



ESUP - AULA 7

Linguagem de Programação II

Prof. Esp. Silas Augusto Alves Júnior



PYTHON



PYTHON

SOBRE A LINGUAGEM

- Criado por Guido Van Rossum em 1991
- Seu nome originou de um grupo de humoristas: Monty Python
- Fácil de usar
- É uma linguagem de alto nível
- Sintaxe simples
- Possui várias bibliotecas nativas
- Comunidade gigante

Afinal, quem usa Python?

SOBRE A LINGUAGEM



Dropbox

Configuração inicial

WINDOWS

1. Instalar o interpretador disponível: <https://www.python.org/>
2. Instalar a **IDE - Visual Studio Code**
 - a. Instalar a extensão Python

Bora começar?

Hello World

ALGORITMO PODE SER COMPARADO A
UMA RECEITA DE BOLO

1. **Entrada de Dados** / Ingredientes
2. **Processamento de Dados** / Receita
3. **Saída de dados** / Bolo de chocolate com cobertura :)

Print

OUTPUT/SAÍDA

Comando para imprimir mensagens na tela;

O uso mais comum da função `print()` é a exibição de mensagens na tela para orientar o usuário.

Exemplo:

```
print ('Os alunos do terceiro período de SI são um fenômeno')
```

Input

ENTRADA

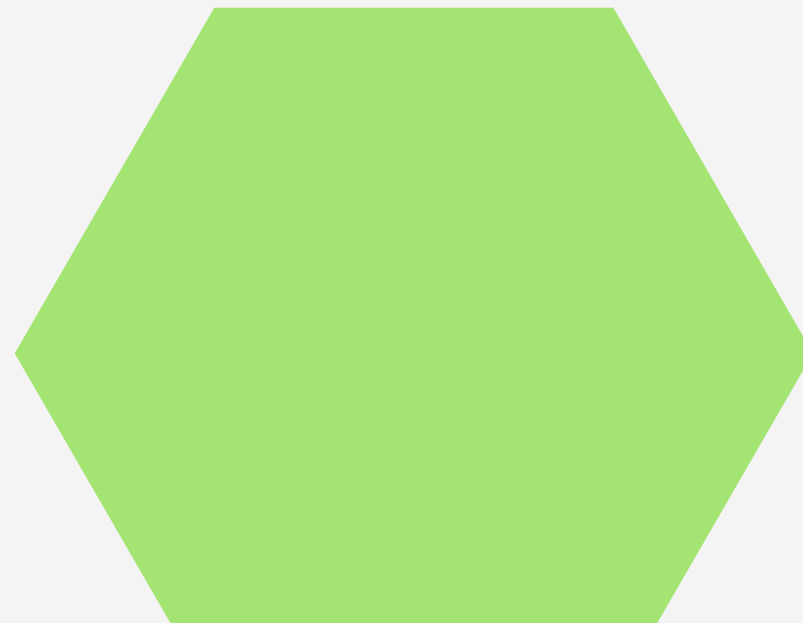
Função para coletar dados de entrada do usuário;

Exemplo:

```
input ('Quantos alunos cursam SI na ESUP?')
```


Variáveis

- As variáveis servem para armazenar valores como: textos, números, listas entre outros
- É como uma caixa que guardamos informações e podemos acessar posteriormente.



Tipo de Variáveis

- int: números inteiros, não tem parte decimal: 1,2,-9,-8,100,123
- float: números reais, ou seja, tem parte decimal: 1.9, 1.8, 3.14.
- str: cadeias de caracteres, dados textuais (string)
- bool: valores booleanos, Verdadeiro (True) ou Falso (False)

Na última aula...

- Conhecemos um pouco da história e as características da linguagem Python
- Configuramos o ambiente para programar: Instalação do Python e da IDE (Visual Studio Code)
- Estrutura de um algoritmo: Entrada / Processamento / Saída
- **Funções:**
 - **Print:** função que permite exibir mensagens ao usuário. (saída de dados).
 - **Input:** função que permite que coletar a entrada de dados pelos usuários.
- **Variáveis e seus tipos:** um "pedaço" reservado na memória para guardar informações e acessar posteriormente.

Operadores aritméticos

- Operadores são símbolos especiais que representam cálculos como adições e multiplicações:
 - + Adição
 - - Subtração
 - / Divisão
 - * Multiplicação
 - ** Potenciação

Strings

- O operador + também funciona com strings de uma maneira diferente dos números. Ele funciona **concatenando** strings, ou seja, juntando elas.

```
>> texto1 = 'Faculdade'
```

```
>> texto2 = 'ESUP'
```

```
>> resultado = texto1+texto2
```

```
>> print(resultado)
```

FaculdadeESUP -> mensagem que foi exibida no terminal

Strings

- O operador `*` também funciona com strings, **multiplicando** seu conteúdo por um inteiro

```
>> texto1 = 'Python '
```

```
>> resultado = texto1 * 3
```

```
>> print(resultado)
```

python python python -> mensagem que foi exibida no terminal

CONVERTENDO UMA STRING PARA INTEIRO

- A função `input()` lê o valor digitado pelo usuário como uma **string**

Ou seja se o usuário digitar na `variavel1 = 20` e na `variavel2 = 20` o resultado da soma seria 2020 se não realizarmos esta conversão.

Para conversão utilizaremos o **int** que pode receber uma string e retornar o inteiro correspondente.

CONVERTENDO UMA STRING PARA INTEIRO

```
Num1 = int(print('Digite o primeiro número'))
```


F-Strings em Python

As f-strings vão servir para que você consiga colocar uma variável dentro de um texto, e isso é feito utilizando a letra “f” antes do texto e colocando a sua variável dentro de {} chaves.

Isso é muito útil para que você não tenha que ficar concatenando o seu texto com as variáveis e tenha que fatiar seu texto várias vezes por conta disso.

F-Strings em Python

Ex.: `print(f'Seu número digitado foi {numero}')`

Isso facilita a sua vida na hora de colocar informações dentro de um texto.

Qual o problema de usar somente o operador (+) para concatenação?

O operador (+) só serve para concatenar textos, ou seja, strings.

As f-strings são uma forma poderosa e eficiente de formatar strings em Python. Elas permitem que você inclua expressões dentro de strings, facilitando a concatenação de palavras e variáveis.

Em nossas aulas iremos usar o F para incorporar expressões e variáveis dentro de uma string. Em nossas aulas usaremos o operador (+) apenas para operações aritméticas.

**Resolvendo o exercício da
aula anterior...**



Operadores booleanos / comparação / relacionais

Os operadores de comparação, também chamados de operadores relacionais, como o próprio nome sugere, comparam ou relacionam valores.

OPERADOR	DESCRIÇÃO	EXEMPLO DE APLICAÇÃO
<code>==</code>	Igual	<code>3 == 2</code> #resulta em False
<code>!=</code>	Diferente	<code>3 != 2</code> #resulta em True
<code>></code>	Maior que	<code>3 > 2</code> #resulta em True
<code><</code>	Menor que	<code>3 < 2</code> #resulta em False
<code>>=</code>	Maior ou igual a	<code>3 >= 2</code> #resulta em True
<code><=</code>	Menor ou igual a	<code>3 <= 2</code> #resulta em False

Operadores lógicos

Python, assim como outras linguagens, possui três operadores lógicos para realização de testes compostos.

OPERADOR	EXPRESSÃO	EXEMPLO DE APLICAÇÃO
<code>and</code>	<code>X and Y</code>	<code>True and False #resulta em True</code>
<code>or</code>	<code>X or Y</code>	<code>True or False #resulta em True</code>
<code>not</code>	<code>not X</code>	<code>not False #resulta em True</code> <code>not True #resulta em False</code>

Estrutura IF...ELSE

A estrutura IF...ELSE é o que nos permite que o programa tome decisões com base em condições específicas. Essa estrutura condicional permite que partes específicas do código sejam executadas ou ignoradas, dependendo se uma determinada condição é verdadeira ou falsa.

```
idade = 25
```

```
if idade >= 18:
```

```
    print("Você é maior de idade.")
```

```
else:
```

```
    print("Você é menor de idade.")
```

Estrutura IF...ELSE

A palavra-chave **else** é aplicada nas condições em que é necessário executar instruções quando o teste do if não for satisfeito, ou seja, o resultado do teste é falso.

Para nós (brasileiros) é mais fácil assimilar quando interpretamos como a expressão “senão”.

Estrutura ELIF

A estrutura 'ELIF' em python é uma combinação das palavras "else" e "if" que são usadas para verificar várias condições em sequência, quando a condição do 'IF' não é atendida (false), o código verifica a próxima condição em um bloco 'elif'. Isso nos permite testar várias condições de maneira organizada.

Estrutura ELIF

ELIF

```
cor = "alguma cor"

if cor == 'verde':
    print('Acelerar')

elif cor == 'amarelo':
    print('Atenção')

else:
    print('Parar')
```

Parar

Exercícios

Faça um programa que receba dois número e mostre qual deles é o maior



Exercícios

Faça um programa que receba dois números e mostre o maior. Se por acaso, os dois números forem iguais, imprima a mensagem "São números iguais"

Exercícios

Faça um programa que pede duas notas de um aluno. Em seguida ele deve calcular a média do aluno e dar o seguinte resultado:

A mensagem "Aprovado", se a média alcançada for maior ou igual a sete;

A mensagem "Reprovado", se a média for menor do que sete;

A mensagem "Aprovado com Distinção", se a média for igual a dez.

Exercícios

As Organizações Guanabara resolveram dar um aumento de salário aos seus colaboradores e lhe contrataram para desenvolver o programa que calculará os reajustes.

Faça um programa que recebe o salário de um colaborador e o reajuste segundo o seguinte critério, baseado no salário atual:

salários até R\$ 280,00 (incluindo) : aumento de 20%

salários entre R\$ 280,00 e R\$ 700,00 : aumento de 15%

salários entre R\$ 700,00 e R\$ 1500,00 : aumento de 10%

salários de R\$ 1500,00 em diante : aumento de 5% .

Após o aumento ser realizado, informe na tela:

o salário antes do reajuste;

o percentual de aumento aplicado;

o valor do aumento;

o novo salário, após o aumento.

Exercícios

Faça um Programa que peça os 3 lados de um triângulo. O programa deverá informar se os valores podem ser um triângulo. Indique, caso os lados formem um triângulo, se o mesmo é: equilátero, isósceles ou escaleno.

Dicas:

Três lados formam um triângulo quando a soma de quaisquer dois lados for maior que o terceiro;

Triângulo Equilátero: três lados iguais;

Triângulo Isósceles: quaisquer dois lados iguais;

Triângulo Escaleno: três lados diferentes;

Roteiro da aula de hoje...

- Resolução da Lista de Exercícios (sorteio com a turma)
- Controle de Fluxos: **While** e **For**
- Listas
- Funções

Próxima aula:

- Interface gráfica (GUI) em Python (Tkinter)
- Utilizando Bibliotecas

Comando While

A função while é uma estrutura de repetição útil na linguagem de programação, nela podemos executar um bloco de código repetidamente até que ela seja **VERDADEIRA**. Isso permite que o programa execute um determinado conjunto de instruções várias vezes, facilitando a automação de tarefas e a manipulação de dados.

Podemos interpretar a expressão como: ***enquanto***.

Comando While - Exemplo

```
contador = 0
```

```
while contador < 5:
```

```
    print("O contador é:", contador)
```

```
    contador += 1
```

bloco de código dentro do **while** será executado enquanto a variável contador for menor que 5. A cada iteração do loop, o valor do contador é incrementado em 1 e exibido na tela.

Comando While - Exemplo

```
senha = input("Digite uma senha: ")
```

```
while senha != "1234":  
    print("Senha incorreta!")  
    senha = input("Digite uma senha: ")
```

```
print("Senha correta! Acesso permitido.")
```

Nesse exemplo, o programa solicita ao usuário que digite uma senha. Enquanto a senha digitada não for igual a “1234”, uma mensagem de senha incorreta é exibida e o usuário é solicitado a digitar novamente. O loop continua até que a senha correta seja digitada.

Exercício para fixação

- Faça um programa, utilizando while, que mostre na tela os números de 0 a 30.

Exercício para fixação

- Faça um programa que receba um número inteiro positivo como entrada e exiba a tabuada desse número até 10.

Comando FOR

A função **FOR** também é uma estrutura de laços de repetição usada para iterar sobre uma sequência de elementos, seu funcionamento é muito parecido com o while.

A diferença entre o FOR e WHILE consiste em:

- No for é usado quando se sabe exatamente quantas vezes você deseja iterar.
- While é usado quando não se sabe exatamente quantas vezes precisa iterar e para continuar é baseada em uma expressão booleana. (falso/verdadeiro)

Comando FOR

A função **FOR** também é uma estrutura de laços de repetição usada para iterar sobre uma sequência de elementos, seu funcionamento é muito parecido com o while.

A diferença entre o FOR e WHILE consiste em:

- No for é usado quando se sabe exatamente quantas vezes você deseja iterar.
- While é usado quando não se sabe exatamente quantas vezes precisa iterar e para continuar é baseada em uma expressão booleana. (falso/verdadeiro)

Comando FOR - Exemplo

Sua sintaxe é:

```
for elemento in sequencia:  
    <comandos>
```

Sendo:

- for a palavra usada para iniciar o loop
- a variável elemento (pode ser qualquer nome) recebe cada elemento da sequência um a um
- a sequência pode ser uma lista, tupla, string, números

Comando FOR - Exemplo

```
frutas = ['maçã', 'banana', 'laranja']
```

```
for fruta in frutas:  
    print(fruta)
```

A saída será:

maçã
banana
laranja

Comando FOR - Exemplo

Entendendo o código anterior:

1. Definimos uma lista frutas com 3 elementos
2. `for fruta in frutas:` Iniciou o loop que irá iterar (passear) sobre cada elemento da lista. Na primeira iteração fruta será igual maçã, na segunda, banana e na terceira laranja.
3. `Print(fruta)` para cada elemento da lista o código irá imprimir o valor da variável fruta.

Comando FOR - Exemplo

Utilizamos a função `range()` para gerar uma sequência de números. Neste exemplo, o laço `for` itera sobre os números de 1 a 5 e imprime cada número na tela.

```
for i in range(1, 6):  
    print(i)
```

Comando FOR - Exemplo

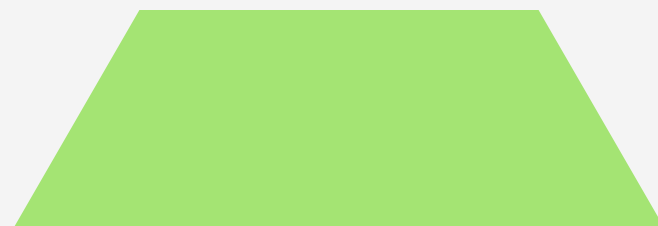
```
for caractere in 'Python':  
    print(caractere)
```

Saída será:

P
y
t
h
o
n

Exercício para fixação

Imprimir na tela o número de 1 até 10



Exercício para fixação

Faça um programa que peça uma nota, entre zero e dez. Mostre uma mensagem caso o valor seja inválido e continue pedindo até que o usuário informe um valor válido.

Exercício para fixação

RESOLUÇÃO:

```
nota = float(input("Digite uma nota entre zero e 10: "))
```

```
while nota > 10 or nota < 0:
```

```
    nota = float(input("Informe um valor válido: "))
```

Exercício para fixação

Faça um programa que leia um nome de usuário e a sua senha e não aceite a senha igual ao nome do usuário, mostrando uma mensagem de erro e voltando a pedir as informações.

Exercício para fixação

RESOLUÇÃO

```
nome = input("Digite o seu nome de usuario: ")
```

```
senha = input("Digite a sua senha: ")
```

```
while senha == nome:
```

```
    print("Erro, a senha não pode ser igual ao nome de usuario!")
```

```
    nome = input("Digite o seu nome de usuario: ")
```

```
    senha = input("Digite a sua senha: ")
```

Exercício para fixação

Faça um programa que receba dois números inteiros e gere os números inteiros que estão no intervalo compreendido por eles.

Exercício para fixação

Faça um programa que receba dois números inteiros e gere os números inteiros que estão no intervalo compreendido por eles.

Exercício para fixação

RESOLUÇÃO

```
n1 = int(input("Digite um número: "))  
n2 = int(input("Digite outro número: "))  
  
for i in range(n1 + 1, n2):  
    print(i)
```

Listas

Uma lista (**list**) em Python é uma sequência ou coleção ordenada de valores. Cada valor na lista é identificado por um índice. Os valores que formam uma lista são chamados elementos ou itens. Em resumo é um tipo de estrutura de dados que permite armazenar uma coleção de itens em uma única variável. As listas são mutáveis, o que significa que podem ser modificadas após serem criadas.

Listas

Existem várias maneiras de se criar uma nova lista. A maneira mais simples é envolver os elementos da lista por colchetes [e]

```
vocabulario = ["iteracao", "selecao", "controle"]
```

```
numeros = [17, 123]
```

```
vazia = []
```

```
lista_mista = ["ola", 2.0, 5*2, [10, 20]]
```

```
print(numeros)
```

```
print(lista_mista)
```

```
nova_lista = [numeros, vocabulario]
```

```
print(nova_lista)
```

Comprimento de uma lista

Da mesma forma que ocorre com strings, a função `len` retorna o comprimento de uma lista (o número de elementos na lista).

```
cursosEsup = ['Sistemas de Informação', 'Direito',  
'Ciências Contábeis', 'Pedagogia',  
'Adminitração']
```

```
print(cursosEsup) #Irá mostrar os elementos da lista  
print(len(cursosEsup)) #Irá mostrar o número 5 que é o tamanho da lista
```

Acessando os elementos

Suponha que no código anterior é necessário só mostrar o terceiro item da lista, nosso código ficaria assim:

```
cursosEsup = ['Sistemas de Informação', 'Direito',  
'Ciências Contábeis', 'Pedagogia',  
'Adminitração']
```

```
print(cursosEsup[3]) #Nosso resultado seria Pedagogia, pois o primeiro  
elemento de uma lista é 0
```


Funções

O conceito de **função** é um dos mais importantes na matemática. Em computação, uma função é uma sequência de instruções que computa um ou mais resultados que chamamos de parâmetros. No capítulo anterior, utilizamos algumas funções já prontas do Python, como o `print()`, `input()` e `type()`.

Também podemos criar nossas próprias funções.

Funções

Em Python, uma função é um bloco de código que realiza uma tarefa específica. Ela é definida usando a palavra-chave **def**, seguida pelo nome da função e parênteses. As funções oferecem uma maneira de organizar e reutilizar código, promovendo a legibilidade e a eficiência.

Ex.:

```
def saudacao():  
    print("Olá, mundo!")
```

Funções - Sintaxe

Formato geral:

```
def nome (arg, arg, ... arg):  
    comando  
    ...  
    comando
```

Onde:

- nome é o nome da função
- args são especificações de argumentos da função
- Uma função pode ter 0, 1 ou mais argumentos
- comandos contêm as instruções a ser executadas quando a função é invocada

Funções - Parâmetros e Argumentos

Uma função pode aceitar parâmetros, ou, argumentos que são valores que ela espera receber. Esses parâmetros são fornecidos entre os parênteses durante a chamada da função. Veja um exemplo:

```
def saudacao(nome):  
    print("Olá, " + nome + "!")
```

Funções - Parâmetros e Argumentos

```
def soma(n1,n2):  
    resultado=n1+n2  
    return resultado
```

```
n1 = int(input ('Digite N1'))  
n2 = int(input ('Digite N2'))
```

```
resultados = soma(n1,n2)  
print (resultados)
```

Funções - Parâmetros e Argumentos

```
def calcular_pagamento(qtd_horas, valor_hora):  
    horas = float(qtd_horas)  
    taxa = float(valor_hora)  
    if horas <= 40:  
        salario=horas*taxa  
    else:  
        h_excd = horas - 40  
        salario = 40*taxa+(h_excd*(1.5*taxa))  
    return salario
```

Funções - Parâmetros e Argumentos

Repare que usamos o return, o comando **return** é usado para encerrar a execução de uma função e enviar um valor de volta para o chamador. Quando uma função é chamada e encontra o comando return, ela retorna imediatamente ao local de chamada, levando consigo o valor especificado. Isso permite que o valor seja utilizado ou armazenado para uso posterior no programa.

Funções - Parâmetros e Argumentos

```
def calcular_pagamento(qtd_horas, valor_hora):  
    horas = float(qtd_horas)  
    taxa = float(valor_hora)
```

```
    if horas <= 40:  
        salario=horas*taxa  
    else:  
        h_excd = horas - 40  
        salario = 40*taxa+(h_excd*(1.5*taxa))  
    return salario
```

```
res = calcular_pagamento(220,130)  
print (res)
```


Funções - Exercício para fixação

Faça um programa, com uma função que necessite de três argumentos, e que forneça a soma desses três argumentos.

Funções - Exercício para fixação

Faça um programa, com uma função que necessite de três argumentos, e que forneça a soma desses três argumentos.

Resolução

```
def exercicio3(a, b, c):  
    return a + b + c
```

```
resul = exercicio3(1,3,4)  
print(resul) # Irá mostrar 8
```

Funções - Exercício para fixação

Faça um programa com uma função chamada `somaImposto`. A função possui dois parâmetros formais: `taxaImposto`, que é a quantia de imposto sobre vendas expressa em porcentagem e `custo`, que é o custo de um item antes do imposto. A função “altera” o valor de `custo` para incluir o imposto sobre vendas.

Funções - Exercício para fixação

Faça um programa com uma função chamada `somaImposto`. A função possui dois parâmetros formais: `taxaImposto`, que é a quantia de imposto sobre vendas expressa em porcentagem e `custo`, que é o custo de um item antes do imposto. A função “altera” o valor de `custo` para incluir o imposto sobre vendas.

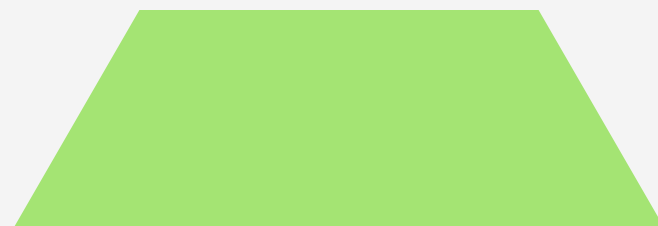
Resolução

```
def somaImposto(taxaImposto,custo):  
    return (0.01*taxaImposto)*custo + custo
```

```
resul = somaImposto(3,100)  
print(resul)
```

Funções - Exercício para fixação

Função para calcular uma média da lista de números



Funções - Exercício para fixação

Função para calcular uma média da lista de números

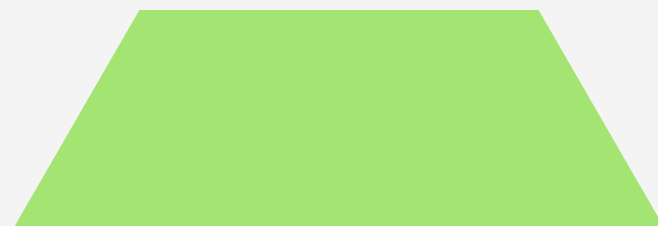
Resolução

```
def calcular_media(lista):  
    total = sum(lista)  
    return total/len(lista)
```

```
numeros = [10,20,230,30,40]  
media = calcular_media(numeros)  
print (media)
```

Na aula de hoje....

- Exercícios de fixação de conteúdo anteriores
- Interface gráfica utilizando o Tkinter



Exercício 1

Crie 2 listas: uma com 5 nomes(João, Maria, Kleber, Caio e Sarah) e outra com 5 valores em reais(R\$) correspondentes ao saldo da conta do usuário(1350,20; 240,50; 30,00; 830,15 e 50,00), e usando laços de repetição imprima os dados da seguinte forma:

Exercício 1

Saída:

Saída/Impressão:

LISTA DE CLIENTES - BANCO NACIONAL

NOME	SALDO	CONTA
nome0	saldo0	#0
nome1	saldo1	#2
nome2	saldo2	#4

Exercício 1 - resolução

```
nomes = ["João", "Maria", "Kleber", "Caio", "Sarah"]  
saldos = [1350.20, 240.50, 30.00, 830.15, 50.00]
```

```
# Imprimindo os dados na forma solicitada
```

```
print("Nome   Saldo")
```

```
print("-----")
```

```
for i in range(5):
```

```
    print(f"{nomes[i]} R$ {saldos[i]}")
```

Exercício 2

Elaborar um programa Python para gerar a sequência de Fibonacci até o décimo dígito.

Obs.: A sequência de Fibonacci é uma sequência numérica infinita em que cada termo a partir do terceiro é a soma dos dois termos anteriores. Portanto, a sequência de Fibonacci é (1,1,2,3,5,8,13,21,34,55...)

Exercício 2 - Resolução

```
print("Sequência de Fibonacci")
```

```
fib1 = 1
```

```
fib2 = 1
```

```
print(fib1)
```

```
print(fib2)
```

```
for i in range(2, 11):
```

```
    fib = fib1 + fib2
```

```
    fib1 = fib2
```

```
    fib2 = fib
```

```
    print(fib)
```

Exercício 3

Escreva uma função em Python chamada **soma_pares** que recebe uma lista de números e retorna a soma dos números pares presentes na lista.

lembrando que: Um número inteiro é par se ele é divisível por 2, ou seja, se a divisão desse número por 2 tem resto igual a 0, ou seja, $\text{num} \% 2 == 0$:

Exercício 3 - Resolução

```
def soma_pares(lista):
```

```
    soma = 0
```

```
    for num in lista:
```

```
        if num % 2 == 0:
```

```
            soma += num
```

```
    return soma
```

```
# Exemplo de uso da função
```

```
lista_numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
resultado = soma_pares(lista_numeros)
```

```
print("A soma dos números pares na lista é:", resultado)
```

Exercício 4

Escreva uma função em Python chamada `maior_palavra` que recebe uma lista de palavras como entrada e retorna a maior palavra presente na lista.

Exercício 4 - Resolução

```
def maior_palavra(lista_palavras):  
    maior = ""  
    for palavra in lista_palavras:  
        if len(palavra) > len(maior):  
            maior = palavra  
    return maior
```

Exemplo de uso da função

```
lista_palavras = ["python", "exercício", "desafiador", "programação",  
"linguagem"]  
resultado = maior_palavra(lista_palavras)  
print("A maior palavra na lista é:", resultado)
```


Exercício 5

Escreva uma função em Python chamada `contador_vogais` que recebe uma string como entrada e retorna o número de vogais (A, E, I, O, U) presentes na string.

Exercício 5 - Resolução

```
def contador_vogais(texto):
```

```
    vogais = "aeiouAEIOU"
```

```
    contador = 0
```

```
    indice = 0
```

```
    while indice < len(texto):
```

```
        if texto[indice] in vogais:
```

```
            contador += 1
```

```
        indice += 1
```

```
    return contador
```

```
# Exemplo de uso da função
```

```
texto = "O 3º período de Sistemas de Informação está aprendendo  
Python"
```

```
resultado = contador_vogais(texto)
```

```
print("O número de vogais na string é:", resultado)
```

Tkinter: Interfaces gráficas em Python

Tkinter é uma biblioteca da linguagem Python que acompanha a instalação padrão e permite desenvolver interfaces gráficas. Isso significa que qualquer computador que tenha o interpretador Python instalado é capaz de criar interfaces gráficas usando o Tkinter, com exceção de algumas distribuições Linux, exigindo que seja feita o download do módulo separadamente

Tkinter: Interfaces gráficas em Python

Um dos motivos de estarmos usando o Tkinter como exemplo é a sua facilidade de uso e recursos disponíveis. Outra vantagem é que é nativo da linguagem Python, tudo o que precisamos fazer é importá-lo no momento do uso, ou seja, estará sempre disponível.

O Tkinter já vem por padrão na maioria das instalações Python, então tudo que temos que fazer é importar a biblioteca.

Importando a biblioteca Tkinter

Para importar todo o conteúdo do módulo usamos o seguinte comando:

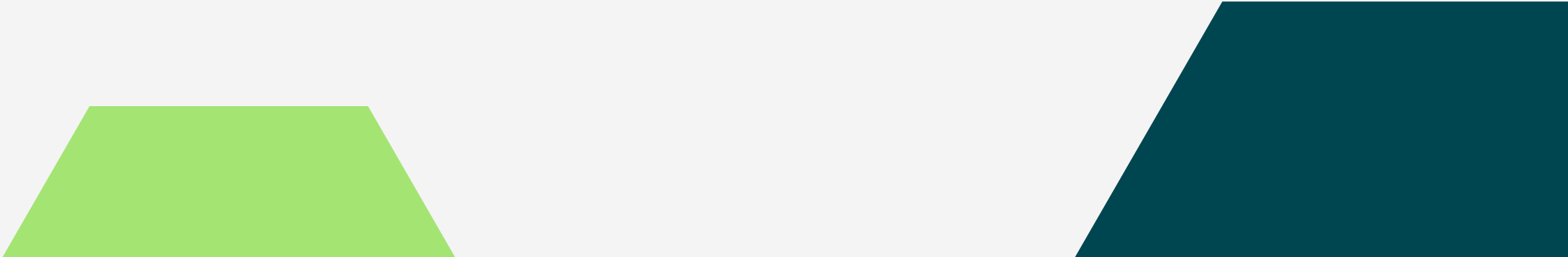
From Tkinter import *

Conceitos de GUI (Graphic User Interface)

- Container – É uma analogia a um container físico e tem como objetivo organizar e guardar objetos. Da mesma forma este conceito serve para um container em interface. Nesse caso, os objetos que estamos armazenando são os widgets;
- Widget – É um componente qualquer na tela, que pode ser um botão, um ícone, uma caixa de texto, etc.;
- Event Handler – São tratadores de eventos. Por exemplo, ao clicarmos em um botão para executar uma ação, uma rotina é executada. Essa rotina é chamada de event handler;
- Event Loop – O event loop verifica constantemente se outro evento foi acionado. Caso a hipótese seja verdadeira, ele irá executar a rotina correspondente.

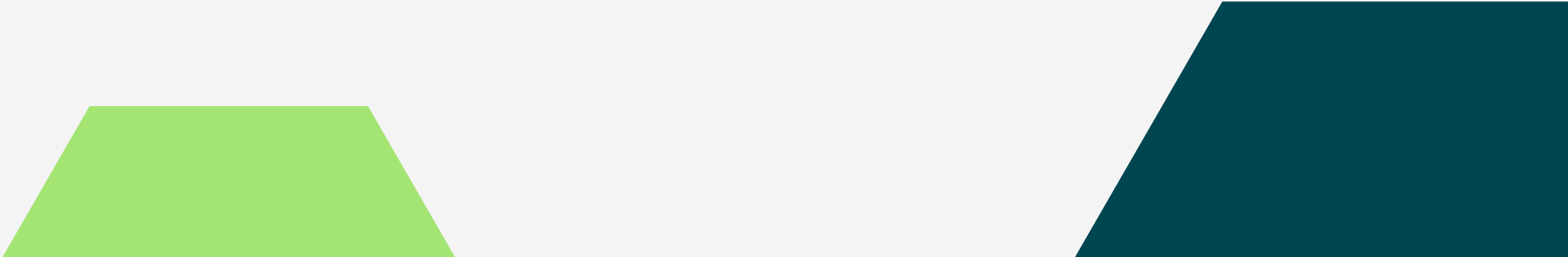
Classes e Métodos

Classes:

- Em Python, as classes são estruturas que permitem criar tipos de dados personalizados.
 - Elas servem como um modelo para criar objetos que compartilham características comuns.
 - As classes são definidas usando a palavra-chave ***class***, seguida pelo nome da classe.
 - Elas encapsulam dados (na forma de atributos) e comportamentos (na forma de métodos) relacionados.
- 

Classes e Métodos

Métodos:

- Os métodos são funções definidas dentro de uma classe que operam nos atributos da classe.
 - Eles representam o comportamento do objeto e são usados para realizar operações específicas.
 - Métodos podem acessar e modificar os atributos da classe.
- 

Montando a Interface

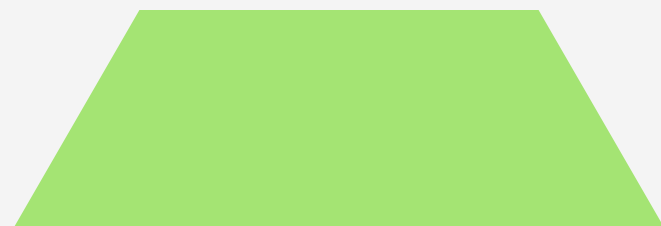
```
class Application:  
    def __init__(self, master=None):  
        pass  
root = Tk()  
Application(root)  
root.mainloop()
```

Entendendo o código...

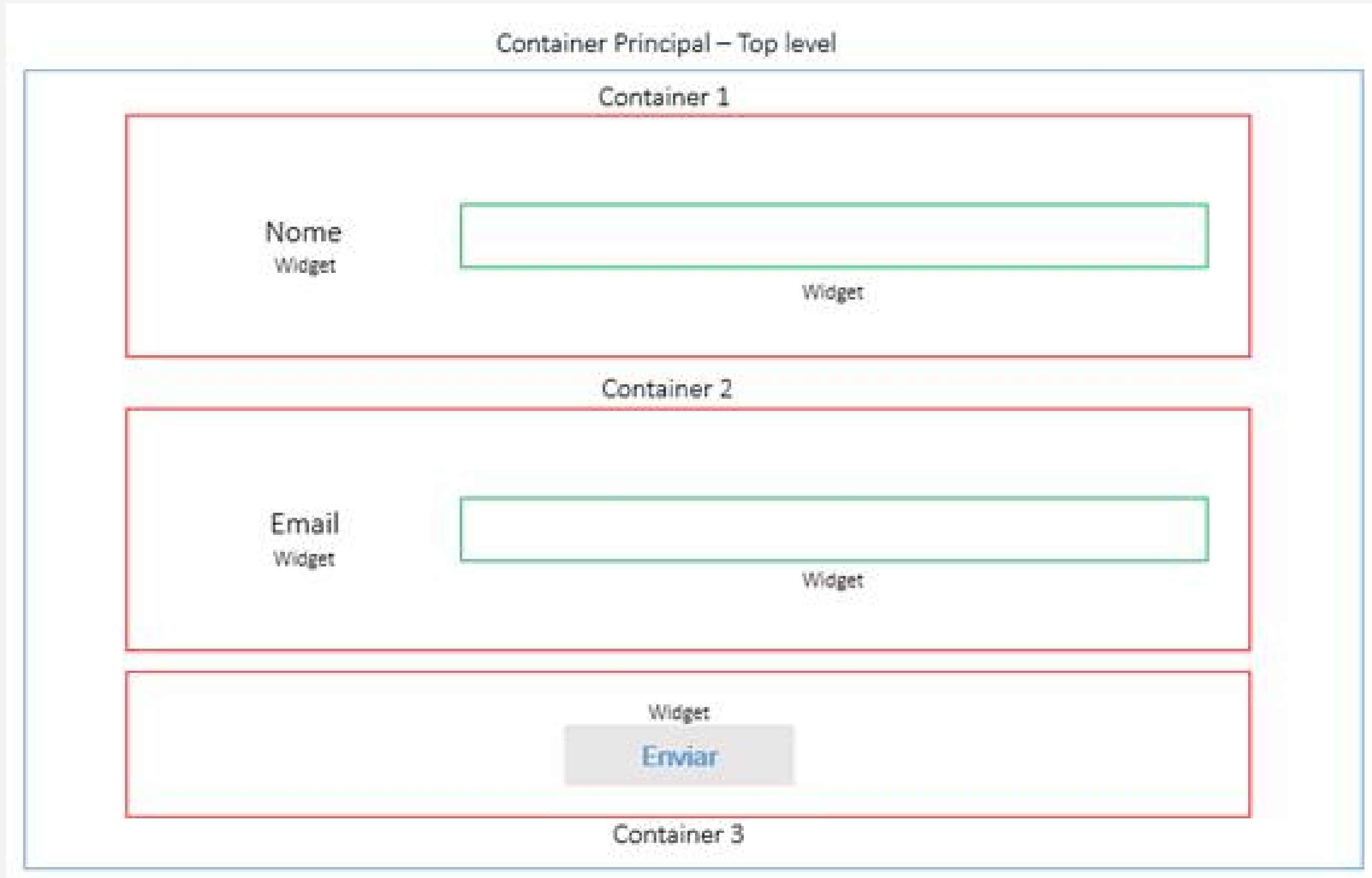
- Criamos uma classe chamada Application
- Este é o método especial **`__init__`**, que é chamado automaticamente quando um objeto da classe é criado. Ele inicializa os atributos do objeto. **`self`** é uma referência ao objeto sendo criado. **`master`** é um parâmetro que representa o widget principal da aplicação.
- Classe TK() através da variável root, que foi criada no final do código. Essa classe permite que os widgets possam ser utilizados na aplicação.
- Em Application(root) passamos a variável root como parâmetro do método construtor da classe Application. E para finalizar, chamamos o método root.mainloop() para exibirmos a tela. Sem o event loop, a interface não será exibida.

Adicionando elementos

- Para poder trabalhar com widgets é necessário entender o conceito de container, que é uma estrutura onde os widgets são colocados. Por questão de organização e para sua correta criação, definimos os containeres, e dentro de cada container, um ou mais widgets que o compõe.



Adicionando elementos



Adicionando elementos

- Sempre que um container for criado dentro de outro, devemos, no momento de sua criação, definir qual é o container pai. A mesma questão de hierarquia serve para a criação de widgets, devendo ser definido na sua criação qual o container pai, ou seja, em que container ele será incluído.

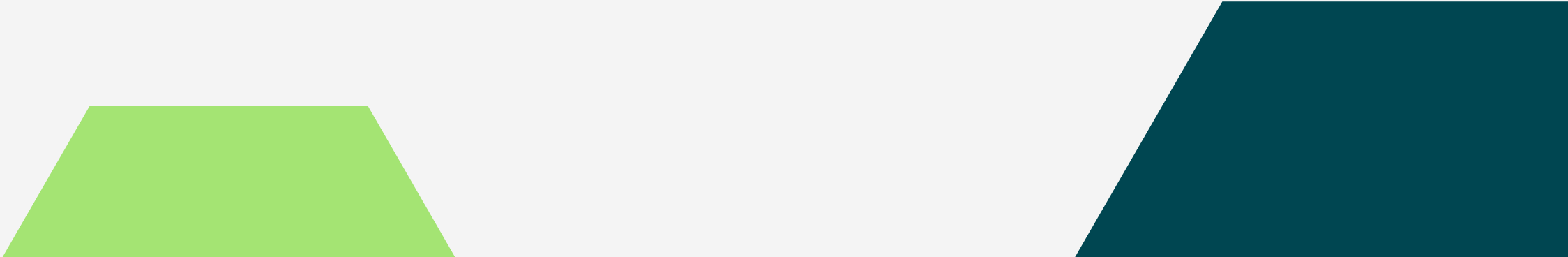
O módulo Tkinter oferece três formas de trabalharmos com geometria e posicionamento:

1. Pack;
2. Grid;
3. Place.

Pack

Em Tkinter, a biblioteca padrão de interface gráfica do Python, `pack()` é um método utilizado para organizar e posicionar widgets dentro de um container, como uma janela ou um frame. Ele é um dos métodos de gerenciamento de geometria disponíveis em Tkinter.

O método **`pack()`** aceita vários argumentos opcionais que controlam o posicionamento e o comportamento do widget, como **`side`** para especificar em qual lado do contêiner o widget deve ser empacotado ("**`top`**", "**`bottom`**", "**`left`**", "**`right`**"), **`fill`** para especificar como o widget deve se expandir para preencher o espaço disponível ("**`x`**", "**`y`**", ou "**`both`**"), e **`expand`** para especificar se o widget deve expandir para preencher qualquer espaço extra disponível no contêiner.



Adicionando elementos

```
from tkinter import *

class Application:
    def __init__(self, master=None):
        self.container1 = Frame(master)
        self.container1.pack()
        self.msg = Label(self.container1, text="ESUP - Sistemas de Informação")
        self.msg.pack ()

root = Tk()
Application(root)
root.mainloop()
```

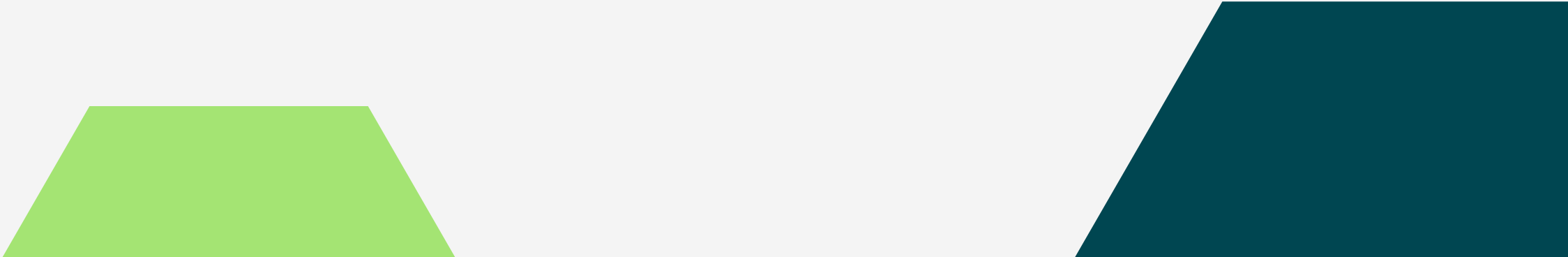
Adicionando elementos

- Na linha 5 criamos o container que irá receber os widgets e passamos como referência o container pai.
 - O frame master que é o elemento máximo da hierarquia, ou seja, é a nossa janela de aplicação, que contém o título, e botões de maximizar, minimizar e fechar.
- Na linha 5 informamos o gerenciador de geometria pack e usamos um widget label para imprimir na tela as palavras "ESUP - Sistemas de informação" e informamos que o container pai é o container1

Atribuindo valores a elementos da tela

```
1  from tkinter import *
2
3  class Application:
4      def __init__(self, master=None):
5          self.container1 = Frame(master)
6          self.container1.pack()
7          self.msg = Label(self.container1, text="ESUP - Sistemas de Informação")
8          self.msg["font"] = ("Verdana", "10", "italic", "bold")
9          self.msg.pack ()
10         self.sair = Button(self.container1)
11         self.sair["text"] = "Sair"
12         self.sair["font"] = ("Calibri", "10")
13         self.sair["width"] = 5
14         self.sair["command"] = self.container1.quit
15         self.sair.pack ()
16
17 root = Tk()
18 Application(root)
19 root.mainloop()
```

Atribuindo valores a elementos da tela

- **Width** – Largura do widget;
 - **Height** – Altura do widget;
 - **Text** – Texto a ser exibido no widget;
 - **Font** – Família da fonte do texto;
 - **Fg** – Cor do texto do widget;
 - **Bg** – Cor de fundo do widget;
 - **Side** – Define em que lado o widget se posicionará (Left, Right, Top, Bottom).
- 

Event handler

Os event handlers são ações que executadas como resposta a determinado evento. Sempre que um evento ocorre, o event loop o interpreta como uma string.

```
self.sair.bind("<Button-1>", self.mudarCurso)
```

Event handler

Os event handlers são ações que executadas como resposta a determinado evento. Sempre que um evento ocorre, o event loop o interpreta como uma string.

```
self.sair.bind("<Button-1>", self.mudarCurso)
```

<Button-1> se refere ao evento de clicar no botão esquerdo do mouse.

```
1  from tkinter import *
2
3  class Application:
4      def __init__(self, master=None):
5          self.container1 = Frame(master)
6          self.container1.pack()
7          self.msg = Label(self.container1, text="ESUP - Sistemas de Informação")
8          self.msg["font"] = ("Verdana", "10", "italic", "bold")
9          self.msg.pack ()
10         self.sair = Button(self.container1)
11         self.sair["text"] = "Clique"
12         self.sair["font"] = ("Calibri", "10")
13         self.sair["width"] = 5
14         self.sair.bind("<Button-1>", self.mudarCurso)
15         self.sair.pack ()
16
17         def mudarCurso(self, event):
18             if self.msg["text"] == "ESUP - Sistemas de Informação":
19                 self.msg["text"] = "ESUP - Ciências Contábeis"
20             else:
21                 self.msg["text"] = "ESUP - Sistemas de Informação"
22
23 root = Tk()
24 Application(root)
25 root.mainloop()
```

Event handler

Quando executamos este código e o botão que foi criado recebe um clique, o texto é modificado na tela, graças ao event handler associado ao **bind**

Perceba que quando criamos o método mudarCurso() adicionamos dois parâmetros: o self e o event. Nesse caso eles são obrigatórios para o funcionamento e devem ser sempre os dois primeiros parâmetros do método.

Recebendo dados dos usuários

Utilizamos o **Entry** para receber os dados dos usuários onde os mesmos são capturados como string, de forma semelhante ao método *input*.



Recebendo dados dos usuários

```
from tkinter import *

class Application:
    def __init__(self, master=None):
        self.container1 = Frame(master)
        self.container1["pady"]=20
        self.container1.pack()
        self.msg = Label(self.container1, text="Dados do Aluno")
        self.msg["font"] = ("Verdana", "10", "italic", "bold")
        self.msg.pack ()

        self.containerTexto = Frame(master)
        self.containerTexto.pack()
        self.msg = Label (self.containerTexto,text="Inserir seus dados:")
        self.msg.pack (side=LEFT)

        self.nomeAluno = Entry(self.containerTexto)
        self.nomeAluno.pack(side=RIGHT)
```

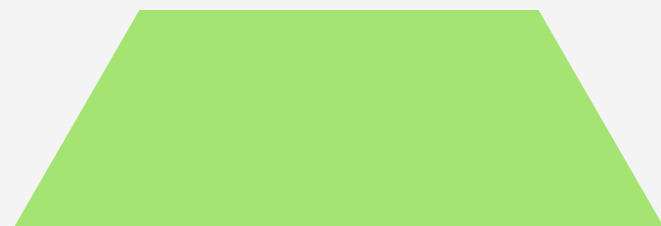

Recebendo dados dos usuários

```
21     self.container2 = Frame(master)
22     self.container2["pady"] = 30
23     self.container2.pack()
24     self.botaoMensagem = Button(self.container2)
25     self.botaoMensagem["text"] = "Exibir Mensagem"
26     self.botaoMensagem["width"] = 12
27     self.botaoMensagem.bind("<Button-1>",self.MensagemBoasVindas)
28     self.botaoMensagem.pack()
29
30     self.ExibirMsg = Label(self.container2,text="")
31     self.ExibirMsg.pack()
32
33     def MensagemBoasVindas(self,event):
34         aluno = self.nomeAluno.get()
35         self.ExibirMsg["text"] = aluno
36
37
38 root = Tk()
39 Application(root)
40 root.mainloop()
41
```

Para lembrar....

Como podemos mostrar a mensagem misturando strings e variáveis no widget de texto?

Ex.: Bem vindo, Giovana.



Método geometry()

O método geometry é usado para definir a geometria (tamanho e posição) da janela Tkinter. Ele aceita uma string no formato larguraxaltura+posicao_x+posicao_y, onde largura é a largura da janela, altura é a altura da janela, posicao_x é a posição horizontal da janela na tela e posicao_y é a posição vertical da janela na tela. Por exemplo, `root.geometry("500x300+100+50")` define uma janela com largura de 500 pixels, altura de 300 pixels, posicionada 100 pixels à direita e 50 pixels abaixo do canto superior esquerdo da tela.

Método geometry()

Exemplo:

```
root = tk.Tk()  
root.geometry("400x400+250+250")  
app = Application(root)  
root.mainloop()
```

Método title()

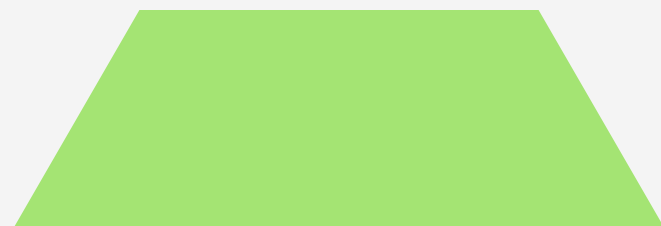
title: O método title é usado para definir o título da janela Tkinter. Ele aceita uma string que será exibida como o título da janela. Por exemplo, `root.title("Minha Janela")` define o título da janela como "Minha Janela".

Método title()

```
root = tk.Tk()  
root.geometry("400x400+250+250")  
root.title("Minha aplicação")  
app = Application(root)  
root.mainloop()
```

Método `configure`

Em Tkinter, o método **`configure()`** é uma forma de alterar dinamicamente as opções de configuração de um widget após ele ter sido criado. Ele permite modificar várias propriedades de um widget, como cor, fonte, texto, tamanho, estado, entre outros, mesmo após a sua inicialização.



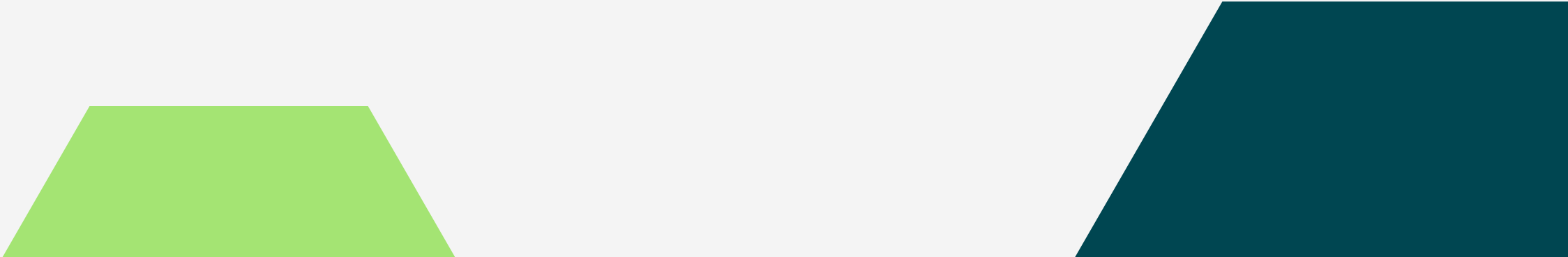
Método configure

A sintaxe geral do método configure() é a seguinte:

widget.configure(opção1=valor1, opção2=valor2, ...)

Método configure

Aqui está uma explicação detalhada:

- **widget:** É o objeto do widget ao qual você deseja aplicar as configurações.
 - **opção:** Refere-se à propriedade específica do widget que você deseja modificar.
 - **valor:** É o novo valor que você deseja atribuir à opção do widget.
- 

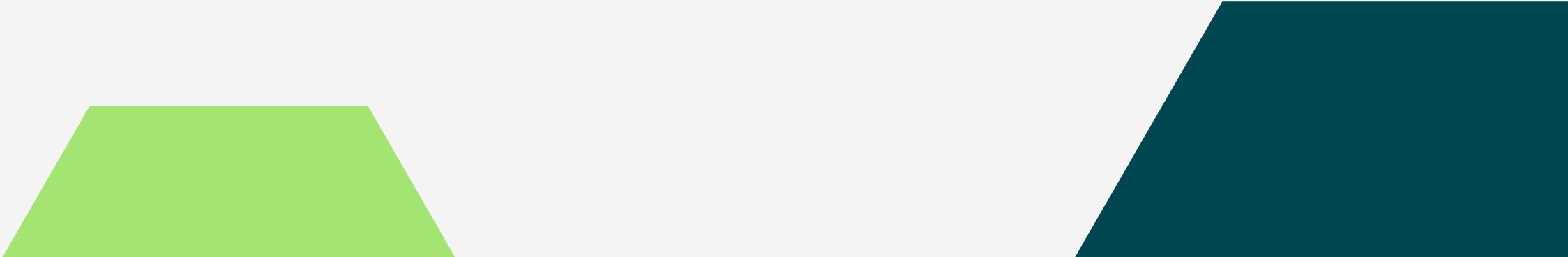
Método configure

Por exemplo, se você tiver um botão em sua interface gráfica e deseja alterar a cor de fundo para vermelho, você pode usar o método `configure()` assim:

```
botao.configure(bg='red')
```

Método configure

O método `configure()` é útil quando você precisa fazer alterações dinâmicas na aparência ou no comportamento dos widgets em sua aplicação Tkinter em resposta a eventos do usuário ou a mudanças de estado do programa. Ele oferece flexibilidade para ajustar as configurações dos widgets conforme necessário durante a execução do programa.



Classe: `ttk.Style()`

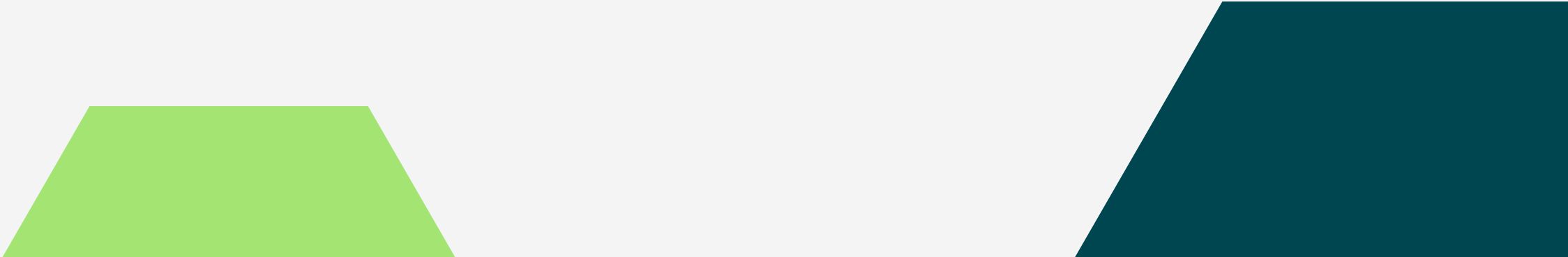
A classe `ttk.Style()` é uma parte essencial da biblioteca Tkinter que permite personalizar a aparência dos widgets (componentes gráficos de uma interface de usuário) de forma mais avançada do que com a classe `tkinter`. É especialmente útil para criar interfaces gráficas mais modernas e estilizadas.

Classe: `ttk.Style()`

1. **Estilos de widgets:** Com a `ttk.Style()`, você pode definir estilos para diferentes tipos de widgets, como botões, rótulos, entradas de texto, barras de progresso, entre outros. Isso inclui propriedades como cor, fonte, borda, tamanho e preenchimento.

Classe: `ttk.Style()`

2. **Temas:** Tkinter oferece uma variedade de temas pré-construídos que você pode aplicar aos seus widgets. Esses temas mudam a aparência padrão dos widgets para algo mais moderno e estilizado. Alguns exemplos de temas incluem 'clam', 'default', 'alt' e 'vista'. Você pode aplicar um tema usando o método `theme_use()` da classe `ttk.Style()`.



Classe: `ttk.Style()`

3. Personalização avançada: Além dos temas pré-construídos, você pode criar estilos personalizados para os widgets. Isso permite uma personalização mais detalhada, onde você pode definir propriedades específicas para cada widget. Por exemplo, você pode criar um estilo para botões com uma cor de fundo específica e uma fonte diferente.

Classe: `ttk.Style()`. Como utilizar?

Antes de começar, você precisa importar os módulos `tkinter` e `ttk`:

```
from tkinter import ttk
```


Classe: `ttk.Style()`. Como utilizar?

Crie um objeto `ttk.Style()` que será usado para configurar os estilos dos widgets:

```
estilo = ttk.Style()
```

Configurar os estilos:

Você pode configurar os estilos dos widgets usando métodos específicos da classe `ttk.Style()`. Por exemplo, para alterar a cor de fundo de todos os botões para vermelho, você pode usar o método `configure()`:

```
estilo.configure('TButton', foreground='red')
```



Classe: `ttk.Style()`. Como utilizar?

Aplicando os estilos:

```
botao = ttk.Button(janela, text='Clique Aqui')  
botao['style'] = 'TButton'
```

Classe: `ttk.Style()`. Aplicando temas

Além de definir estilos personalizados, você também pode aplicar temas pré-definidos a todos os widgets em sua aplicação. Por exemplo, para aplicar o tema 'clam' à sua aplicação, você pode fazer o seguinte:

```
estilo.theme_use('alt')
```

Propriedades do ttk.Style

1. foreground (ou fg): Define a cor do texto do widget.
2. background (ou bg): Define a cor de fundo do widget.
3. font: Define a fonte do texto do widget.
4. bordercolor (ou bd): Define a cor da borda do widget.
5. borderwidth (ou bw): Define a largura da borda do widget.
6. padding: Define o preenchimento interno do widget.
7. relief: Define o estilo da borda do widget (por exemplo, 'flat', 'raised', 'sunken', 'ridge', 'groove').

Propriedades do ttk.Style

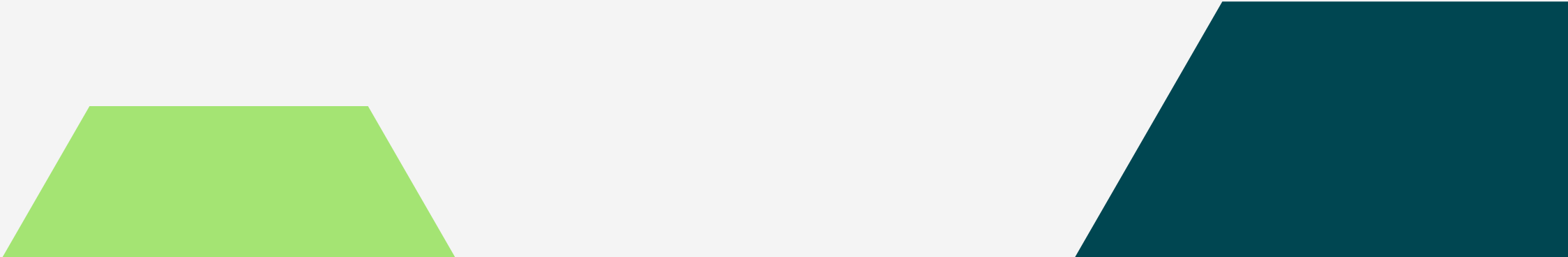
1. `width`: Define a largura do widget.
2. `height`: Define a altura do widget.
3. `highlightcolor`: Define a cor do contorno do widget quando ele está em foco.
4. `highlightthickness`: Define a largura do contorno do widget quando ele está em foco.
5. `troughcolor`: Define a cor da calha (área entre as extremidades de uma barra de rolagem) para barras de rolagem.

Propriedades do ttk.Style

1. `arrowcolor`: Define a cor da seta para widgets que possuem setas (por exemplo, barras de rolagem).
2. `arrowpadding`: Define o preenchimento interno para a área da seta.
3. `arrowshape`: Define a forma da seta para widgets que possuem setas (por exemplo, barras de rolagem).

Bibliotecas em Python

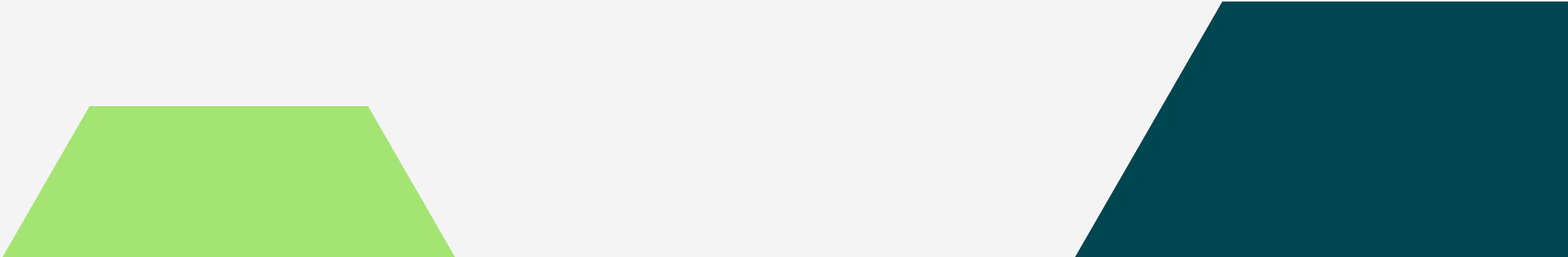
Em Python, uma biblioteca (também conhecida como módulo ou pacote) é um conjunto de funcionalidades e código pré-escrito que pode ser reutilizado em diferentes projetos. As bibliotecas fornecem uma variedade de recursos, desde operações básicas até funcionalidades avançadas, permitindo que os desenvolvedores economizem tempo e esforço ao implementar soluções para problemas comuns.



Bibliotecas em Python

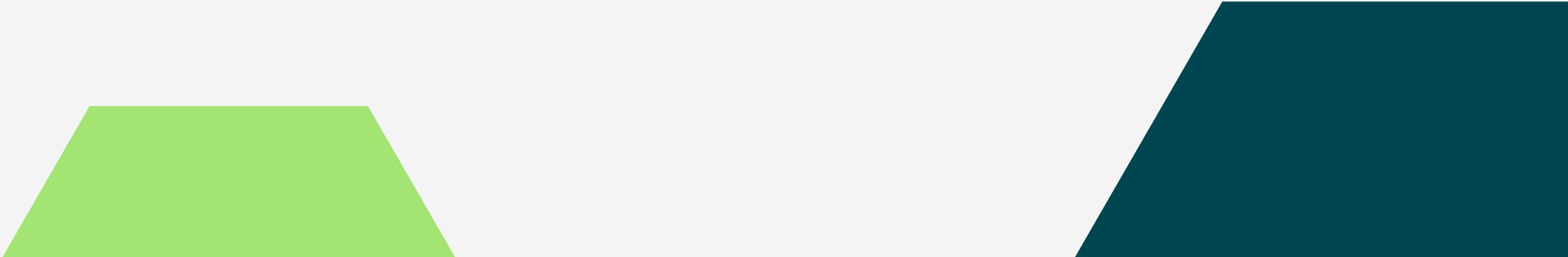
Existem dois tipos:

Bibliotecas padrão: Essas são bibliotecas que vêm pré-instaladas com a instalação padrão do Python. Elas fornecem funcionalidades básicas para tarefas como manipulação de strings, operações de arquivo, processamento de dados, matemática, acesso à rede e muito mais. Exemplos incluem os, sys, math, datetime, random, json, entre outras.



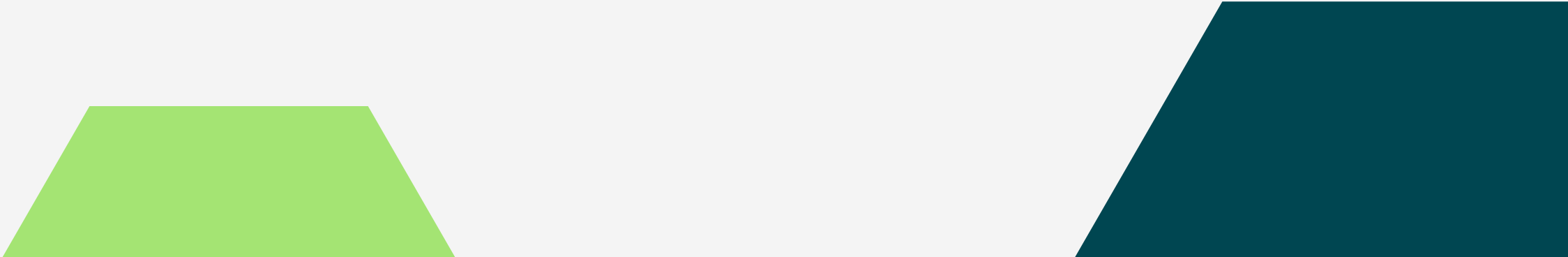
Bibliotecas em Python

Bibliotecas de terceiros: Estas são bibliotecas desenvolvidas por terceiros e disponibilizadas para uso público. Elas geralmente fornecem funcionalidades mais especializadas e podem abordar uma ampla gama de áreas, desde desenvolvimento web e científico até aprendizado de máquina e visualização de dados. Alguns exemplos populares incluem numpy, pandas, matplotlib, requests, Django, Flask, TensorFlow, PyTorch, entre outras.



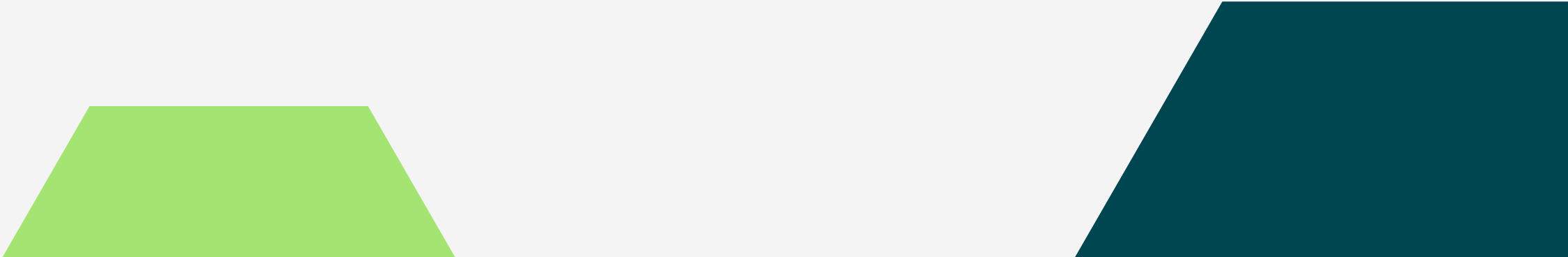
Bibliotecas populares

CSV (`csv`): A biblioteca `csv` em Python fornece funcionalidades para ler e escrever arquivos CSV (Comma-Separated Values). Ela permite que você processe facilmente dados em formato CSV, que é comumente usado para armazenar e trocar dados tabulares. A biblioteca `csv` oferece maneiras simples e eficientes de trabalhar com arquivos CSV em Python.



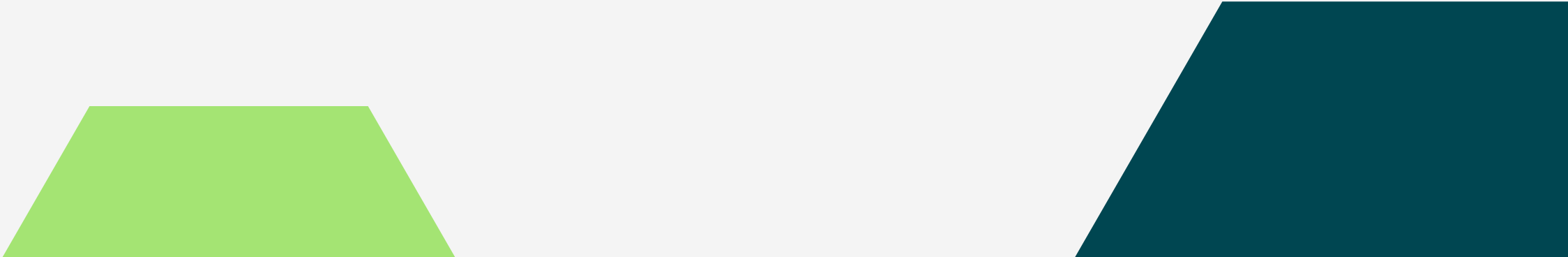
Bibliotecas populares

Random (random): A biblioteca random em Python fornece funcionalidades para gerar números aleatórios e realizar operações relacionadas à aleatoriedade. Ela inclui funções para gerar números inteiros aleatórios, números de ponto flutuante, escolher elementos aleatórios de uma lista, embaralhar sequências e muito mais. A biblioteca random é útil em uma variedade de contextos, incluindo simulações, jogos e criptografia.



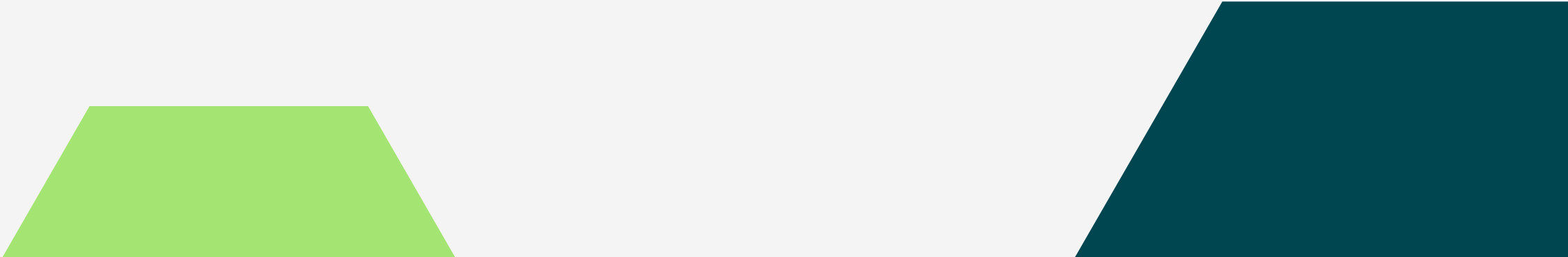
Bibliotecas populares

Requests (requests): Como mencionado anteriormente, a biblioteca requests em Python facilita o envio de solicitações HTTP e o processamento de respostas. Ela é amplamente usada para interagir com APIs da web, fazer raspagem de dados na web e realizar comunicação com serviços web. A requests oferece uma interface simples e intuitiva para trabalhar com solicitações e respostas HTTP em Python.



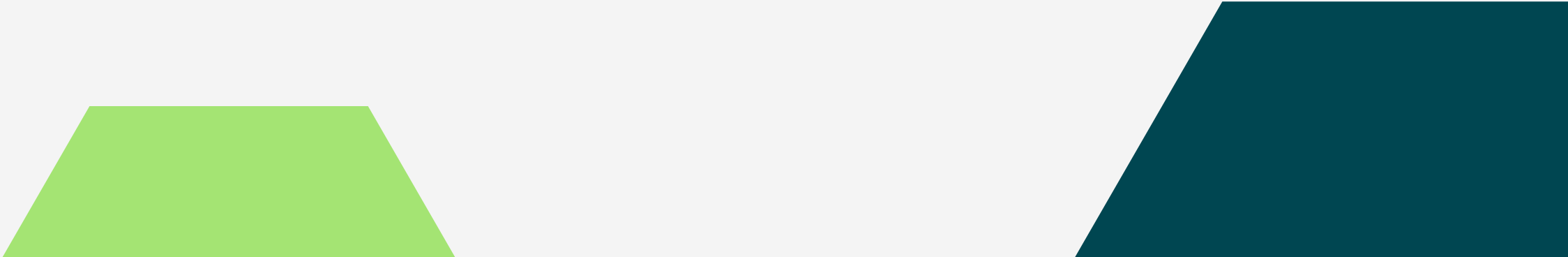
Bibliotecas populares

Beautiful Soup (beautifulsoup4): Beautiful Soup é uma biblioteca em Python para analisar documentos HTML e XML e extrair informações úteis deles. Ela fornece maneiras convenientes de navegar na estrutura do documento, buscar e manipular elementos HTML/XML, e extrair dados estruturados. Beautiful Soup é frequentemente usada para fazer web scraping e análise de documentos HTML/XML.



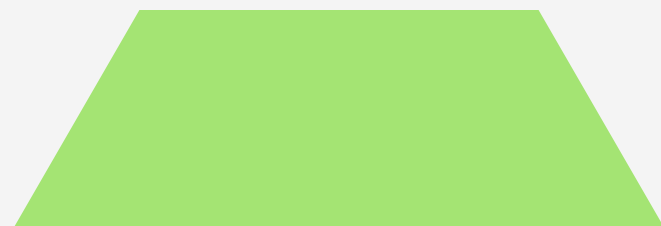
Bibliotecas populares

TensorFlow (tensorflow): TensorFlow é uma biblioteca de código aberto para aprendizado de máquina e inteligência artificial desenvolvida pelo Google. Ela oferece uma ampla gama de ferramentas e recursos para construir e treinar modelos de aprendizado de máquina, incluindo redes neurais profundas, redes neurais convolucionais, redes neurais recorrentes e muito mais. TensorFlow é uma das bibliotecas mais populares para desenvolvimento de modelos de aprendizado de máquina em Python.



Ambientes virtuais

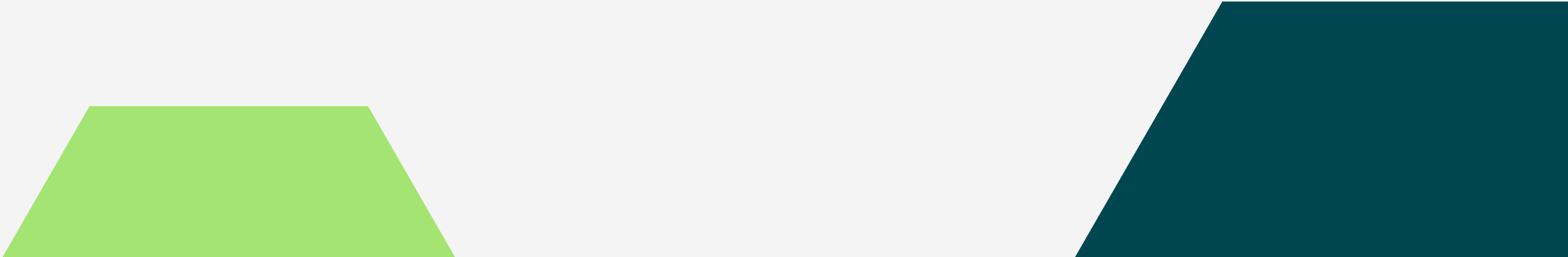
Ambientes virtuais em Python são uma ferramenta crucial para o desenvolvimento de software. Eles fornecem um meio de isolar as dependências de um projeto, permitindo que você tenha diferentes conjuntos de bibliotecas e versões instaladas em cada ambiente. Isso é especialmente útil quando você trabalha em múltiplos projetos Python ou quando precisa garantir a compatibilidade de versões de bibliotecas específicas



Ambientes virtuais e instalando bibliotecas

Para criar um ambiente virtual em Python, você pode usar a ferramenta integrada venv. Aqui estão os passos para criar um ambiente virtual:

Navegue até o diretório do seu projeto: Abra o terminal ou prompt de comando e navegue até o diretório onde você deseja criar o ambiente virtual.



Ambientes virtuais e instalando bibliotecas

Crie o ambiente virtual: No terminal, execute o seguinte comando, substituindo nome_do_ambiente_virtual pelo nome que deseja dar ao seu ambiente virtual:

```
python3 -m venv nome_do_ambiente_virtual
```

ou

```
py -3 -m venv nome_ambiente
```



Ambientes virtuais e instalando bibliotecas

Ative o ambiente virtual: Após criar o ambiente virtual, você precisa ativá-lo. No terminal, use o seguinte comando:

Windows: `nome_do_ambiente_virtual\Scripts\activate`

Mac: `source nome_do_ambiente_virtual/bin/activate`



Ambientes virtuais e instalando bibliotecas

Instale bibliotecas: Com o ambiente virtual ativado, você pode instalar as bibliotecas necessárias usando o pip. Por exemplo:

pip install nome_da_biblioteca



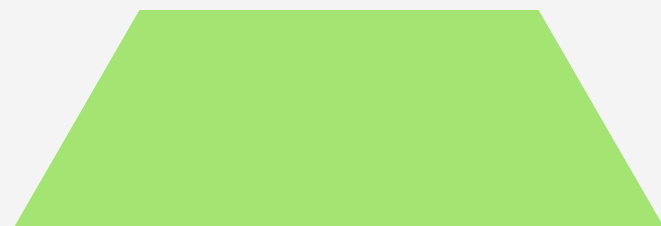
Biblioteca Turtle

A biblioteca Turtle é uma ferramenta de desenho gráfico que faz parte da biblioteca padrão do Python. Ela é usada para criar imagens gráficas por meio de programação, especialmente útil para aprender programação para iniciantes, pois permite visualizar os resultados do código de maneira interativa.



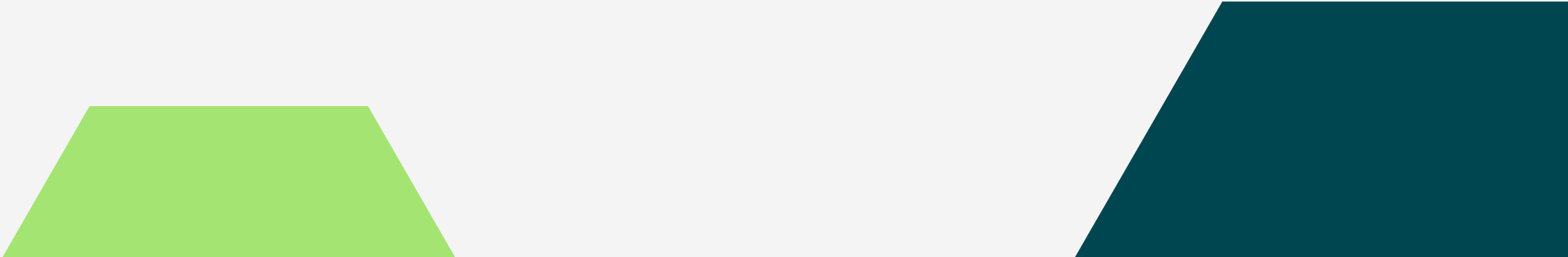
Biblioteca Turtle

A biblioteca Turtle é inspirada na linguagem de programação Logo, que foi desenvolvida para fins educacionais nos anos 60. O nome "Turtle" vem da ideia de uma tartaruga que se move em uma tela e deixa um rastro conforme se move.



Biblioteca Turtle - EXEMPLO

```
import turtle  
tela = turtle.Screen()  
tartaruga = turtle.Turtle()  
  
for _ in range(4):  
    tartaruga.forward(100)  
    tartaruga.left(90)  
  
tela.mainloop()
```



Biblioteca Turtle - EXEMPLO

Podemos usar o shape para indicar o formato que a tartaruga deve ter:

```
import turtle
```

```
# Possíveis formatos
```

```
# -> arrow (seta)
```

```
# -> blank (invisível)
```

```
# -> circle (círculo)
```

```
# -> classic (clássica)
```

```
# -> square (quadrado)
```

```
# -> triangle (triângulo)
```

```
# -> turtle (tartaruga)
```

```
tartaruga = turtle.Turtle()
```

```
tartaruga.shape("turtle")
```



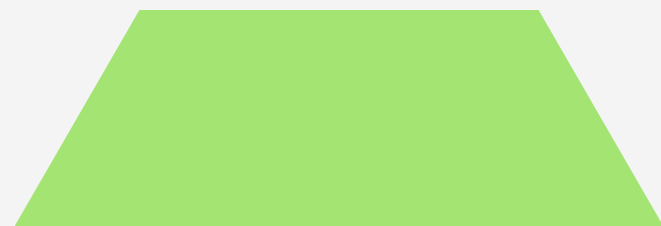
Biblioteca Turtle - EXEMPLO

Podemos usar o color para indicar a cor que a tartaruga deve ter:

Ex.: `tartaruga.color("blue")`

Biblioteca Python Imaging Library (PIL)

A biblioteca Python Imaging Library (PIL), agora conhecida como Pillow, é uma poderosa biblioteca para processamento de imagens em Python. Ela fornece uma ampla gama de funcionalidades para abrir, manipular e salvar imagens em vários formatos de arquivo.



Biblioteca Python Imaging Library (PIL)

1. Abrir e Salvar Imagens: A PIL/Pillow suporta vários formatos de arquivo de imagem, como JPEG, PNG, BMP, GIF, TIFF e muitos outros. Você pode abrir uma imagem usando a função `open()` e salvar uma imagem usando o método `save()`.
2. Manipulação de Imagens: A biblioteca oferece uma variedade de métodos para manipular imagens, como redimensionamento, rotação, recorte, conversão de cores, ajuste de brilho, contraste e nitidez, entre outros.
3. Processamento de Imagens: Você pode realizar várias operações de processamento de imagens, como filtragem, suavização, detecção de bordas e segmentação.

Biblioteca Python Imaging Library (PIL)

1. Criação de Imagens: Além de manipular imagens existentes, a PIL/Pillow permite criar novas imagens de forma programática, preenchendo-as com cores sólidas, desenhando formas geométricas ou textos, e combinando várias imagens em uma única imagem.
2. Conversão de Formato de Imagem: Você pode converter imagens de um formato de arquivo para outro facilmente usando os métodos fornecidos pela PIL/Pillow.
3. Processamento de Imagens em Lote: A biblioteca facilita o processamento em lote de várias imagens, permitindo automatizar tarefas repetitivas em um conjunto de imagens.

Renderizando uma imagem

```
from PIL import Image, ImageTk
```

```
imagem = Image.open("caminho_para_sua_imagem.png")
```

```
imagem.thumbnail((200, 200))
```

```
imagem = ImageTk.PhotoImage(imagem)
```

ATENÇÃO

Essa biblioteca não é nativa, ou seja, precisamos fazer a instalação.

No terminal digite:

pip install Pillow

PARA AMPLIAR SEU CONHECIMENTO

PESQUISE SOBRE AS BIBLIOTECAS PYTHON E SUAS APLICAÇÕES.

