Programação em Python

Feito por: Yann Xavier
Análise De Sistemas
Uninassau Paulista

Índice

---

1. Conceitos Básicos

Nesta seção, vamos abordar os conceitos básicos da linguagem Python, incluindo:

O que é Python?

Instalação e Configuração

Sintaxe Básica

Tipos de Dados

Operadores

Comentários

Exemplos práticos

---

2. Testes Condicionais

Aqui, aprenderemos sobre testes condicionais em Python, incluindo:

Estruturas de decisão (if, elif, else)

Operadores de comparação

Operadores lógicos

Exemplos e prática

---

3. Laços e Loops

Nesta seção, vamos explorar os laços e loops em Python:

For loops

While loops

Controle de fluxo (break, continue)

Exemplos práticos

---

4. Funções

Aprenderemos sobre funções em Python, incluindo:

Definição e chamada de funções

Parâmetros e argumentos

Retorno de valores

Escopo de variáveis

Funções lambda

Recursão

Exemplos e exercícios

---

5. Módulos

Nesta seção, vamos discutir sobre módulos em Python:

Importação de módulos

Criando e utilizando módulos personalizados

Pacotes

Exemplos práticos

---

6. Listas

Aqui, abordaremos um dos tipos de dados mais utilizados em Python: listas:

Definição de listas

Acesso a elementos

Manipulação de listas (adição, remoção, fatiamento)

List comprehensions

Exemplos e exercícios

---

7. Tuplas

Vamos aprender sobre tuplas em Python:

Definição e características

Acesso a elementos

Imutabilidade

Funções de tuplas

Exemplos práticos

---

8. Dicionários

Nesta seção, exploraremos os dicionários em Python:

Definição e estrutura

Acesso a elementos

Manipulação de dicionários

Métodos de dicionários

Exemplos e exercícios

---

9. Arquivos

Aprenderemos sobre manipulação de arquivos em Python:

Abrindo e fechando arquivos

Leitura e escrita de arquivos

Manipulação de arquivos de texto e binários

Trabalhando com JSON

Exemplos práticos

---

10. Strings

Aqui, vamos discutir sobre manipulação de strings em Python:

Definição de strings

Indexação e fatiamento

Métodos de strings

Formatação de strings

Expressões regulares

Exemplos e exercícios

---

11. Orientação a Objetos

Nesta seção, introduziremos a programação orientada a objetos em Python:

Classes e objetos

Atributos e métodos

Encapsulamento

Herança

Polimorfismo

Exemplos práticos

---

12. GUI - Programação Gráfica

Finalmente, vamos abordar a criação de interfaces gráficas em Python:

Introdução às GUIs

Tkinter: uma biblioteca para GUIs em Python

Criando janelas e widgets

Gerenciamento de layouts

Eventos e interatividade

Exemplos de aplicações GUI

Esta apostila visa fornecer uma base sólida para programação em Python, desde os conceitos básicos até tópicos mais avançados, como programação orientada a objetos e desenvolvimento de interfaces gráficas. Ao longo do material, serão fornecidos exemplos práticos e exercícios para reforçar o aprendizado. Esperamos que este recurso seja útil em sua jornada de aprendizado em Python.

Programação em Python

1. Conceitos Básicos

Python é uma linguagem de programação de alto nível, interpretada, de script, imperativa, orientada a objetos, de tipagem dinâmica e forte. Ela é conhecida por

sua sintaxe simples e legibilidade, o que a torna uma ótima opção tanto para iniciantes quanto para profissionais experientes.

Instalação e Configuração

Para começar a programar em Python, você precisa ter o interpretador Python instalado em seu sistema. Você pode baixá-lo gratuitamente em python.org. Siga as instruções de instalação adequadas para o seu sistema operacional.

Após a instalação, você pode verificar se o Python está corretamente instalado digitando python --version no terminal. Isso deve exibir a versão do Python que você instalou.

Sintaxe Básica

Vamos começar com um programa simples em Python, o famoso "Hello, World!":

pythonCopy code

```
print("Hello, World!")
```

Este programa exibirá "Hello, World!" na tela quando executado. Em Python, print() é uma função embutida que imprime o texto ou variável passada como argumento na tela.

Tipos de Dados

Python possui diversos tipos de dados embutidos, incluindo:

int: números inteiros, por exemplo, 5, -3, 1000.

float: números de ponto flutuante, por exemplo, 3.14, -0.001, 2.0.

str: strings de texto, por exemplo, "Olá, Mundo!", "Python é incrível!".

bool: valores booleanos, True ou False.

list: coleção ordenada de itens, por exemplo, [1, 2, 3, 4], ['a', 'b', 'c'].

tuple: coleção ordenada e imutável de itens, por exemplo, (1, 2, 3), ('a', 'b', 'c').

dict: coleção não ordenada de itens representados como pares chave-valor, por exemplo, {'nome': 'João', 'idade': 25}.

Operadores

Python suporta uma variedade de operadores, incluindo:

Operadores aritméticos: +, -, *, /, // (divisão inteira), % (resto da divisão), ** (exponenciação).

Operadores de comparação: == (igual), != (não igual), <, >, <=, >=.

Operadores lógicos: and, or, not.

Operadores de atribuição: =, +=, -=, *=, /=.

E muitos outros.

Comentários

Comentários são trechos de texto que são ignorados pelo interpretador Python. Eles são úteis para documentar seu código e fazer anotações. Em Python, os comentários começam com o caractere # e vão até o final da linha.

pythonCopy code

Este é um comentário em Python print("Olá, Mundo!") # Esta linha imprime "Olá, Mundo!"

Os comentários não afetam o funcionamento do código, mas são úteis para explicar o que o código faz.

Exemplos práticos

A melhor maneira de aprender Python é praticando! Aqui estão alguns exemplos práticos para você começar:

Calculadora Simples:

pythonCopy code

Adição resultado = 10 + 5 print("Resultado da adição:", resultado) # Subtração resultado = 10 - 5 print("Resultado da subtração:", resultado) # Multiplicação resultado = 10 * 5 print("Resultado da multiplicação:", resultado) # Divisão resultado = 10 / 5 print("Resultado da divisão:", resultado)

Verificar se um número é par ou ímpar:

pythonCopy code

numero = int(input("Digite um número: ")) if numero % 2 == 0: print("O número é par.") else: print("O número é ímpar.")

_____

2. Testes Condicionais

Os testes condicionais são usados em Python para tomar decisões com base em condições específicas. Aqui estão os principais tópicos desta seção:

Estruturas de decisão (if, elif, else)

Em Python, a estrutura de decisão if é usada para executar um bloco de código se uma condição for verdadeira. Podemos usar também elif (abreviação de "else if") e else para lidar com múltiplas condições.

pythonCopy code

idade = 18 if idade < 18: print("Você é menor de idade.") elif idade == 18: print("Você acabou de atingir a maioridade.") else: print("Você é maior de idade.")

Operadores de comparação

Python suporta vários operadores de comparação, como == (igual a), != (diferente de), < (menor que), > (maior que), <= (menor ou igual a), >= (maior ou igual a).

pythonCopy code

x = 5 y = 10 if x == y: print("x é igual a y.") elif x != y: print("x é diferente de y.") elif x < y: print("x é menor que y.") elif x > y: print("x é maior que y.") elif x <= y: print("x é menor ou igual a y.") elif x >= y: print("x é maior ou igual a y.")

Operadores lógicos

Em Python, podemos usar operadores lógicos como and, or e not para combinar ou inverter condições.

pythonCopy code

idade = 25 genero = "Feminino" if idade >= 18 and genero == "Feminino": print("Você é uma mulher maior de idade.") else: print("Você não é uma mulher maior de idade.")

Exemplos e prática

Pratique escrevendo seus próprios testes condicionais em Python para melhorar sua compreensão desses conceitos.

---

3. Laços e Loops

Os laços e loops são usados em Python para executar um bloco de código várias vezes. Aqui estão os principais tópicos desta seção:

For loops

O loop for é usado para iterar sobre uma sequência (como uma lista, tupla, string, etc.) e executar um bloco de código para cada elemento na sequência.

pythonCopy code

frutas = ["maçã", "banana", "laranja"] for fruta in frutas: print(fruta)

While loops

O loop while é usado para executar um bloco de código enquanto uma condição especificada for verdadeira.

pythonCopy code

numero = 1 while numero <= 5: print(numero) numero += 1

Controle de fluxo (break, continue)

Você pode usar break para sair de um loop antes que ele seja concluído e continue para pular para a próxima iteração do loop.

pythonCopy code

for numero in range(10): if numero == 5: break print(numero)

pythonCopy code

for numero in range(10): if numero % 2 == 0: continue print(numero)

Exemplos práticos

Pratique escrevendo seus próprios loops em Python para entender como eles funcionam em diferentes situações.

---

4. Funções

As funções em Python permitem agrupar um conjunto de instruções em um bloco e reutilizá-lo em diferentes partes do código. Aqui estão os principais tópicos desta seção:

Definição e chamada de funções

Para definir uma função em Python, usamos a palavra-chave def, seguida pelo nome da função e seus parâmetros. Para chamar uma função, simplesmente digitamos seu nome, seguido pelos argumentos entre parênteses.

pythonCopy code

def saudacao(nome): print("Olá,", nome) saudacao("João")

Parâmetros e argumentos

As funções podem ter zero ou mais parâmetros, que são variáveis usadas para passar informações para a função. Os argumentos são os valores reais passados para a função quando ela é chamada.

pythonCopy code

def soma(a, b): return a + b resultado = soma(5, 3) print(resultado)

Retorno de valores

Uma função pode retornar um valor usando a palavra-chave return. Isso permite que a função forneça um resultado de volta para onde foi chamada.

pythonCopy code

def soma(a, b): return a + b resultado = soma(5, 3) print(resultado)

Escopo de variáveis

As variáveis em Python têm escopo, o que significa que elas só podem ser acessadas dentro de certas partes do código. Variáveis definidas dentro de uma função têm escopo local, enquanto as definidas fora de qualquer função têm escopo global.

pythonCopy code

def minha_funcao(): x = 10 print(x) minha_funcao() print(x) # Isso gerará um erro, pois x não está definido neste escopo.

Funções lambda

As funções lambda são pequenas funções anônimas que podem ter qualquer número de argumentos, mas apenas uma expressão. Elas são definidas usando a palavra-chave lambda.

pythonCopy code

dobro = lambda x: x * 2 print(dobro(5))

Recursão

A recursão é um conceito em programação onde uma função chama a si mesma. É útil para resolver problemas que podem ser divididos em casos menores e idênticos.

pythonCopy code

def fatorial(n): if n == 1: return 1 else: return n * fatorial(n - 1) print(fatorial(5))

Exemplos e exercícios

Pratique escrevendo suas próprias funções em Python e resolvendo exercícios para reforçar o aprendizado.

---

5. Módulos

Em Python, um módulo é um arquivo contendo definições e instruções Python. O nome do arquivo é o nome do módulo com a extensão .py. Aqui estão os principais tópicos desta seção:

Importação de módulos

Você pode importar módulos em Python usando a instrução import, seguida pelo nome do módulo.

pythonCopy code

import math print(math.sqrt(16)) # Calcula a raiz quadrada de 16 usando a função sqrt do módulo math

Criando e utilizando módulos personalizados

Você pode criar seus próprios módulos em Python escrevendo código em um arquivo .py e, em seguida, importá-lo em outro script Python.

Suponha que você tenha um arquivo chamado meu_modulo.py com o seguinte conteúdo:

pythonCopy code

def saudacao(nome): print("Olá,", nome)

Você pode importar e usar esse módulo em outro arquivo Python:

pythonCopy code

import meu_modulo meu_modulo.saudacao("Maria")

Pacotes

Um pacote em Python é uma coleção de módulos organizados em um diretório. Um pacote deve conter um arquivo especial chamado __init__.py. Você pode importar um módulo específico de um pacote usando a notação de ponto.

Suponha que você tenha um pacote chamado meu_pacote com os seguintes arquivos:

markdownCopy code

meu_pacote/ __init__.py modulo1.py modulo2.py

Você pode importar modulo1 da seguinte maneira:

pythonCopy code

from meu_pacote import modulo1 modulo1.funcao()

Exemplos práticos

Experimente criar seus próprios módulos e pacotes em Python e importá-los em outros scripts para ver como eles funcionam.

---

6. Listas

As listas são uma estrutura de dados comuns em Python usadas para armazenar uma coleção ordenada de itens. Aqui estão os principais tópicos desta seção:

Definição de listas

Em Python, uma lista é definida colocando os elementos entre colchetes [], separados por vírgulas.

pythonCopy code

numeros = [1, 2, 3, 4, 5] nomes = ["João", "Maria", "José"] misturado = [1, "dois", True, 3.14]

Acesso a elementos

Você pode acessar elementos individuais de uma lista usando a indexação. A indexação em Python começa em 0.

pythonCopy code

numeros = [1, 2, 3, 4, 5] primeiro_numero = numeros[0] segundo_numero = numeros[1] print(primeiro_numero) # Saída: 1 print(segundo_numero) # Saída: 2

Manipulação de listas (adição, remoção, fatiamento)

Python oferece várias maneiras de manipular listas, como adicionar elementos, remover elementos e fatiar listas.

pythonCopy code

Adicionar elementos numeros.append(6) # Adiciona 6 ao final da lista numeros.insert(0, 0) # Insere 0 na posição 0 da lista # Remover elementos numeros.pop() # Remove e retorna o último elemento da lista numeros.remove(3) # Remove o elemento 3 da lista # Fatiar listas parte_da_lista = numeros[1:3] # Retorna uma nova lista contendo os elementos da posição 1 à 2

List comprehensions

As list comprehensions são uma maneira concisa de criar listas em Python. Elas permitem que você crie listas usando uma única linha de código.

pythonCopy code

quadrados = [x**2 for x in range(10)] pares = [x for x in range(10) if x % 2 == 0]

Exemplos e exercícios

Pratique trabalhar com listas em Python, escrevendo seu próprio código e resolvendo exercícios para melhorar suas habilidades.

---

7. Tuplas

As tuplas são estruturas de dados semelhantes às listas, mas com a diferença fundamental de que elas são imutáveis, ou seja, uma vez criadas, não podem ser alteradas. Aqui estão os principais tópicos desta seção:

Definição e características

Uma tupla é definida colocando os elementos entre parênteses (), separados por vírgulas.

pythonCopy code

tupla1 = (1, 2, 3) tupla2 = ('a', 'b', 'c')

As tuplas têm tamanho fixo e os elementos não podem ser adicionados, removidos ou alterados após a criação.

Acesso a elementos

Assim como nas listas, você pode acessar elementos individuais de uma tupla usando a indexação.

pythonCopy code

tupla = (1, 2, 3, 4, 5) primeiro_elemento = tupla[0] segundo_elemento = tupla[1] print(primeiro_elemento) # Saída: 1 print(segundo_elemento) # Saída: 2

Imutabilidade

Uma vez que uma tupla é criada, seus elementos não podem ser alterados.

pythonCopy code

tupla = (1, 2, 3) tupla[0] = 10 # Isso gerará um erro, pois as tuplas são imutáveis

Funções de tuplas

Python fornece algumas funções integradas para trabalhar com tuplas, como len(), max(), min() e sum().

pythonCopy code

tupla = (1, 2, 3, 4, 5) tamanho = len(tupla) maior_valor = max(tupla) menor_valor = min(tupla) soma_dos_valores = sum(tupla)

Exemplos práticos

Experimente criar e trabalhar com tuplas em Python para entender melhor como elas funcionam e em que situações são úteis.

_____

8. Dicionários

Os dicionários são uma estrutura de dados em Python que permite armazenar pares de chave-valor. Aqui estão os principais tópicos desta seção:

Definição e estrutura

Um dicionário é definido colocando os pares de chave-valor entre chaves {}, separados por vírgulas.

pythonCopy code

dicionario = {'nome': 'João', 'idade': 30, 'cidade': 'São Paulo'}

Acesso a elementos

Você pode acessar os valores de um dicionário usando suas chaves.

pythonCopy code

print(dicionario['nome']) # Saída: João print(dicionario['idade']) # Saída: 30

Manipulação de dicionários

Python oferece várias maneiras de manipular dicionários, como adicionar novos pares chave-valor, modificar valores existentes e remover itens.

pythonCopy code

Adicionar elementos dicionario['email'] = 'joao@example.com' # Modificar valores dicionario['idade'] = 31 # Remover elementos del dicionario['cidade']

Métodos de dicionários

Python fornece uma variedade de métodos integrados para trabalhar com dicionários, como keys(), values(), items().

pythonCopy code

chaves = dicionario.keys() # Retorna uma lista de chaves valores = dicionario.values() # Retorna uma lista de valores itens = dicionario.items() # Retorna uma lista de tuplas (chave, valor)

Exemplos e exercícios

Pratique trabalhar com dicionários em Python, escrevendo seu próprio código e resolvendo exercícios para melhorar suas habilidades.

---

9. Arquivos

O trabalho com arquivos é uma parte essencial da programação em Python, pois permite que você armazene e manipule dados de forma persistente. Aqui estão os principais tópicos desta seção:

Abrindo e fechando arquivos

Antes de trabalhar com um arquivo, você precisa abri-lo usando a função open(). Ao terminar de trabalhar com o arquivo, é uma prática recomendada fechá-lo usando o método close().

pythonCopy code

arquivo = open("arquivo.txt", "r") # Abre o arquivo para leitura conteudo = arquivo.read() print(conteudo) arquivo.close() # Fecha o arquivo

Leitura e escrita de arquivos

Você pode ler o conteúdo de um arquivo usando os métodos read(), readline() ou readlines(). Para escrever em um arquivo, você pode usar os métodos write() ou writelines().

pythonCopy code

Leitura arquivo = open("arquivo.txt", "r") conteudo = arquivo.read() print(conteudo) arquivo.close() # Escrita arquivo = open("novo_arquivo.txt", "w") arquivo.write("Olá, mundo!") arquivo.close()

Manipulação de arquivos de texto e binários

Em Python, você pode trabalhar tanto com arquivos de texto quanto com arquivos binários. Ao abrir um arquivo, você precisa especificar o modo de abertura, como "r" para leitura de texto, "rb" para leitura de binário, "w" para escrita de texto, "wb" para escrita de binário, etc.

pythonCopy code

Leitura de arquivo binário with open("imagem.jpg", "rb") as arquivo: conteudo = arquivo.read() # Escrita de arquivo binário with open("nova_imagem.jpg", "wb") as arquivo: arquivo.write(conteudo)

Trabalhando com JSON

JSON (JavaScript Object Notation) é um formato de dados amplamente utilizado para armazenar e trocar dados. Python possui suporte integrado para trabalhar com JSON usando o módulo json.

pythonCopy code

import json # Serialização (Python para JSON) dados = {'nome': 'João', 'idade': 30} json_string = json.dumps(dados) # Desserialização (JSON para Python) dados_de_json = json.loads(json_string)

Exemplos práticos

Pratique trabalhar com arquivos em Python, escrevendo código para ler, escrever e manipular arquivos de texto e binários, bem como trabalhar com JSON para armazenar e recuperar dados.

---

10. Strings

As strings são uma sequência de caracteres em Python, e você pode realizar várias operações e manipulações com elas. Aqui estão os principais tópicos desta seção:

Definição de strings

As strings em Python podem ser definidas usando aspas simples ' ', aspas duplas " ", ou até mesmo aspas triplas ''' ''' ou """ """ para strings multilinhas.

pythonCopy code

saudacao = "Olá, mundo!" mensagem = 'Python é incrível!' paragrafo = '''Este é um exemplo de um parágrafo multilinhas.'''

Indexação e fatiamento

Você pode acessar caracteres individuais de uma string usando a indexação. Além disso, você pode fatiar strings para obter substrings específicas.

pythonCopy code

frase = "Python é uma linguagem poderosa" primeiro_caractere = frase[0] ultimos_caracteres = frase[-3:] substring = frase[7:9]

Métodos de strings

Python fornece uma ampla gama de métodos integrados para trabalhar com strings, como upper(), lower(), strip(), split(), join(), replace(), etc.

pythonCopy code

texto = " Python é incrível " maiusculas = texto.upper() minusculas = texto.lower() sem_espacos = texto.strip() palavras = texto.split()

Formatação de strings

Você pode formatar strings em Python usando o método format() ou f-strings (strings formatadas).

pythonCopy code

nome = "Maria" idade = 30 mensagem = "Olá, {}! Você tem {} anos.".format(nome, idade) mensagem_formatada = f"Olá, {nome}! Você tem {idade} anos."

Expressões regulares

As expressões regulares são sequências de caracteres que formam um padrão de busca. Python oferece suporte a expressões regulares através do módulo re.

pythonCopy code

import re texto = "Python é uma linguagem de programação" padrao = r"Python" correspondencias = re.findall(padrao, texto)

Exemplos e exercícios

Pratique trabalhar com strings em Python, escrevendo seu próprio código e resolvendo exercícios para melhorar suas habilidades.

11. Orientação a Objetos

A orientação a objetos é um paradigma de programação que modela os problemas do mundo real usando objetos e classes. Aqui estão os principais tópicos desta seção:

Classes e objetos

Uma classe é uma estrutura que define o comportamento e as propriedades de um objeto. Um objeto é uma instância de uma classe.

pythonCopy code

class Pessoa: def __init__(self, nome, idade): self.nome = nome self.idade = idade def apresentar(self): print(f"Olá, meu nome é {self.nome} e eu tenho {self.idade} anos.") pessoa1 = Pessoa("João", 30) pessoa1.apresentar()

Atributos e métodos

Os atributos são variáveis associadas a uma classe e os métodos são funções associadas a uma classe.

pythonCopy code

class Retangulo: def __init__(self, comprimento, largura): self.comprimento = comprimento self.largura = largura def calcular_area(self): return self.comprimento * self.largura retangulo1 = Retangulo(5, 3) area = retangulo1.calcular_area() print("Área do retângulo:", area)

Encapsulamento

O encapsulamento é um conceito que impede o acesso direto aos dados de um objeto. Em Python, podemos usar convenções de nomenclatura para indicar que um atributo ou método é privado.

pythonCopy code

class Carro: def __init__(self, marca, modelo): self.marca = marca self.modelo = modelo self.__velocidade = 0 # Atributo privado def acelerar(self, valor): self.__velocidade += valor def get_velocidade(self): return self.__velocidade carro1 = Carro("Toyota", "Corolla") carro1.acelerar(50) print("Velocidade:", carro1.get_velocidade())

Herança

A herança é um conceito em que uma classe pode herdar atributos e métodos de outra classe. A classe pai é chamada de classe base e a classe filha é chamada de classe derivada.

pythonCopy code

class Animal: def __init__(self, nome): self.nome = nome def fazer_som(self): pass class Cachorro(Animal): def fazer_som(self): return "Au au!" class Gato(Animal): def fazer_som(self): return "Miau!" cachorro = Cachorro("Rex") print(cachorro.fazer_som()) gato = Gato("Whiskers") print(gato.fazer_som())

Polimorfismo

O polimorfismo é a capacidade de uma função ou método se comportar de maneira diferente com base no tipo de objeto que está sendo chamado. Em Python, isso é alcançado através do uso de métodos com o mesmo nome em classes diferentes.

pythonCopy code

class Animal: def fazer_som(self): pass class Cachorro(Animal): def fazer_som(self): return "Au au!" class Gato(Animal): def fazer_som(self): return "Miau!" def emitir_som(animal): return animal.fazer_som() cachorro = Cachorro() gato = Gato() print(emitir_som(cachorro)) print(emitir_som(gato))

Exemplos práticos

Pratique trabalhar com orientação a objetos em Python, escrevendo suas próprias classes e objetos para resolver problemas do mundo real.

---

12. GUI - Programação Gráfica

A programação gráfica em Python envolve a criação de interfaces gráficas do usuário (GUIs) para aplicativos desktop. Aqui estão os principais tópicos desta seção:

Introdução às GUIs

As GUIs são interfaces gráficas que permitem aos usuários interagir com os programas através de elementos visuais, como janelas, botões, caixas de texto, etc.

Tkinter: uma biblioteca para GUIs em Python

Tkinter é a biblioteca padrão de GUI para Python. Ela fornece uma maneira simples e rápida de criar interfaces gráficas para seus aplicativos.

pythonCopy code

import tkinter as tk janela = tk.Tk() janela.title("Minha Primeira GUI") janela.geometry("300x200") label = tk.Label(janela, text="Olá, Mundo!") label.pack() janela.mainloop()

Criando janelas e widgets

Em Tkinter, você pode criar janelas e widgets usando classes como Tk, Frame, Label, Button, Entry, etc.

Gerenciamento de layouts

Tkinter oferece vários gerenciadores de layout para organizar widgets dentro de uma janela, como pack(), grid() e place().

Eventos e interatividade

Você pode vincular funções a eventos de widget para tornar sua GUI interativa e responsiva aos cliques do usuário, movimentos do mouse, etc.

pythonCopy code

def botao_clicado(): label.config(text="Botão clicado!") botao = tk.Button(janela, text="Clique aqui", command=botao_clicado) botao.pack()

Exemplos de aplicações GUI

Pratique criando suas próprias GUIs em Python usando Tkinter, criando aplicativos simples e interativos para suas necessidades.