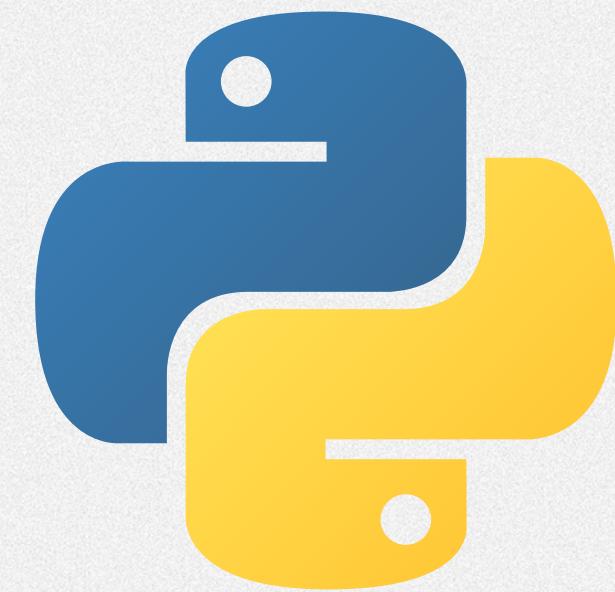




PUC Minas



Python

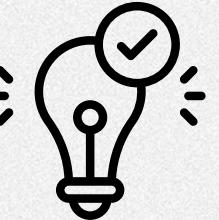
Bruno Rafael Santos Oliveira, Letícia da Silva Rocha, Matheus Eduardo
Campos Soares, Rayssa Mell de Souza Silva, Thiago Pereira de Oliveira

índice

- 01 INTRODUÇÃO**
- 02 HISTÓRICO DA LINGUAGEM**
- 03 PARADIGMAS**
- 04 CARACTERÍSTICAS MARCANTES**
- 05 APLICAÇÕES**
- 06 LINGUAGENS RELACIONADAS**
- 07 EXEMPLOS**
- 08 TUTORIAL DE INSTALAÇÃO**
- 09 PARTE PRÁTICA**
- 10 CONSIDERAÇÕES FINAIS**
- APÊNDICES E BIBLIOGRAFIA**



1. Introdução

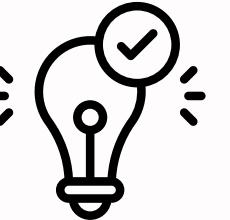


O que é Python?

- **Contextualização:** O Python é uma linguagem de **programação de alto nível**, ou seja, ela foi feita pra ser mais próxima da linguagem humana do que da linguagem de máquina. Além de auto nível ele é uma **linguagem interpretada e multiparadigmática**.
- **Criador:** O Python foi criado por um programador holandês chamado **Guido van Rossum**, no início dos anos 90, mais precisamente, em **1991**.
- **Origem do Nome:** O nome Python, inclusive, não vem da cobra, mas sim do grupo de comédia britânico '**Monty Python**', que o Guido gostava. Ele queria um nome curto e único.
- **Filosofia:** O Python carrega uma filosofia de desenvolvimento muito **clara e intencional**. Essa filosofia está sintetizada em um documento poético e inspirador chamado '**The Zen of Python**'.



1. Introdução



Paradigmas

Python é uma linguagem multiparadigma, e isso é um dos motivos pelos quais ela é tão amada. Isso significa que você pode programar de diferentes formas nela:

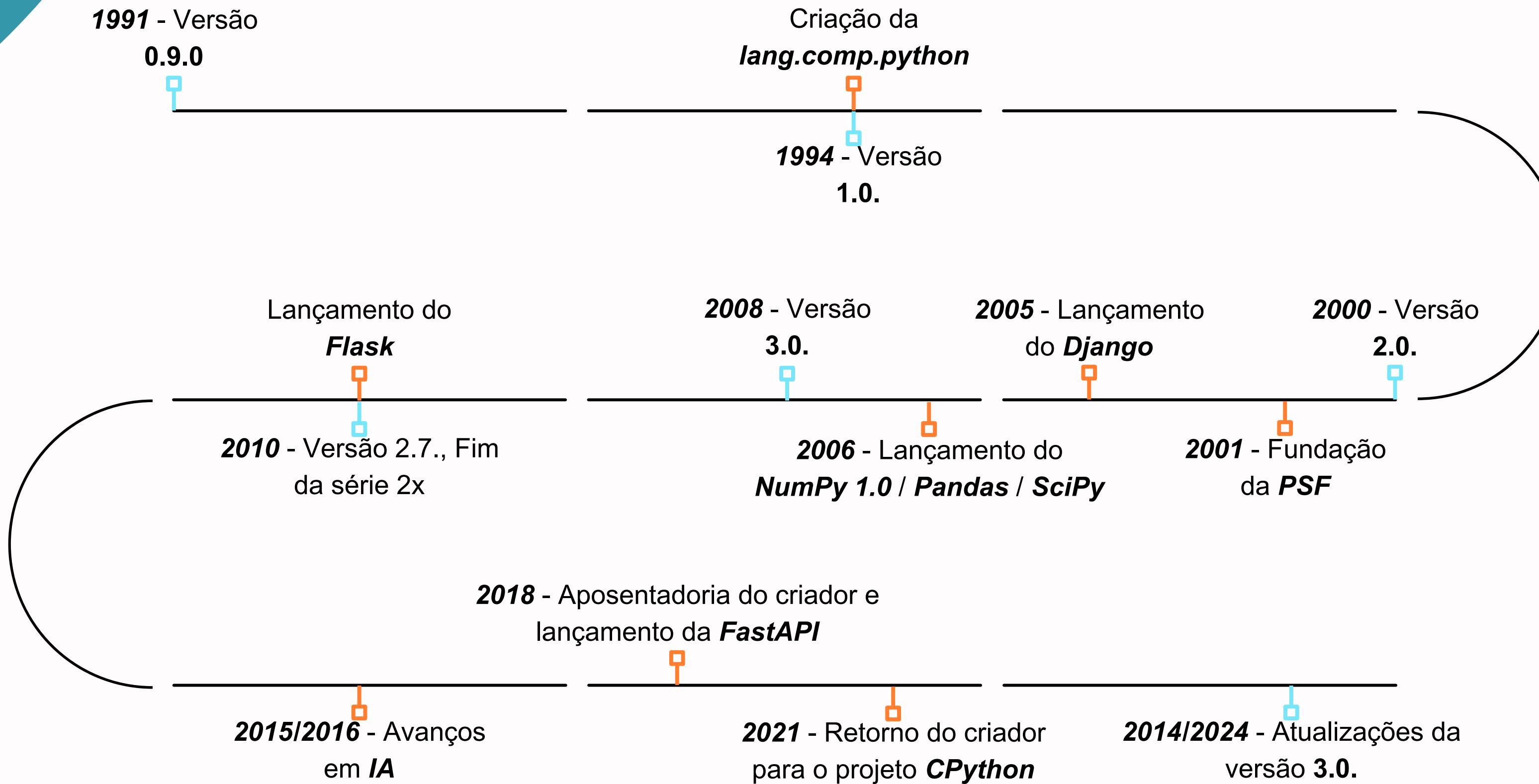
- **Paradigma imperativo**
- **Paradigma orientado a objetos**
- **Paradigma funcional**

Python no Ensino

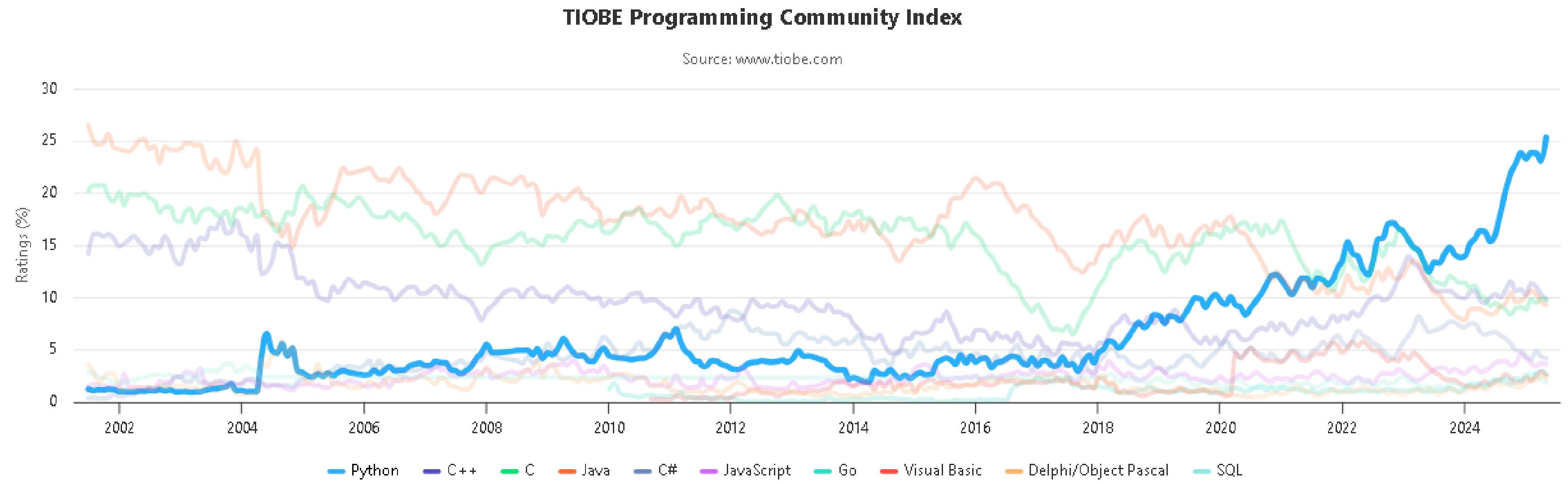
Essa **flexibilidade** torna o Python uma linguagem extremamente versátil, que se adapta a vários estilos e tipos de projetos. Inclusive o Python tem papel em diversos países como linguagem de programação a ser ensinada já no **ensino base**, como no **Reino Unido, Estônia, Nova Zelândia, Israel, Coreia do Sul, Portugal e Brasil**.



2. Histórico



2. Histórico



2. Histórico



3. Paradigmas

Python é uma linguagem multiparadigma, ou seja, permite programar de diferentes formas de acordo com a necessidade do projeto. Essa flexibilidade torna a linguagem mais versátil e poderosa para resolver variados tipos de problemas.

- **Paradigma Procedural**
- **Paradigma Orientado a Objetos (POO)**
- **Paradigma Funcional**

O desenvolvedor pode escolher o estilo mais adequado ao problema, ou até combinar paradigmas no mesmo projeto.

3. Paradigma Procedural

No paradigma procedural, séries de etapas computacionais são divididas em módulos, o que significa que o código é agrupado em funções e executado em série. Basicamente, ele combina o código em série para instruir um computador a executar uma determinada tarefa a cada etapa. Esse paradigma auxilia na modularidade do código, que geralmente é feita pela implementação funcional. Esse paradigma de programação facilita a organização de itens relacionados sem dificuldades, de modo que cada arquivo atua como um contêiner.

```
1 # Forma procedural de encontrar a soma de uma lista
2
3 minhaLista = [10, 20, 30, 40]
4
5 # Modularização usando função
6
7 def soma_de_lista(minhaLista):
8     res = 0 # Inicializa a variável acumuladora com zero
9     for val in minhaLista: # Percorre cada elemento da lista
10        res += val # Soma o valor atual ao acumulador
11    return res # Retorna o resultado final da soma
12
13 # Chamada da função e exibição do resultado
14 print(soma_de_lista(minhaLista)) # Saída esperada: 100|
```

3. Paradigma POO

No paradigma de programação orientada a objetos, os objetos são o elemento-chave dos paradigmas. Objetos podem ser simplesmente definidos como a instância de uma classe que contém tanto os membros de dados quanto as funções de métodos. Além disso, o estilo orientado a objetos relaciona membros de dados e funções de métodos que suportam encapsulamento e, com a ajuda do conceito de herança, o código pode ser facilmente reutilizável.

```
1 - class Pessoa:                      # Classe-base
2 -     def __init__(self, nome):
3 -         self._nome = nome            # _nome: atributo encapsulado (conv. "protegido")
4 -
5 -     def falar(self):
6 -         print(f"Olá, eu sou {self._nome}")
7 -
8 - class Aluno(Pessoa):                 # Herda de Pessoa
9 -     def falar(self):                  # Polimorfismo: novo comportamento
10 -        print(f"Sou {self._nome} e estou estudando Python!")
11 -
12 - # Demonstração rápida
13 - for p in (Pessoa("João"), Aluno("Ana")):
14 -     p.falar()
```

3. Paradigma Funcional

Paradigmas de programação funcional se baseiam em funções matemáticas puras e são considerados declarativos, pois descrevem o que fazer, não como fazer. Cada instrução é uma expressão que retorna um valor, sem estados mutáveis ou comandos sequenciais. Recursos como funções lambda, recursão e funções de ordem superior são comuns. Ao tratar funções como valores (passar e retornar funções), o código se torna mais expressivo, modular e legível.

```
1 # Forma funcional de encontrar a soma de uma lista
2 import functools
3
4 minhaLista = [10, 20, 30, 40]
5
6 # Recursão
7 def soma_de_lista(minhaLista):
8     if len(minhaLista) == 1:
9         return minhaLista[0]
10    else:
11        return minhaLista[0] + soma_de_lista(minhaLista[1:])
12
13 print(functools.reduce(lambda x, y: x + y, minhaLista))
```

4. Características Marcantes

SIMPLICIDADE E LEGIBILIDADE

A sintaxe do Python é projetada para ser clara e concisa, o que facilita a compreensão e o desenvolvimento de código. O Python utiliza indentação em vez de chaves para delimitar blocos de código, o que torna o código mais legível e evita erros comuns de formatação. Já o ponto e vírgula é opcional.

```
1 def fibonacci(n):
2     if n <= 1:
3         return n
4     return fibonacci(n - 1) + fibonacci(n - 2)
5
6
7
8 for i in range(5):
9     print(fibonacci(i))
```

```
1 v def fibonacci(n){
2     if n <= 1:
3         return n
4     return fibonacci(n - 1) + fibonacci(n - 2)
5 }
```

Console

① Executar SyntaxError: expected ':' (<exec>, line 1)

4. Características Marcantes

TIPAGEM DINÂMICA E FORTE

Python é fortemente tipado, ou seja, não permite operações entre tipos incompatíveis. Ao mesmo tempo, é dinamicamente tipado, com tipo definido automaticamente quando um valor é atribuído à uma variável. Como as variáveis não têm tipo fixo, também não há verificação de tipos nos parâmetros das funções. A verificação pode ser feita por meio de funções como: `type()` ou `isinstance()`.

```
1 a = "texto"
2 print(type(a))
```

Console

Executar <class 'str'>

```
1 a = "texto"
2 print(a+3)
3 |
```

Console

① Executar `TypeError: can only concatenate str (not "int") to str`

LINGUAGEM INTERPRETADA (OU COMPILADA)

4. Características Marcantes

Compilada: Os programas podem ser traduzidos para código de máquina e depois executados diretamente pelo computador.

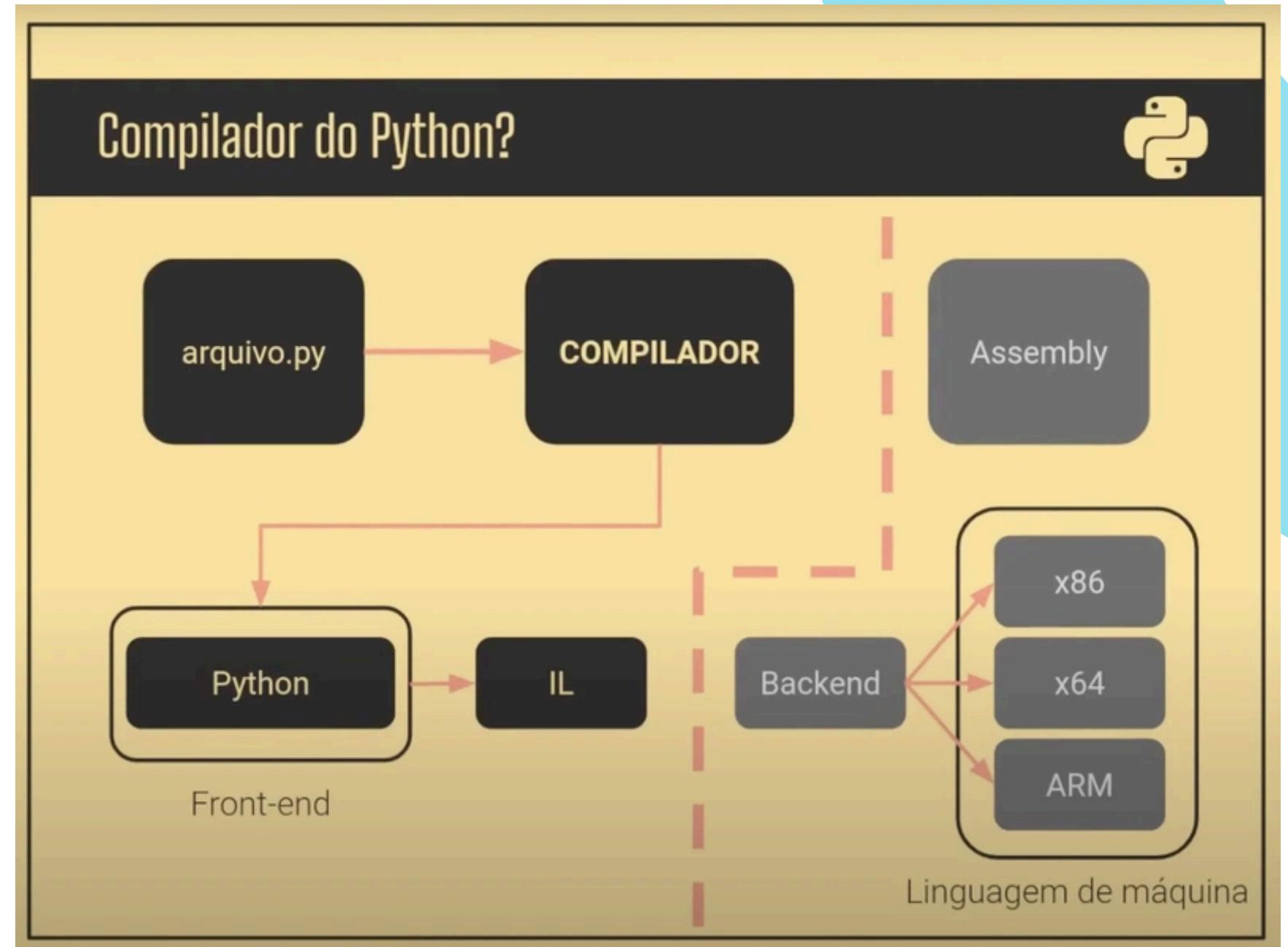
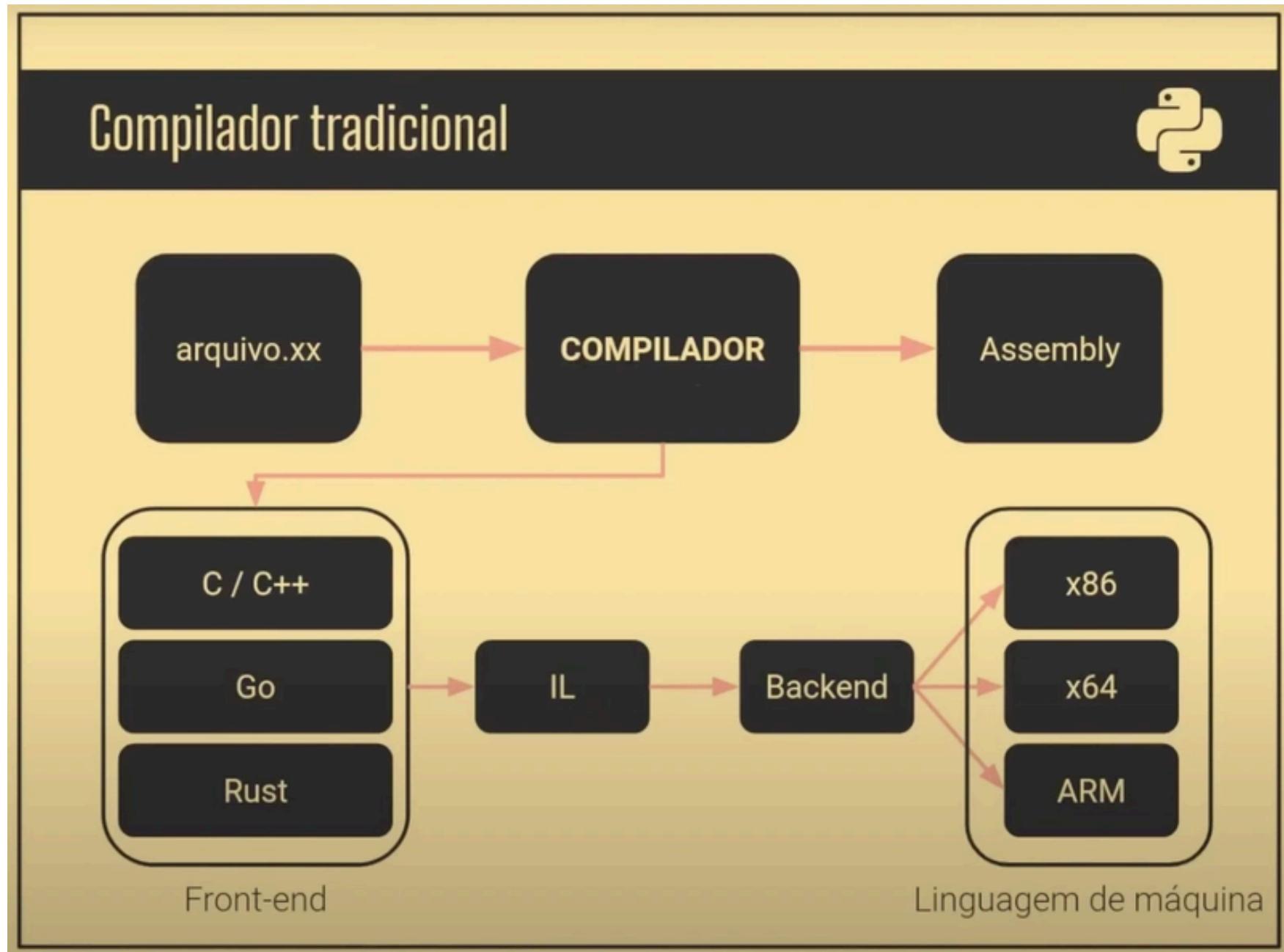
Interpretada: Os programas, escritos em linguagem de programação, são interpretados diretamente por outro software, chamado de interpretador. Não existe a etapa da tradução.

O Python é uma linguagem interpretada, ou seja, significa que ela não precisa passar pelo processo de compilação completo, executando diretamente o código linha por linha. Se houver erros no código do programa, ele será interrompido.

C_{Python}: interpretador referência da linguagem, é um compilador de bytecode e também interpretador.

4. Características Marcantes

LINGUAGEM INTERPRETADA (OU COMPILADA)



4. Características Marcantes

- **EXTENSA BIBLIOTECA PADRÃO**

O Python possui uma vasta coleção de módulos e pacotes que abrangem uma ampla gama de funcionalidades, desde manipulação de arquivos e acesso a bancos de dados até processamento de texto e criação de interfaces gráficas.



5. Aplicações



DESENVOLVIMENTO WEB:

Desenvolvedores Python podem usar frameworks da própria Web para rapidamente e eficientemente construir aplicações dinâmicas da web. Como Django e Flask (micro-framework) e módulos que ajudam a lidar com JSON, sockets, http-requests e mais, tornando o processo de web development mais fácil.



CIÊNCIA DE DADOS:

Python é amplamente usado em ciência de dados devido à sua simplicidade e poder analítico. Bibliotecas como NumPy, pandas e Matplotlib facilitam a manipulação, análise e visualização de dados. Python também integra facilmente com bancos de dados, arquivos CSV, APIs e planilhas.



INTELIGÊNCIA ARTIFICIAL

Python domina o desenvolvimento de IA com bibliotecas como TensorFlow e PyTorch, usadas para criar e treinar modelos inteligentes. A linguagem oferece recursos para trabalhar com machine learning, deep learning, e redes neurais. Sua sintaxe simples e grande comunidade facilitam o aprendizado e a inovação na área.

5. Aplicações

AUTOMAÇÃO / SCRIPTING

Python é amplamente usado para automatizar tarefas do dia a dia, como renomear arquivos em massa, processar planilhas, acessar APIs ou realizar testes automáticos. Sua sintaxe simples e bibliotecas poderosas facilitam a criação de scripts rápidos e eficientes.

DESENVOLVIMENTO DE JOGOS

A biblioteca Pygame permite criar jogos 2D de forma simples e didática. Embora não seja usada para grandes produções, é ótima para aprender lógica de jogos, trabalhar com eventos, animações e gráficos. É ideal para estudantes, prototipagem e projetos independentes.

IOT (MICRO PYTHON)

MicroPython é uma versão leve do Python projetada para rodar em placas como ESP32 e ESP8266. Com ele, é possível controlar sensores, LEDs, motores e outros componentes eletrônicos com poucos comandos. É uma ótima porta de entrada para projetos de automação residencial, robótica e IoT.

6. Linguagens Relacionadas

Linguagens Inspiradas

ABC → Simples e clara e com indexação obrigatória

HOW TO RETURN sum a b:
RETURN a + b

WRITE sum 3 5

C → Limitada e com grandes falhas, mas potente

```
#include <stdio.h>

int soma(int a, int b) {
    return a + b;
}

int main() {
    printf("%d\n", soma(3, 5));
    return 0;
}
```

Guido van Rossum trabalhava no CWI (Centrum Wiskunde & Informatica)

6. Linguagens Relacionadas

Linguagens Influenciadas

Python

```
def soma(a, b):  
    return a + b
```

```
print(soma(3, 5))
```

julia → linguagem simples e objetiva

```
function soma(a, b)  
    return a + b  
end  
  
println(soma(3, 5))
```

Boo → linguagem simples e clara

```
def soma(a as int, b as int) as int:  
    return a + b
```

```
print(soma(3, 5))
```

Swift → linguagem simples e eficiente

```
func soma(_ a: Int, _ b: Int) -> Int {  
    return a + b  
}
```

```
print(soma(3, 5))
```

6. Linguagens Relacionadas

Linguagens Similares

Lua → Criada no Brasil

```
function soma(a, b)
    return a + b
end

print(soma(3, 5))
```

Python

```
def soma(a, b):
    return a + b

print(soma(3, 5))
```

Scala → fácil de entender

```
def soma(a: Int, b: Int): Int = {
    a + b
}

println(soma(3, 5))
```

Ruby → programação bonita e agradável

```
def soma(a, b)
    a + b
end
```

```
puts soma(3, 5)
```

6. Linguagens Relacionadas

Linguagens Opostas

Python

```
def soma(a, b):
    return a + b

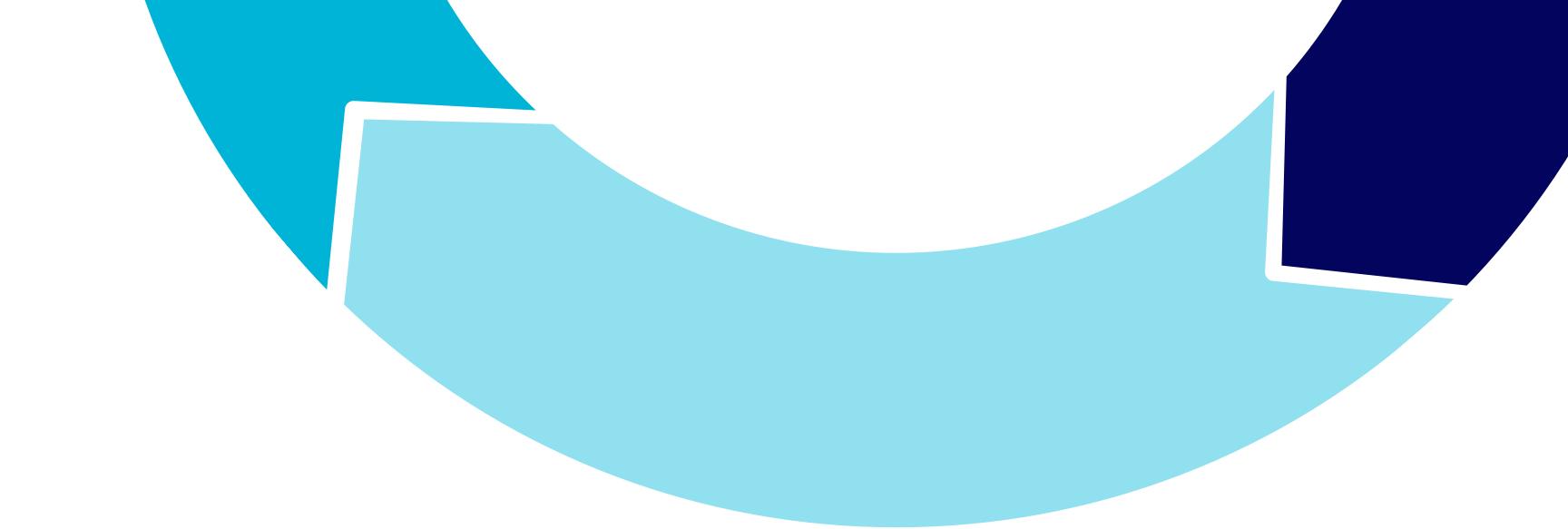
print(soma(3, 5))
```

assembly

```
section .data
msg db "Resultado: %d", 10, 0

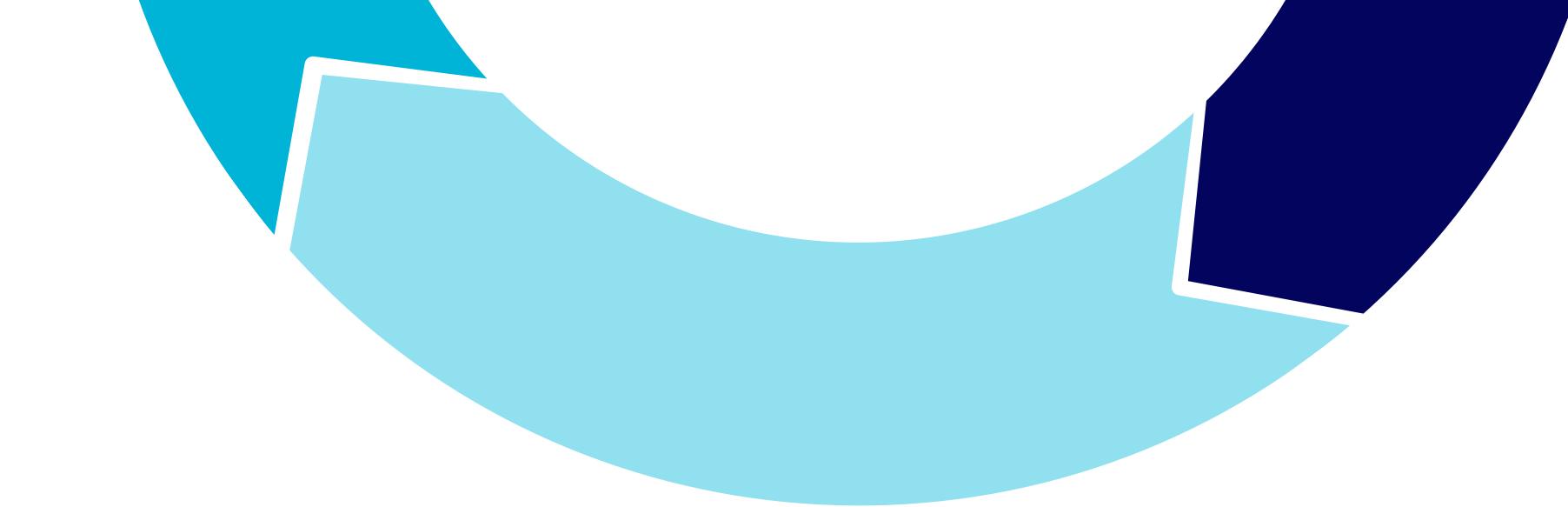
section .text
extern printf
global main

main:
    mov eax, 3
    add eax, 5
    push eax
    push msg
    call printf
    add esp, 8
    ret
```



7. Exemplos de Programa

- Arquivos de código Python são salvos com a extensão “Arquivo.py”
- Para rodar Python no terminal do Windows, use “python Arquivo.py”
- Para rodar no mac ou Linux, use “python3 ./Arquivo.py”



7. Exemplos de Programa

CÓDIGO

```
1  
2  
3  
4  
5     print ("Hello, World!")  
6  
7  
8  
9  
10
```



TERMINAL

```
Hello, World!  
  
** Process exited - Return Code: 0 **  
  
Press Enter to exit terminal
```

7. Exemplos de Programa

CÓDIGO

```
1 from sympy import symbols, integrate, sympify
2
3 # Solicita a expressão e a variável
4 expressao_str = input (
5     "\nDigite a função que deseja integrar (ex: x**2 + 3*x): "
6 )
7
8 variavel_str = input (
9     "\nDigite a variável de integração (ex: x): "
10 )
11
12 variavel = symbols (variavel_str)
13 expressao = sympify (expressao_str)
14
15 # Pergunta se será definida ou indefinida
16 tipo = input (" \nDeseja uma integral definida? (s/n): ")
17
18 if (tipo.lower () == 's'):
19     limite_inferior = float (input ("\nDigite o limite inferior: "))
20     limite_superior = float (input ("\nDigite o limite superior: "))
21     resultado = integrate (
22         expressao,
23         variavel, limite_inferior, limite_superior
24     )
25 else:
26     resultado = integrate (expressao, variavel)
27
28 print ("\nResultado da integral:", resultado)
```

TERMINAL

```
C:\Users\thiag\Desktop\dir>python c1.py
Digite a função que deseja integrar (ex: x**2 + 3*x): x**2
Digite a variável de integração (ex: x): x
Deseja uma integral definida? (s/n): s
Digite o limite inferior: 0
Digite o limite superior: 1
Resultado da integral: 0.3333333333333333
```

7. Exemplos de Programa

CÓDIGO

```
14
15 # 1. Divide o arquivo em blocos menores ordenados
16 arquivos_temp = []
17 with open (arquivo_entrada, "r") as arq:
18     while True:
19         linhas = [arq.readline () for _ in range (tamanho_bloco)]
20         linhas = [int (linha.strip ()) for linha in linhas if linha]
21         if not linhas:
22             break
23         linhas.sort ()
24         arq_temp = tempfile.NamedTemporaryFile (delete=False, mode="w")
25         for numero in linhas:
26             arq_temp.write (str (numero) + "\n")
27         arq_temp.close ()
28         arquivos_temp.append (arq_temp.name)
29
30 # 2. Merge dos arquivos usando heap
31 arquivos_abertos = [open (nome, "r") for nome in arquivos_temp]
32 heap = []
33
34 for i, arq in enumerate (arquivos_abertos):
35     linha = arq.readline ()
36     if linha:
37         heapq.heappush (heap, (int (linha.strip ()), i))
38
39 with open (arquivo_saida, "w") as saida:
40     while heap:
41         menor, origem = heapq.heappop (heap)
42         saida.write (str (menor) + "\n")
43         linha = arquivos_abertos[origem].readline ()
44         if linha:
45             heapq.heappush (heap, (int (linha.strip ()), origem))
46
47 # Fecha e remove arquivos temporários
48 for arq in arquivos_abertos:
49     arq.close ()
50 for nome in arquivos_temp:
51     os.remove (nome)
```

Exemplo de “External Merge Sort”, visto em AEDs3

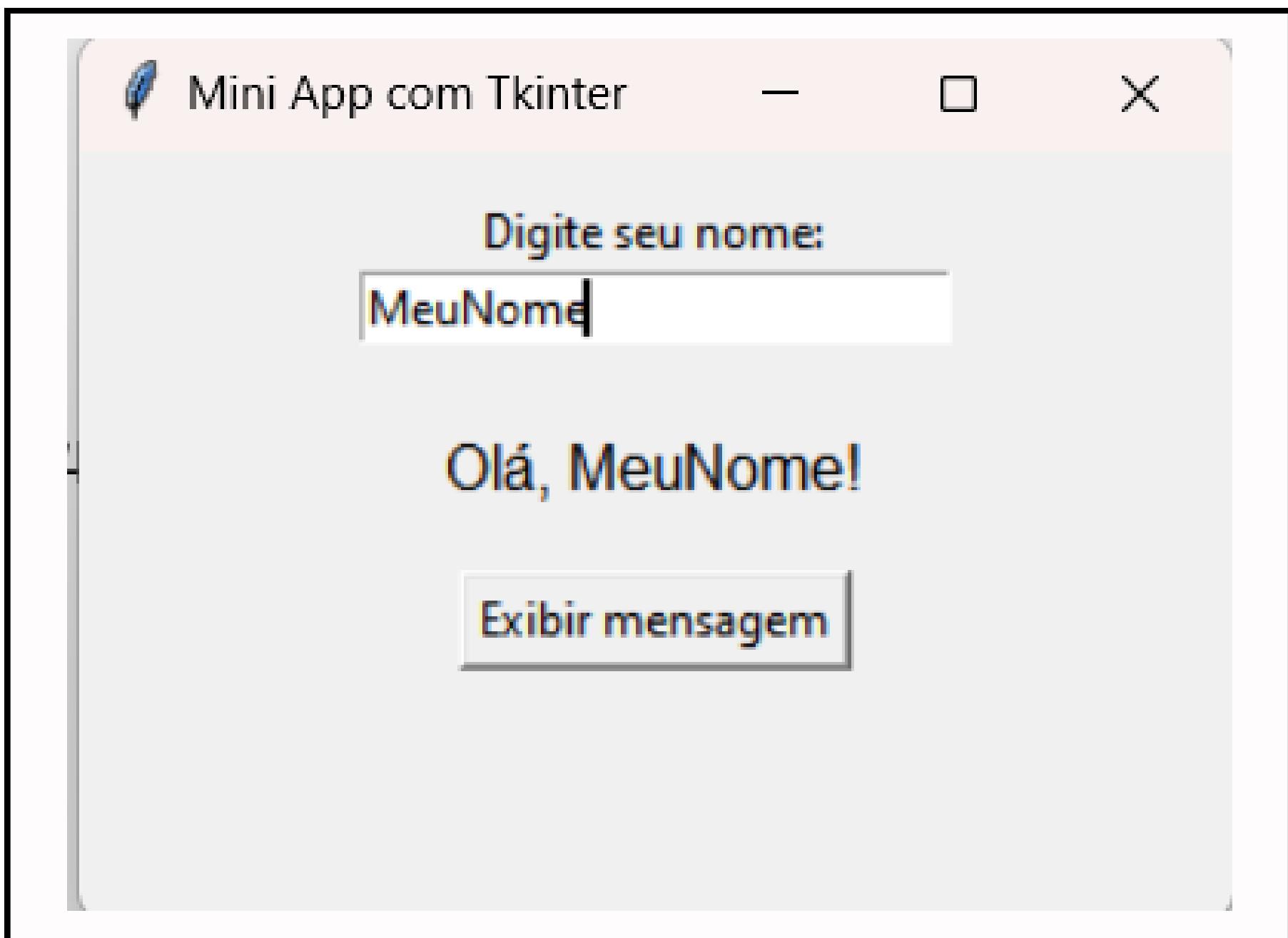
O algoritmo é capaz de ordenar uma quantia de dados muito maior do que a capacidade máxima da RAM permite, e faz isso intercalando entre arquivos usando o Heapsort e o Mergesort.

7. Exemplos de Programa

CÓDIGO

```
1 import tkinter as tk
2
3 janela = tk.Tk ()
4 janela.title ("Mini App com Tkinter")
5 janela.geometry ("300x200")
6
7 frame_superior = tk.Frame (janela)
8 frame_superior.pack (pady=10)
9
10 rotulo_instrucao = tk.Label (frame_superior, text="Digite seu nome:")
11 rotulo_instrucao.pack ()
12
13 entrada_nome = tk.Entry (frame_superior, width=25)
14 entrada_nome.pack ()
15
16 rotulo_mensagem = tk.Label (janela, text="", font=("Arial", 12))
17 rotulo_mensagem.pack (pady=10)
18
19 def atualizarMensagem ():
20     nome = entrada_nome.get ()
21     if nome:
22         rotulo_mensagem.config (text=f"Olá, {nome}!")
23     else:
24         rotulo_mensagem.config (text="Você não digitou nada!")
25
26 def limparMensagem ():
27     entrada_nome.delete (0, tk.END)
28     rotulo_mensagem.config (text="")
29
30 frame_botoes = tk.Frame (janela)
31 frame_botoes.pack (pady=5)
32
33 botao_exibir = tk.Button (frame_botoes, text="Exibir mensagem", command=atualizarMensagem)
34 botao_exibir.pack (side=tk.LEFT, padx=5)
35
36 janela.mainloop ()
```

TELA

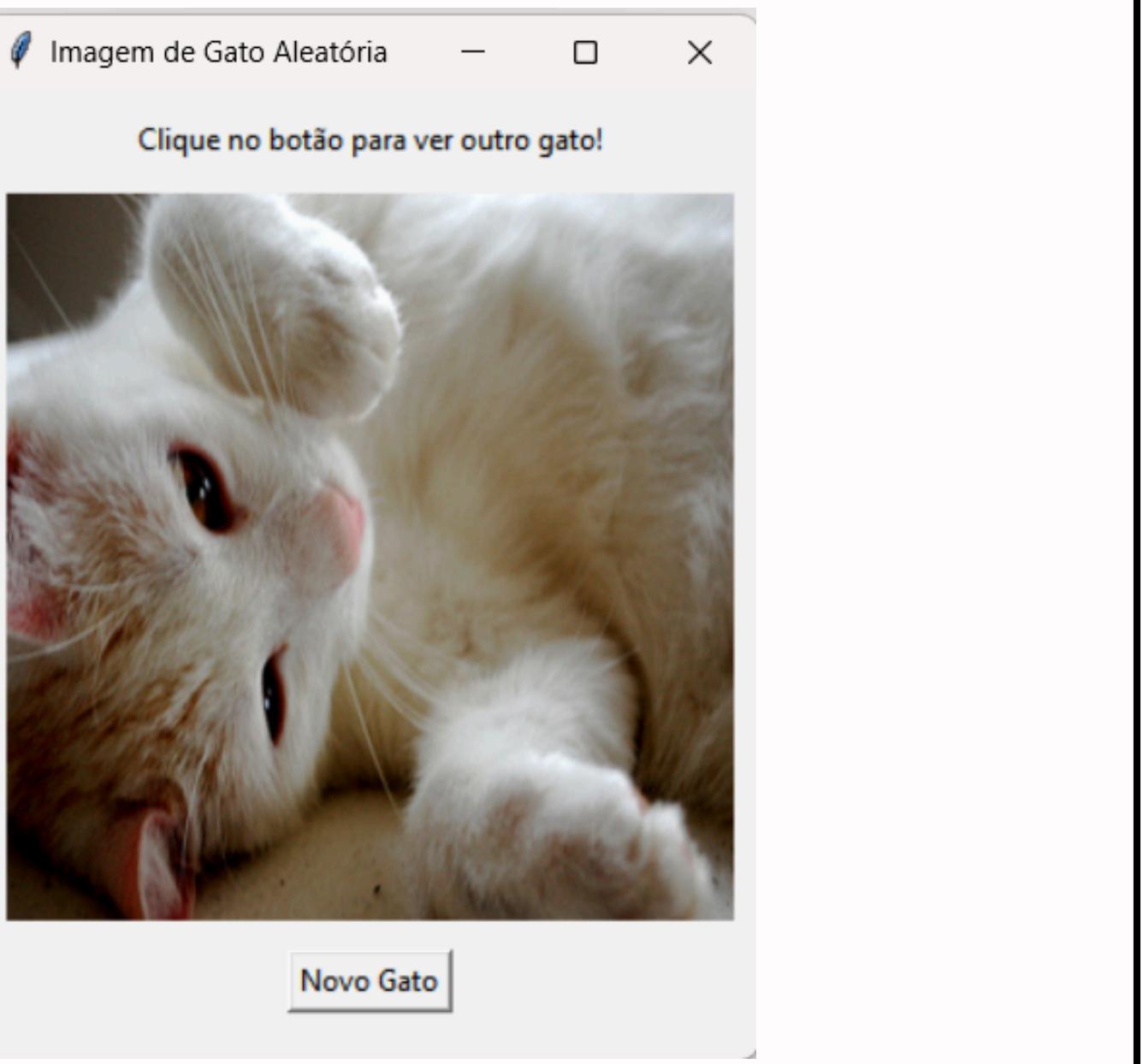


7. Exemplos de Programa

CÓDIGO

```
1 import tkinter as tk
2 from PIL import Image, ImageTk
3 import requests
4 from io import BytesIO
5
6 def carregarImagenGato():
7     url_api = "https://api.thecatapi.com/v1/images/search"
8     resposta = requests.get(url_api)
9
10    if resposta.status_code == 200:
11        dados = resposta.json()
12        url_imagem = dados[0]["url"]
13        imagem_resposta = requests.get(url_imagem)
14        imagem_bytes = Image.open(BytesIO(imagem_resposta.content))
15
16        imagem_redimensionada = imagem_bytes.resize((300, 300))
17        imagem_tk = ImageTk.PhotoImage(imagem_redimensionada)
18
19        rotulo_imagem.config(image=imagem_tk)
20        rotulo_imagem.image = imagem_tk
21        rotulo_info.config(text="Clique no botão para ver outro gato!")
22    else:
23        rotulo_info.config(text="Erro ao carregar a imagem.")
24
25 janela = tk.Tk()
26 janela.title("Imagen de Gato Aleatória")
27 janela.geometry("320x400")
28
29 rotulo_info = tk.Label(janela, text="Clique no botão para carregar um gato 🐱")
30 rotulo_info.pack(pady=10)
31
32 rotulo_imagem = tk.Label(janela)
33 rotulo_imagem.pack()
34
35 botao = tk.Button(janela, text="Novo Gato", command=carregarImagenGato)
36 botao.pack(pady=10)
37
38 janela.mainloop()
```

TELA

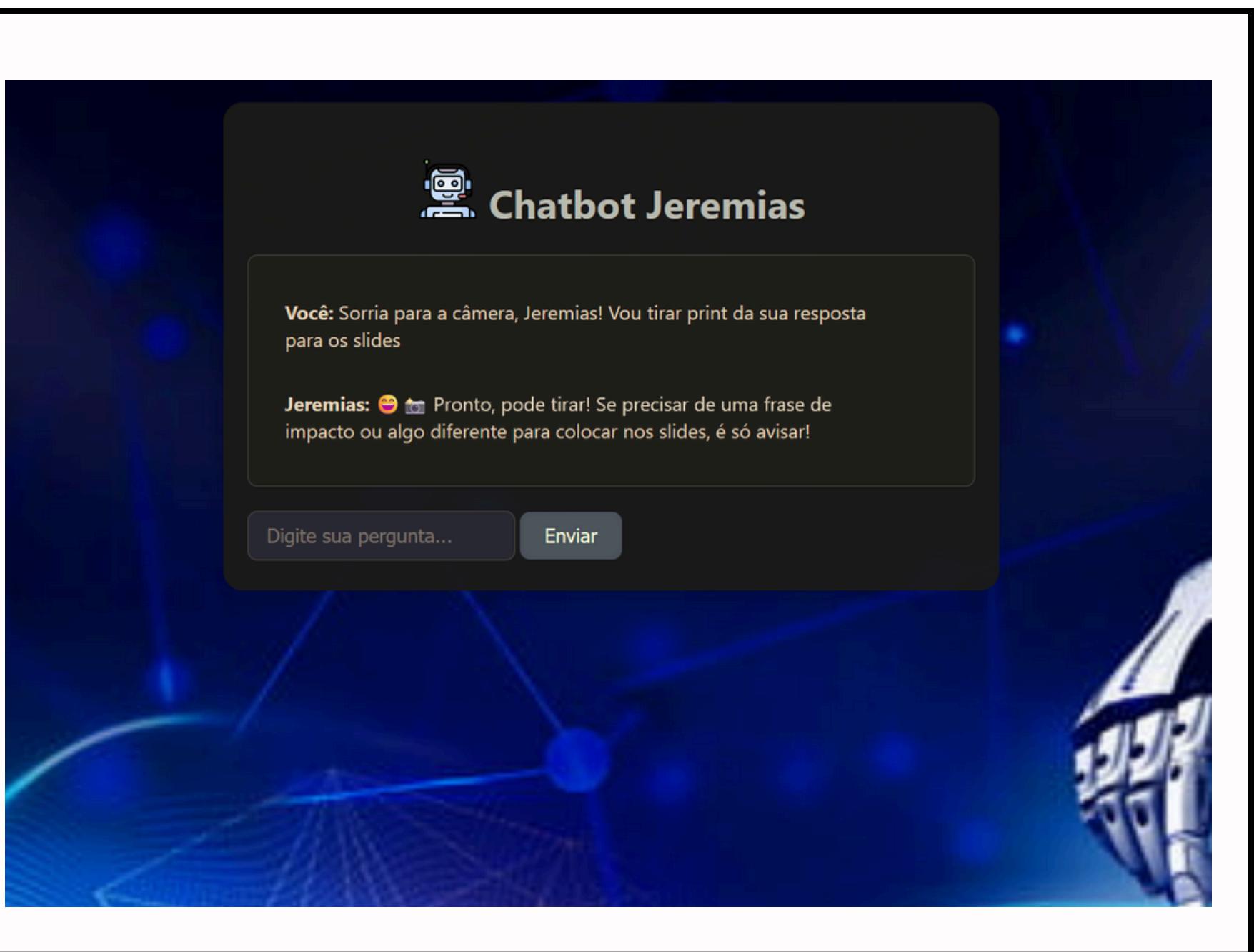


7. Exemplos de Programa

CÓDIGO

```
5 # Configurações Azure
6 endpoint = "https://14798-max9hw08-eastus2.cognitiveservices.azure.com/"
7 model_name = "gpt-4.1"          # Nome do modelo que está vinculado ao seu deployment
8 deployment = "gpt-4.1"          # Nome exato do deployment criado no portal
9 subscription_key = "#####"
10 api_version = "2024-12-01-preview"
11
12 # Cria o cliente AzureOpenAI
13 client = AzureOpenAI(
14     api_version=api_version,
15     azure_endpoint=endpoint,
16     api_key=subscription_key,
17 )
18
19 # Função que responde perguntas
20 def responder(pergunta: str) -> str:
21     global conversa_completa
22     try:
23         conversa_completa += f"eu: {pergunta}\n"
24         response = client.chat.completions.create(
25             model=deployment,
26             messages=[
27                 {"role": "system", "content": "Você é um assistente útil."},
28                 {"role": "user", "content": conversa_completa},
29             ],
30             max_tokens=800,
31             temperature=1.0,
32             top_p=1.0,
33             frequency_penalty=0.0,
34             presence_penalty=0.0,
35         )
36
37         resposta = response.choices[0].message.content.strip()
38         conversa_completa += f"voce: {resposta}\n"
39         return resposta
40     except Exception as e:
41         print(f"[ERRO AZURE] {e}")
42         return "Erro ao processar a pergunta com a IA."
43
```

TELA



8. Tutorial de instalação

Python vem pré-instalado em algumas máquinas porque é uma linguagem muito usada para automação, ciência de dados e desenvolvimento. Fabricantes ou o próprio Sistema Operacional podem incluir o Python para facilitar o uso por desenvolvedores ou para rodar scripts internos do sistema.

O primeiro passo para instalar Python é garantir que Python ainda não está instalado.

WINDOWS

```
C:\usuário> python --version
```

Python 3.11.9

LINUX/MACOS

```
$ python3 --version
```

Python 3.11.9

8. Tutorial de instalação

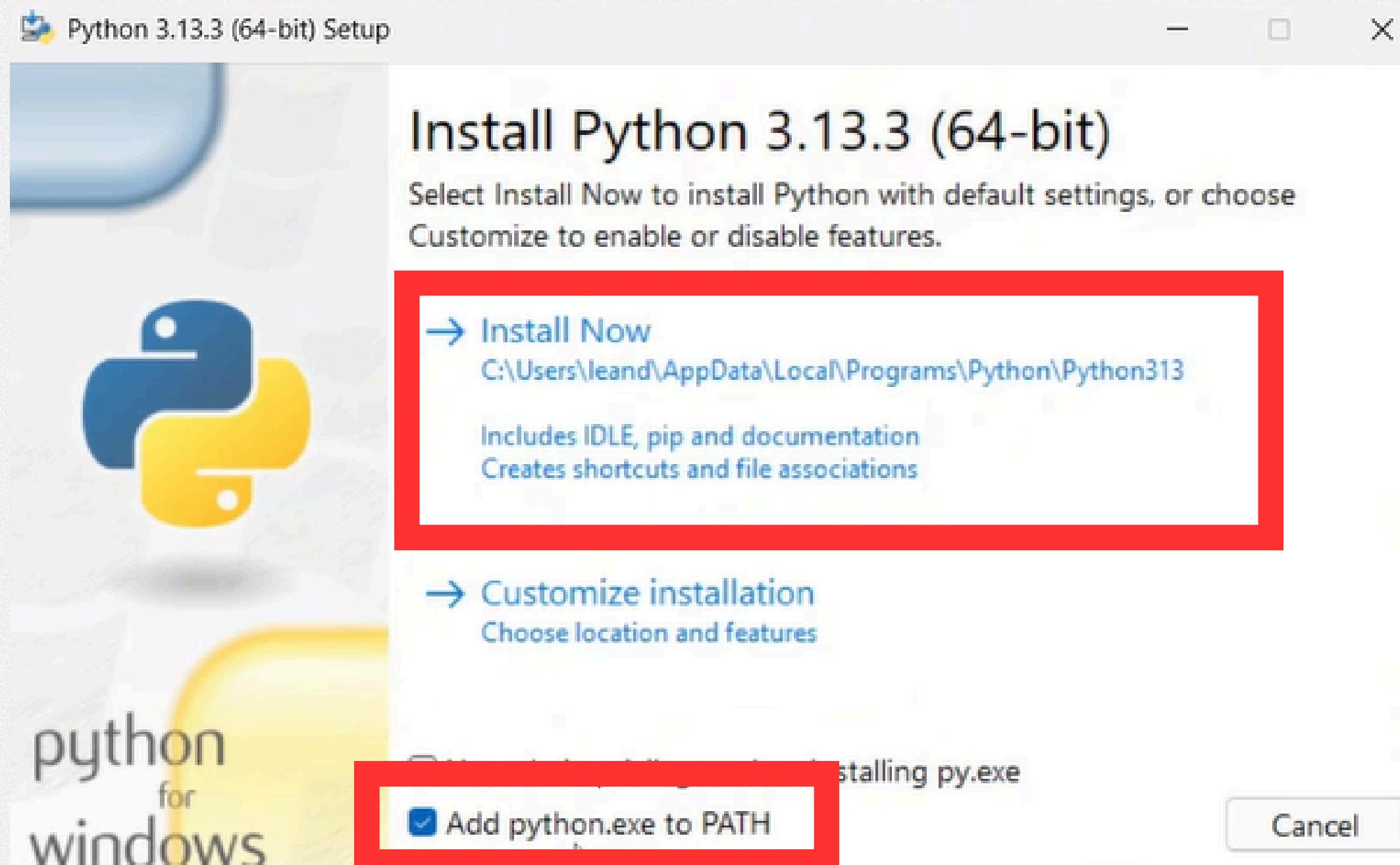
Site oficial do python:

<https://www.python.org/downloads/>

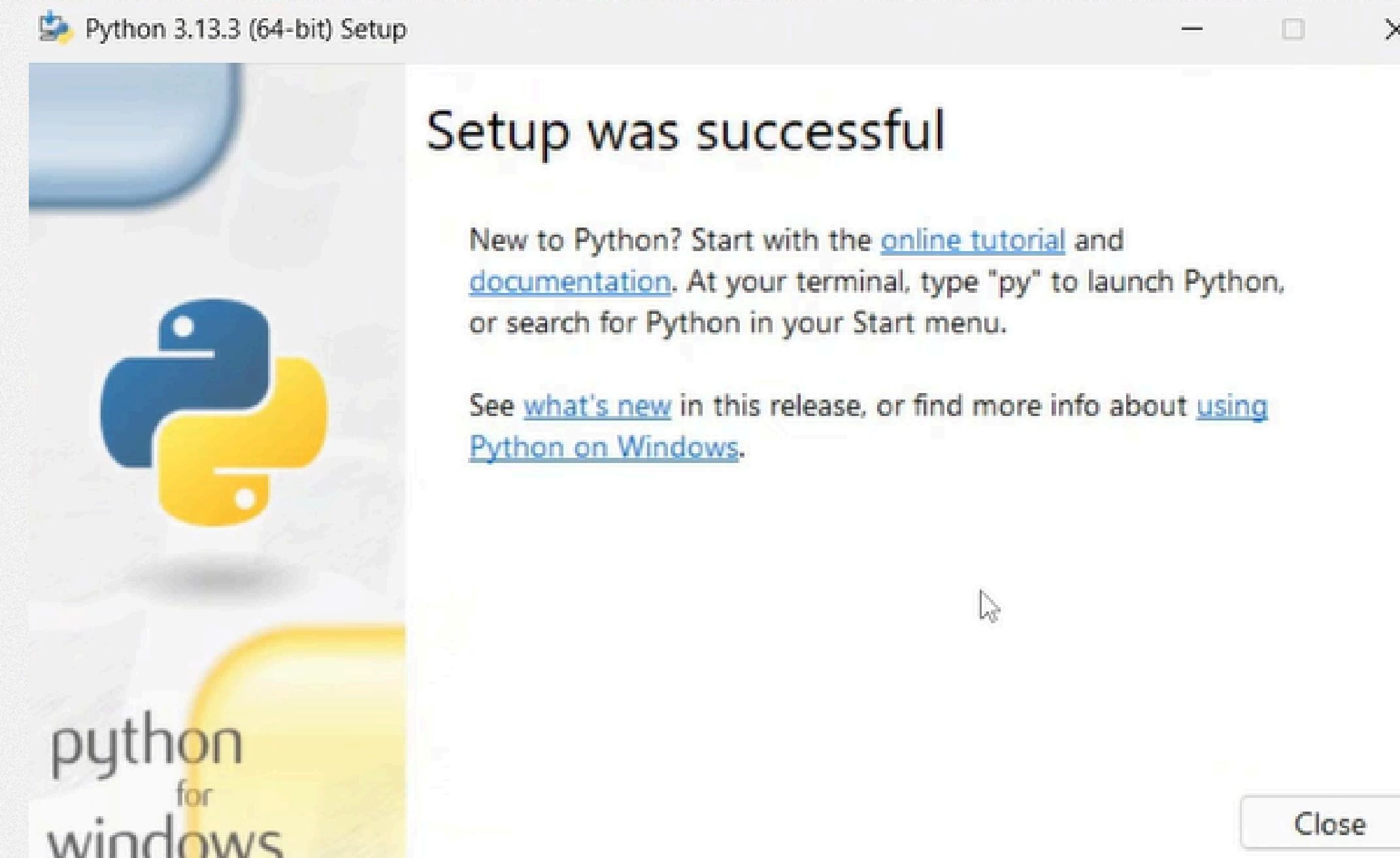
The screenshot shows the Python.org website's download section. At the top, there's a navigation bar with links for Python, PSF, Docs, PyPI, Jobs, and Community. Below the navigation is the Python logo and a search bar with a magnifying glass icon. A "Donate" button is also visible. The main content area features a large graphic of two yellow and white striped parachutes descending from the sky, each carrying a wooden crate. The text "Download the latest version for Windows" is prominently displayed, followed by a "Download Python 3.13.3" button. Below this, there are links for other operating systems and developer resources like pre-releases and Docker images. A section titled "Active Python Releases" provides details about the current versions, and a table at the bottom lists the Python versions, their maintenance status, first release date, end of support date, and release schedule.

Python version	Maintenance status	First released	End of support	Release schedule
3.14	pre-release	2025-10-01 (planned)	2030-10	PEP 745
3.13	bugfix	2024-10-07	2029-10	PEP 719

8. Tutorial de instalação para Windows



8. Tutorial de instalação para Windows



8. Tutorial de instalação para macOS

TERMINAL

```
# Verifica se o Homebrew está instalado; se não estiver, instala  
# O Homebrew é um gerenciador de pacotes para macOS  
/bin/bash -c "$(curl https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)" -fsSL  
  
# Usa o Homebrew para instalar o Python 3  
brew install python  
  
# Verifica se o Python foi instalado corretamente  
python3 --version
```

8. Tutorial de instalação para Linux

TERMINAL

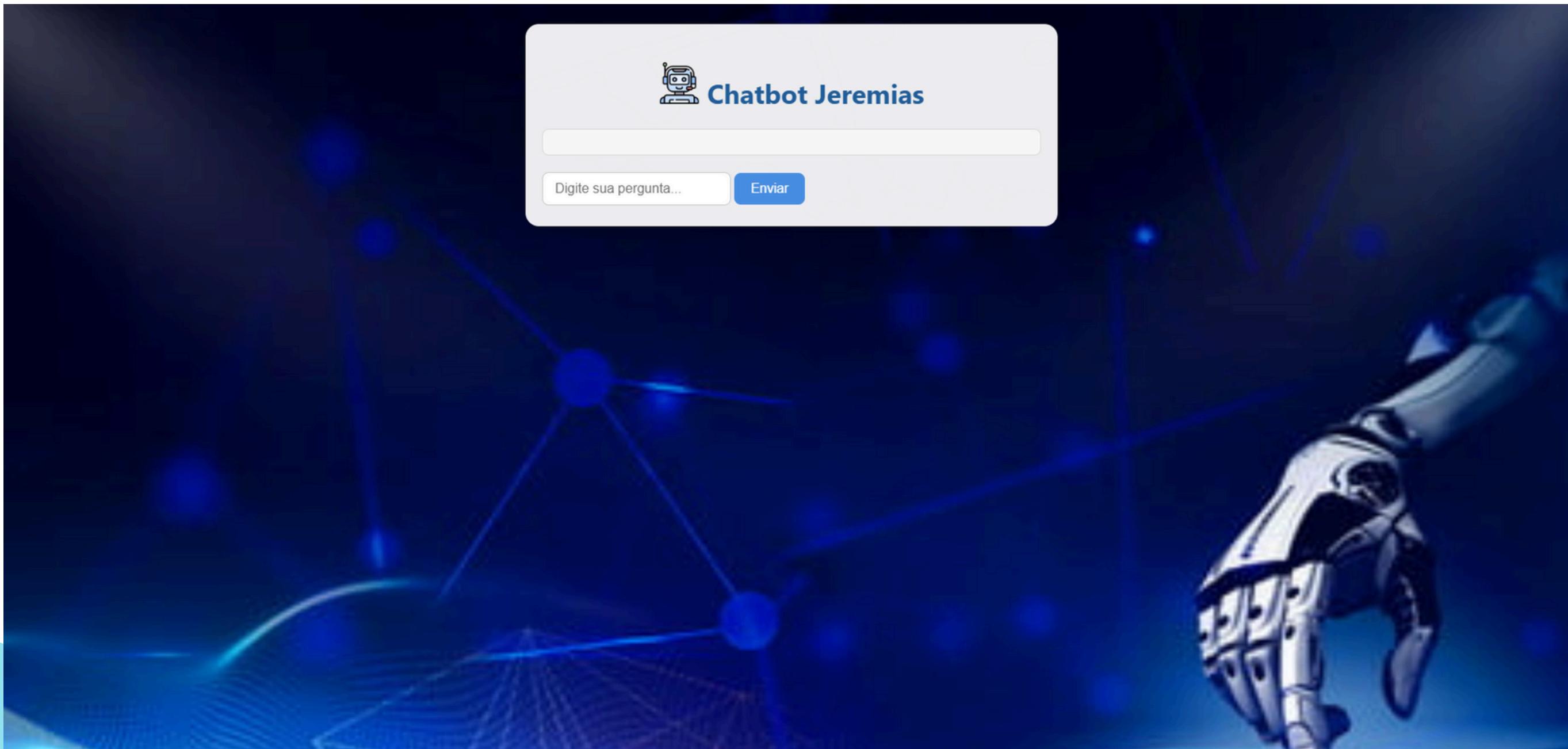
```
# Atualiza a lista de pacotes  
sudo apt update
```

```
# Instala o python 3  
sudo apt install python3
```

```
# Instala o pip separadamente  
sudo apt install python3-pip
```

```
# Verifica a instalação  
python3 --version
```

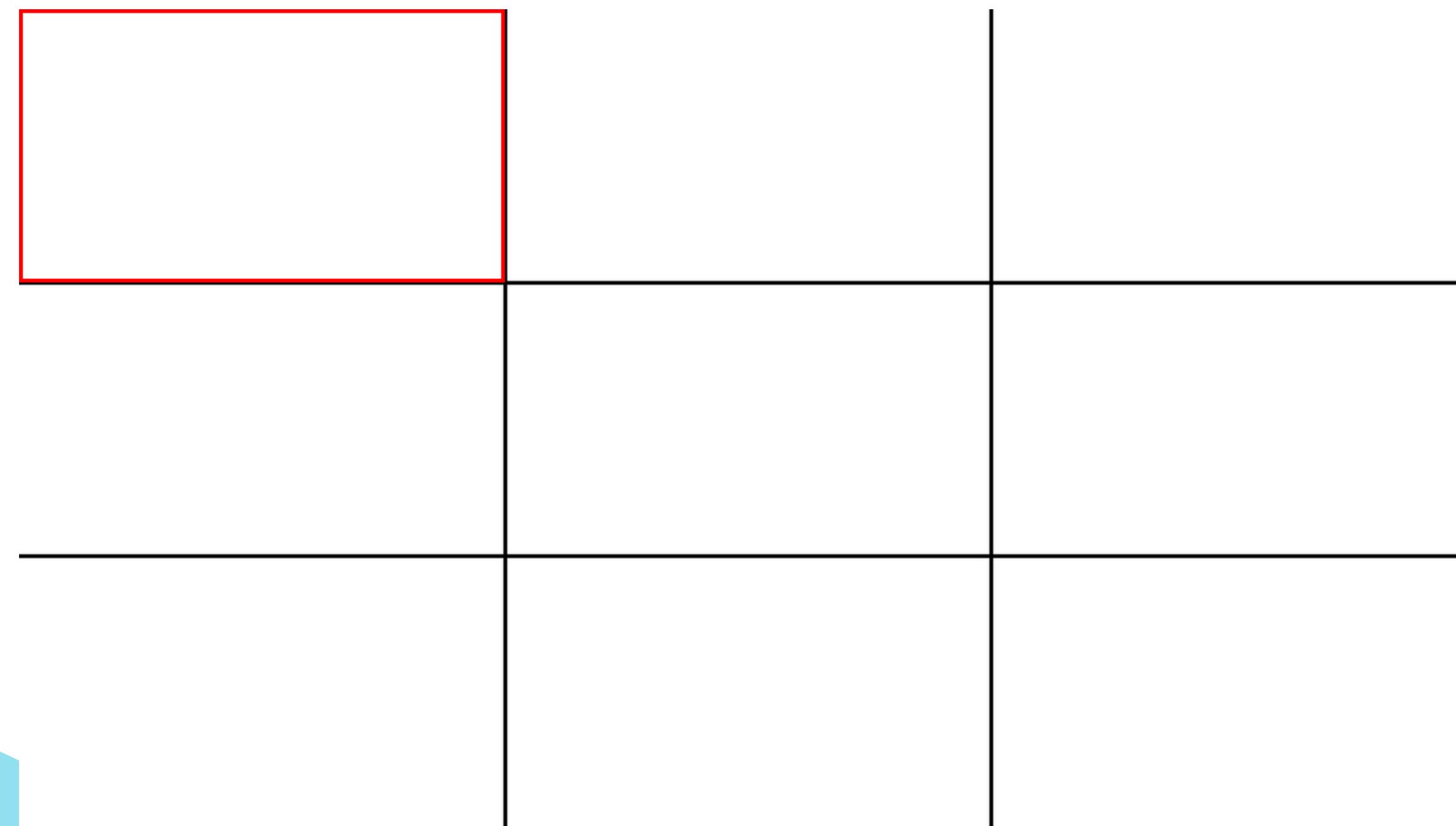
9. Trabalho prático



9. Trabalho prático



9. Trabalho prático



9. Estrutura do Jeremias

- Static com css e imagens
- Templates com o front-end html
- Dois códigos em Python

```
  <img alt="static folder icon" /> static
    <img alt="image file icon" /> fundo.png
    <img alt="image file icon" /> robo.png
    # style.css
  <img alt="templates folder icon" /> templates
    <img alt="HTML file icon" /> index.html
    <img alt="Python file icon" /> app.py
    <img alt="Python file icon" /> chatbot.py
```

9. Estrutura do Jeremias

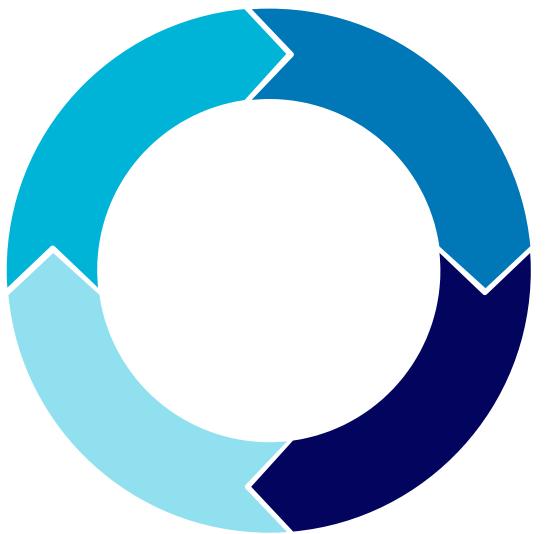
- App.py
- Pega a função “responder” de Chatbot.py
- Desenha o HTML na tela
- Se o HTML enviar uma pergunta, responde ele usando o “responder”

```
1  from flask import Flask, render_template, request, jsonify
2  from chatbot import responder
3
4  app = Flask(__name__)
5
6  @app.route("/")
7  def index():
8      return render_template("index.html")
9
10 @app.route("/perguntar", methods=["POST"])
11 def perguntar():
12     try:
13         dados = request.get_json()
14         if not dados or "pergunta" not in dados:
15             return jsonify({"resposta": "Erro: pergunta não recebida."}), 400
16
17         pergunta = dados["pergunta"]
18         resposta = responder(pergunta)
19         return jsonify({"resposta": resposta})
20
21     except Exception as e:
22         print(f"Erro ao processar pergunta: {e}")
23         return jsonify({"resposta": "Erro interno no servidor."}), 500
24
25 if __name__ == "__main__":
26     app.run(debug=True)
```

9. Estrutura do Jeremias

- Chatbot.py
- Armazena toda a conversa como contexto
- Define a função responder, que usa a API do AZURE

```
1 from openai import AzureOpenAI
2
3 conversa_completa = "Histórico da Conversa até agora: \n"
4
5 # Configurações Azure
6 ### Bloco com chave de API e informações do AZURE removida por segurança
7
8 # Cria o cliente AzureOpenAI
9 client = AzureOpenAI(
10     api_version=api_version, # type: ignore
11     azure_endpoint=endpoint, # type: ignore
12     api_key=subscription_key, # type: ignore
13 )
14
15 # Função que responde perguntas
16 def responder(pergunta: str) -> str:
17     global conversa_completa
18     try:
19         conversa_completa += f"eu: {pergunta}\n"
20         response = client.chat.completions.create(
21             model=deployment, # type: ignore
22             messages=[
23                 {"role": "system", "content": "Você é um assistente útil."},
24                 {"role": "user", "content": conversa_completa},
25             ],
26             max_tokens=800,
27             temperature=1.0,
28             top_p=1.0,
29             frequency_penalty=0.0,
30             presence_penalty=0.0,
31         )
32
33         resposta = response.choices[0].message.content.strip()
34         conversa_completa += f"voce: {resposta}\n"
35         return resposta
36     except Exception as e:
37         print(f"[ERRO AZURE] {e}")
38         return "Erro ao processar a pergunta com a IA."
```



9. Instalando bibliotecas

Usamos o “pip install” para instalar bibliotecas de python

pip já vem instalado com o Python desde a versão 3.4

VERSÃO MAIS RECENTE SUPORTADA

```
pip install requests
```

```
pip install pandas
```

```
pip install numpy
```

ALGUMA VERSÃO ESPECÍFICA

```
pip install nomeDaBib1==1.2.3
```

```
pip install nomeDaBib2>=1.2.3
```

```
pip install nomeDaBib3>=1.2.0,<2.0.0
```

10. Considerações Finais

O sucesso do Python vem de sua simplicidade, versatilidade e forte comunidade. Ele uniu acessibilidade com poder técnico, ganhando espaço em diversas áreas da tecnologia. Hoje, Python é essencial em empresas, pesquisas e educação, consolidando-se como uma das linguagens mais influentes do mundo.

É a linguagem que conecta áreas como inteligência artificial, análise de dados, automação, ciência e web. Sua importância vai além da tecnologia: é um agente de inovação, acessibilidade e transformação digital. Entre áreas de crescimento do Python se tem:

- Crescimento contínuo em IA, machine learning e automação
- Expansão no desenvolvimento de APIs modernas e aplicações assíncronas
- Uso intensivo em ciência, saúde e pesquisas ambientais
- Adoção crescente em Internet das Coisas (IoT)
- Aumento de performance com iniciativas como o Faster CPython
- Fortalecimento na educação e formação de novos programadores

Apêndices

Rayssa

- Mesmo sem declarar tipos, o Python impede operações inválidas entre tipos, mantendo segurança em tempo de execução.
- Ampla Biblioteca que permite o desenvolvimento em diversas áreas, e podem ser utilizadas em outras linguagens de programação.

Apêndices

Thiago

- A ampla biblioteca e a facilidade de uso fazem do Python uma das linguagens mais preparadas para implementações em IoT.
- Essa mesma facilidade também torna o Python uma excelente escolha para a prototipação de projetos, permitindo desenvolver soluções de forma ágil e intuitiva.

Apêndices

Matheus

- Python combina uma sintaxe clara e acessível com grande capacidade técnica, permitindo expressar conceitos complexos de forma concisa e eficiente.
- Python é multiplataforma e integra-se facilmente com outras linguagens (C, C++, Java), oferecendo flexibilidade para projetos que demandam desempenho e facilidade de desenvolvimento.

Apêndices

Letícia

- Praticidade e eficiência
- vasta gama de bibliotecas e frameworks

Apêndices

Bruno

- Python é muito versátil
- Python em áreas diferentes: jogos, automações, IoT, scripts.
- Python é simples



Agradecemos a atenção!



Bibliografia

ALURA. Python: o que é, para que serve e por que aprender. Disponível em: <https://www.alura.com.br/artigos/python>. Acesso em: 17 de maio de 2025

ARDUINO. Internet of Things with MicroPython. Arduino Docs. Disponível em: <https://docs.arduino.cc/micropython/micropython-course/course/internet-of-things/>. Acesso em: 17 de maio de 2025.

DA SILVA, Rogério Oliveira; SOUSA SILVA, Igor Rodrigues. Linguagem de Programação Python. TECNOLOGIAS EM PROJEÇÃO, [S. I.], v. 10, n. 1, p. 55–71, 2019. Disponível em: <https://projecaociencia.com.br/index.php/Projecao4/article/view/1359>. Acesso em: 17 de maio de 2025.

HABBEMA, Gabriel. Explorando o Python. Medium, 2021. Disponível em: <https://medium.com/@habbema/explorando-o-python-edb6c4784dc9>. Acesso em: 17 de maio de 2025.

MENEZES, Nilo Ney Coutinho. Introdução à programação com Python: algoritmos e lógica de programação para iniciantes. 4. ed. Rio de Janeiro: Novatec, 2020.



Bibliografia

MICROPYTHON. *MicroPython – Python for Microcontrollers.* MicroPython.org. Disponível em: <https://micropython.org/>. Acesso em: 17 de maio de 2025.

PRATEEKSHA. Top Python Libraries and Scripts for Workflow Automation. Prateeksha Web Services. Disponível em: <https://prateeksha.com/blog/top-python-libraries-and-scripts-for-workflow-automation/>. Acesso em: 17 de maio de 2025.

PYAUTO GUI DOCUMENTATION. *PyAutoGUI Documentation.* Read the Docs. Disponível em: <https://pyautogui.readthedocs.io/en/latest/>. Acesso em: 17 de maio de 2025.

PYTHON PRO. Linguagens de programação: tipos, paradigmas e mais. YouTube, 2023. Disponível em: <https://www.youtube.com/watch?v=pxfZTAJDipY>. Acesso em: 17 de maio de 2025.

PYTHON SOFTWARE FOUNDATION. *Why is Python a dynamic language and also a strongly typed language.* Disponível em: <https://wiki.python.org/moin/Why%20is%20Python%20a%20dynamic%20language%20and%20also%20a%20strongly%20typed%20language>. Acesso em: 17 de maio de 2025.



Bibliografia

RANI, Bhumika. *Introduction of Programming Paradigms*. GeeksforGeeks. Disponível em: <https://www.geeksforgeeks.org/introduction-of-programming-paradigms/>.

SEBESTA, Robert W. Concepts of Programming Languages. 9. ed. Boston: Pearson/Addison Wesley, 2010.

UNIVERSIDADE DA TECNOLOGIA. Estudo sobre a linguagem Python. Disponível em: <https://universidadatedtecnologia.com.br/estudo-linguagem-python-2018/>. Acesso em: 17 de maio de 2025.

UNIVERSIDADE DA TECNOLOGIA. Classificações de linguagens de programação. Disponível em: <https://universidadatedtecnologia.com.br/linguagem-de-programacao-classificacoes/>. Acesso em: 17 de maio de 2025.