla variedade de extensões, o que o torna uma ferramenta popular para desenvolvimento de software em muitos ambientes e plataformas.

Figura 3 - Visual Studio Code

Fonte: <https://miro.medium.com/v2/resize:fit:720/0*S0gllBsD11p4kfwO.png>

2 METODOLOGIA

2.1 Calculadora

O primeiro passo para a elaboração do código foi o estudo aprofundado das características e propriedades de cada cor que influenciaria no resistor, além de como o número de faixas iria interagir com o cálculo final. Após o estabelecimento dos alicerces conceituais que serviram como sustentação fundamental para o desenvolvimento da calculadora, restou traduzi-los em código.

O código responsável pelo cálculo da resistência está concentrado nas funções "calcular_resistencia_e_tolerancia_4" e "calcular_resistencia_e_tolerancia_5", bem como nas funções auxiliares "obter_cor", "obter_tolerancia" e "obter_multiplicador". Estas funções desempenham papéis específicos na lógica matemática por trás do cálculo da resistência, baseado nas cores das faixas do resistor.

As funções auxiliares, como "obter_cor", ajudam a mapear as cores para valores numéricos específicos, enquanto "obter_tolerancia" e "obter_multiplicador" obtêm os valores associados à tolerância e multiplicador, respectivamente.

As funções principais, "calcular_resistencia_e_tolerancia_4" e "calcular_resistencia_e_tolerancia_5", realizam os cálculos propriamente ditos, utilizando os valores obtidos anteriormente, e atualizam o rótulo de resultado na interface gráfica.

Figura 4 - Processo de Cálculo

```
def obter_cor(var, cores):
    return cores.get(var.get(), 0)

def obter_tolerancia(var):
    return TOLERANCIAS.get(var.get(), 0)

def obter_multiplicador(var, multiplicadores):
    return multiplicadores.get(var.get(), 0)

def calcular_resistencia_e_tolerancia_4():
    faixa1 = obter_cor(faixa_vars_4[0], CORES)
    faixa2 = obter_cor(faixa_vars_4[1], CORES)
    faixa3 = obter_multiplicador(faixa_vars_4[2], MULTIPLICADORES)
    faixa4 = obter_tolerancia(faixa_vars_4[3])

    resistencia = (faixa1 * 10 + faixa2) * faixa3
    resultado_label.config(text=f"Resistência: {resistencia} ohms\nTolerância: +- {faixa4:.2f}%")

def calcular_resistencia_e_tolerancia_5():
    faixa1 = obter_cor(faixa_vars_5[0], CORES)
    faixa2 = obter_cor(faixa_vars_5[1], CORES)
    faixa3 = obter_cor(faixa_vars_5[2], CORES)
    faixa4 = obter_multiplicador(faixa_vars_5[3], MULTIPLICADORES)
    faixa5 = obter_tolerancia(faixa_vars_5[4])

    resistencia = (faixa1 * 100 + faixa2 * 10 + faixa3) * faixa4
    resultado_label.config(text=f"Resistência: {resistencia} ohms\nTolerância: +- {faixa5:.2f}%")
```

Fonte: O próprio autor.

2.2 Interface

O código relacionado à interface gráfica é responsável por criar a janela, os widgets e gerenciar a interação com o usuário. Utilizando a biblioteca Tkinter, a interface é construída de maneira intuitiva e interativa.

A configuração inicial da interface inclui a definição da janela principal ("*root*"), seu tamanho e título, bem como a criação de variáveis de controle ("*faixa_vars_4*" e "*faixa_vars_5*") para armazenar as escolhas do usuário. O layout é organizado em um frame ("*frame*"), que contém rótulos, menus suspensos e botões, todos criados com o intuito de facilitar a interação.

A função "*atualizar_widgets"* permite uma atualização dinâmica dos widgets com base na escolha do usuário quanto ao número de faixas. Esta função é essencial para garantir uma interface coesa e adaptável. O loop principal ("*root.mainloop()*") mantém a interface ativa, permitindo a interação contínua do usuário. Ao dividir o código dessa maneira, a estrutura e a lógica tanto do cálculo quanto da interface se tornam mais acessíveis e compreensíveis.

Figura 5 - Configuração da Interface

```
# Configuração da interface gráfica
root = tk.Tk()
root.title("Calculadora de Resistores - IEEE")

# Ajuste do tamanho da janela
largura_janela = 400
altura_janela = 300
root.geometry(f"{largura_janela}x{altura_janela}")

# Variáveis de controle para as opções do usuário
faixa_vars_4 = [tk.StringVar() for _ in range(4)]
faixa_vars_5 = [tk.StringVar() for _ in range(5)]

# Configuração do layout
frame = ttk.Frame(root, padding="10")
frame.grid(column=0, row=0, sticky=(tk.W, tk.E, tk.N, tk.S))

# Número de faixas
faixas_label = ttk.Label(frame, text="Número de faixas:")
faixas_label.grid(column=0, row=0, padx=5, pady=5, sticky=tk.W)

faixas_combobox = ttk.Combobox(frame, values=["4", "5"], state="readonly")
faixas_combobox.grid(column=1, row=0, padx=5, pady=5, sticky=tk.W)
faixas_combobox.bind("<<ComboboxSelected>>", atualizar_widgets)

# Cores das faixas e tolerância
faixa_labels = [None] * 5
faixa_comboboxes = [None] * 5
```

Fonte: O próprio autor.

Figura 6 - Atualização Dinâmica dos Widgets

```
def atualizar_widgets(*args):
    numero_faixas = int(faixas_combobox.get())

    # Remover todos os widgets existentes
    for i in range(5):
        if faixa_labels[i]:
            faixa_labels[i].destroy()
        if faixa_comboboxes[i]:
            faixa_comboboxes[i].destroy()

    # Criar novos widgets
    if numero_faixas == 4:
        for i in range(numero_faixas):
            label = ttk.Label(frame, text=f"Faixa {i + 1}:")
            label.grid(column=0, row=i + 1, padx=5, pady=5, sticky=tk.W)
            faixa_labels[i] = label

            if i == 2:  # Terceira faixa (faixa 3) em resistores de 4 faixas
                combobox = ttk.Combobox(frame, values=list(MULTIPLICADORES.keys()), state="readonly", textvariable=faixa_vars_4[i])
            elif i == 3:  # Quarta faixa (faixa 4) em resistores de 4 faixas
                combobox = ttk.Combobox(frame, values=list(TOLERANCIAS.keys()), state="readonly", textvariable=faixa_vars_4[i])
            else:
                combobox = ttk.Combobox(frame, values=list(CORES.keys()), state="readonly", textvariable=faixa_vars_4[i])

            combobox.grid(column=1, row=i + 1, padx=5, pady=5, sticky=tk.W)
            faixa_comboboxes[i] = combobox
        calcular_button.config(command=calcular_resistencia_e_tolerancia_4)  # Atualiza a função do botão
    elif numero_faixas == 5:
        for i in range(numero_faixas):
            label = ttk.Label(frame, text=f"Faixa {i + 1}:")
            label.grid(column=0, row=i + 1, padx=5, pady=5, sticky=tk.W)
            faixa_labels[i] = label

            if i == 3:  # Quarta faixa (faixa 4) em resistores de 5 faixas
                combobox = ttk.Combobox(frame, values=list(MULTIPLICADORES.keys()), state="readonly", textvariable=faixa_vars_5[i])
            elif i == 4:  # Quinta faixa (faixa 5) em resistores de 5 faixas
                combobox = ttk.Combobox(frame, values=list(TOLERANCIAS.keys()), state="readonly", textvariable=faixa_vars_5[i])
            else:
                combobox = ttk.Combobox(frame, values=list(CORES.keys()), state="readonly", textvariable=faixa_vars_5[i])

            combobox.grid(column=1, row=i + 1, padx=5, pady=5, sticky=tk.W)
            faixa_comboboxes[i] = combobox
        calcular_button.config(command=calcular_resistencia_e_tolerancia_5)  # Atualiza a função do botão
```

Fonte: O próprio autor.

Figura 7 - Final da Configuração da Interface

```
# Cores das faixas e tolerância
faixa_labels = [None] * 5
faixa_comboboxes = [None] * 5

for i in range(5):
    label = ttk.Label(frame, text=f"Faixa {i + 1}:")
    label.grid(column=0, row=i + 1, padx=5, pady=5, sticky=tk.W)
    faixa_labels[i] = label

    combobox = ttk.Combobox(frame, values=list(CORES.keys()), state="readonly", textvariable=faixa_vars_5[i])
    combobox.grid(column=1, row=i + 1, padx=5, pady=5, sticky=tk.W)
    faixa_comboboxes[i] = combobox

# Resultado
resultado_label = ttk.Label(frame, text="")
resultado_label.grid(column=0, row=6, columnspan=2, pady=10)

# Botão de cálculo
calcular_button = ttk.Button(frame, text="Calcular", command=calcular_resistencia_e_tolerancia_5)
calcular_button.grid(column=0, row=7, columnspan=2, pady=10)

# Inicie a interface gráfica
root.mainloop()
```

Fonte: O próprio autor.

3 RESULTADOS E DISCUSSÕES

Após a finalização do código, foi checado o funcionamento do cálculo e da interface. Ao inserir as cores desejadas, a interface demonstrava-se responsiva e os resultados mantiveram-se dentro do esperado, assim, verificando o funcionamento correto do programa.

CONCLUSÕES

A partir da testagem do funcionamento do código foi possível aferir que tanto a calculadora quanto a interface do software estavam funcionando e, logo, que a programação foi feita de maneira correta, assim, finalizando o projeto.

REFERÊNCIAS

O que é Python?. Disponível em:<https://aws.amazon.com/pt/what-is/python/>. Acesso em: 10 dez. 2023.

Tkinter. Disponível em:<https://deinfo.uepg.br/~alunoso/2019/AEP/TKINTER/TKINTER.html>. Acesso em: 10 dez. 2023.

Código de cores de resistores. Disponível em:<https://www.mundodaeletrica.com.br/codigo-de-cores-de-resistores/>. Acesso em: 10 dez. 2023.