

# 1 Introdução

Neste material, desenvolveremos uma aplicação capaz de conversar com o ChatGPT por meio de sua API. A intenção é que ela

- permita que o usuário informe um assunto sobre o qual deseja que uma pergunta de alternativas seja formulada

- solicite a correção de uma pergunta de alternativas

A Figura 1.1 mostra a interface gráfica desejada.

Figura 1.1

ChatGPT - Gerador/Corretor de Questões

Gerar Questão

Assunto:

Tipo: ☐ Dissertativa ☐ Alternativa

Dificuldade: ☐ Fácil ☐ Média ☐ Difícil

Pergunta exemplo:

Resposta:

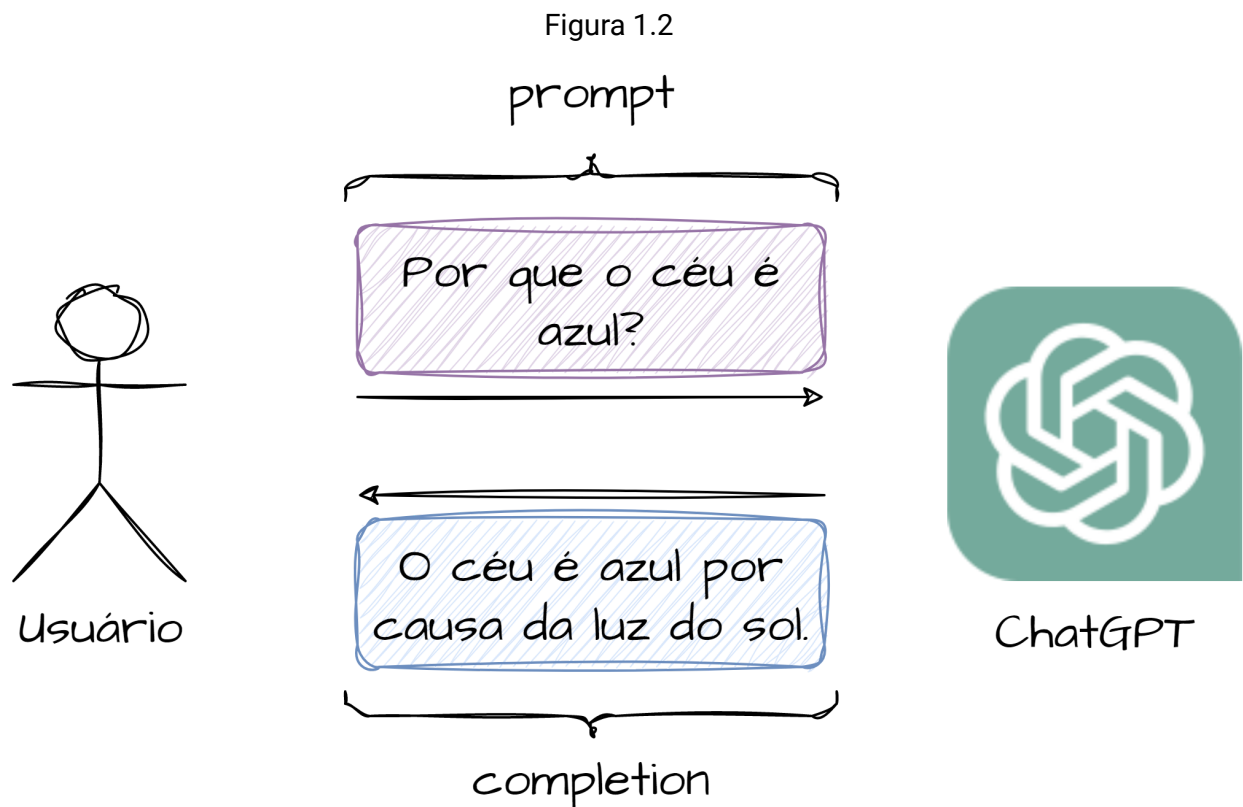
OK

O ChatGPT é capaz de realizar diferentes tarefas. Veja alguns exemplos:

- Geração de conteúdo (como gerar perguntas sobre assuntos diversos)
- Tradução de texto
- Resumos de texto
- Análise de sentimentos em textos (embora não tenha sido projetado com esse foco)
- Geração de imagens em função de texto

E muito mais!

Seu funcionamento, em geral, se baseia num modelo em que o usuário informa um **prompt** a partir do qual ele produz um **completion**. Veja a Figura 1.2.



A elaboração de prompts precisos é fundamental para obter com completions de qualidade. Fazemos alguns testes. Visite o Link 1.1 para ter acesso ao ChatGPT por meio de sua interface gráfica.

Link 1.1

<https://chat.openai.com/>

Faça um primeiro teste enviando o prompt

### **Elabore uma questão**

Observe como o resultado é, evidentemente, genérico. Podemos ser mais precisos dizendo um assunto. Tente esse prompt

### **Elabore uma questão sobre Java**

Já melhorou. Sejam os ainda mais específicos com esse prompt

### **Elabore uma questão sobre estruturas de seleção em Java**

E se desejarmos garantir que a questão é de alternativas? Tente esse prompt agora:

### **Elabore uma questão de alternativas sobre estruturas de seleção em Java**

Claro, também podemos dizer quantas alternativas desejamos, como mostra esse prompt

### **Elabore uma questão de alternativas sobre estruturas de seleção em Java. Use 5 alternativas.**

E se desejarmos escolher o nível de dificuldade da questão? Tente esse prompt

### **Elabore uma questão de alternativas sobre estruturas de seleção em Java. Use 5 alternativas. Nível muito fácil.**

Tente esse também

### **Elabore uma questão de alternativas sobre estruturas de seleção em Java. Use 5 alternativas. Nível ultra hard core.**

Peça também a resposta certa com esse prompt

### **Elabore uma questão de alternativas sobre estruturas de seleção em Java. Use 5 alternativas. Nível ultra hard core. Inclua a resposta certa.**

Para fechar, peça que ele tente resolver uma questão com o seguinte prompt

Uma pessoa acaba de encontrar uma lâmpada mágica e tem direito a fazer um pedido. A lógica está retratada no programa a seguir.

```
import javax.swing.JOptionPane;
public class SwitchCase{
    public static void main (String [] args){
        String menu = "Faça seu pedido\n1. Ficar rico\n2. Tirar nota boa em todas as
provas\n3. Um país sem corrupção\n";
        int op = Integer.parseInt(JOptionPane.showInputDialog(menu));
        switch(op){
            case 1:
                JOptionPane.showMessageDialog(null, "Você ganhou R$ 100.000.000,00");
            case 2:
                JOptionPane.showMessageDialog(null, "Você tirou 10 em todas as provas");
                break;
            case 3:
                JOptionPane.showMessageDialog(null, "Aí você pediu demais");
                break;
            default:
                if (op > 0){
                    JOptionPane.showMessageDialog(null, "Agora você terá direito a mais " + op
+ " pedidos");
                }
                else{
                    JOptionPane.showMessageDialog(null, "Infelizmente você não tem direito a
nenhum pedido");
                }
            }
        }
    }
}
```

Analise as seguintes proposições.

- I Se um usuário optar por ficar rico, ele também tirará 10 em todas as provas.
- II Se um usuário optar por tirar 10 em todas as provas, ele também ficará rico.
- III O programa termina com um erro em tempo de execução se o usuário digitar um valor negativo.

É correto apenas o que se afirma em

a I  
b II  
c III  
d I e II  
e II e III

## 2 Desenvolvimento

**2.2 (Nova pasta, VS Code, Terminal interno, ambiente virtual)** Crie uma pasta para abrigar os arquivos deste novo projeto. No Windows, uma sugestão é

```
C:\Users\usuario\Documents\dev\chatgpt_python
```

Em sistemas Unix like, use

```
/home/usuario/dev/chatgpt_python
```

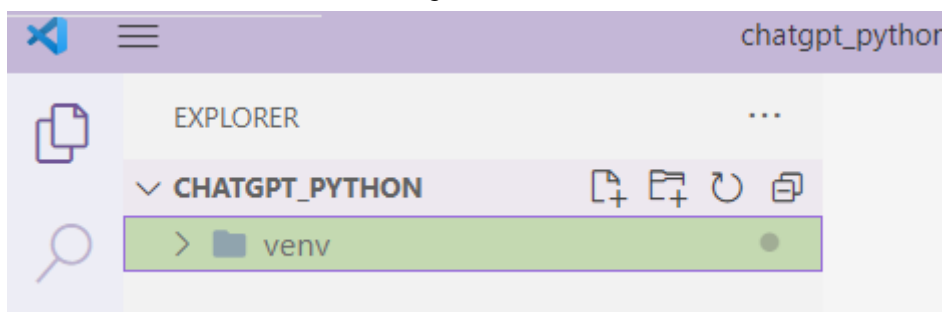
A seguir, abra o VS Code e clique em **File >> Open Folder** para mantê-lo vinculado à nova pasta. Por fim, ainda no VS Code, clique em **Terminal >> New Terminal** para abrir um novo terminal interno do VS Code, o que vai simplificar diversas tarefas.

No terminal, use

```
python -m venv venv
```

para criar um ambiente virtual Python chamado **venv** utilizando o módulo **venv**. Após a sua execução, uma pasta chamada venv deve ser criada na raiz de seu projeto, como na Figura 2.2.1.

Figura 2.2.1



**Nota.** Um ambiente virtual Python permite o uso de uma versão específica do Python e também de pacotes diversos. Isso permite que diferentes projetos Python com dependências de pacotes de versões diferentes possam ter vida sem impactar uns aos outros.

Depois da criação do ambiente virtual, é preciso ativá-lo. Usuários Windows, podem estar utilizando um terminal “cmd” ou um terminal “Powershell”. Você pode verificar o seu no próprio VS Code, como na Figura 2.2.2.

Figura 2.2.2



A Tabela 2.2.1 resume a forma como o ambiente virtual Python pode ser ativado em qualquer caso.

Tabela 2.2.1

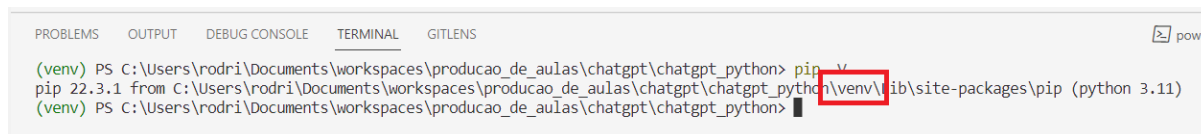
Terminal	Comando(s)
Windows cmd	venv\Scripts\activate.bat
Windows Powershell	Set-ExecutionPolicy -Scope CurrentUser unrestricted  venv\Scripts\Activate.ps1
Unix Like terminals	. venv/bin/activate

Em qualquer caso, você pode verificar se o ambiente foi ativado com sucesso com

`pip -V`

Como mostra a Figura 2.2.3, a pasta de seu ambiente virtual deve ser exibida.

Figura 2.2.3



**2.3 (Arquivo .env e chave de acesso ChatGPT)** O acesso ao ChatGPT por meio de sua API é pago. Entretanto, novos usuários podem fazer seus testes utilizando cinco dólares de crédito pelos primeiros três meses, na versão TRIAL.

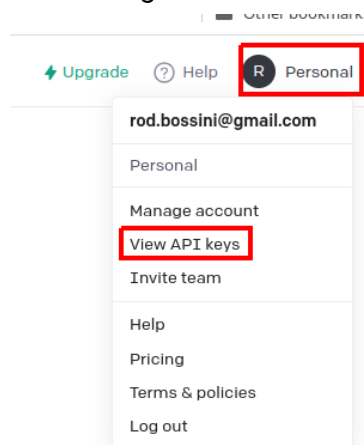
Para obter uma chave de API, basta fazer login no sítio do ChatGPT por meio do Link 2.3.1.

Link 2.3.1

<https://platform.openai.com/>

A seguir, clique no seu avatar no canto superior direito e escolha **View API Keys**, como mostra a Figura 2.3.1.

Figura 2.3.1



**Nota.** Caso deseje visualizar seu consumo e a data de expiração de seu período trial, visite o Link 2.3.2.

Link 2.3.2

<https://platform.openai.com/account/usage>





Na parte central da tela, você deverá ver um botão **Create new secret key**", como na Figura 2.3.2.

Figura 2.3.2

### API keys

Your secret API keys are listed below. Please note that we do not display your secret API keys again after you generate them.

Do not share your API key with others, or expose it in the browser or other client-side code. In order to protect the security of your account, OpenAI may also automatically rotate any API key that we've found has leaked publicly.

NAME	KEY	CREATED	LAST USED	
Secret key	sk-...xZqC	Apr 4, 2023	Never	 
python quick start	sk-...QuhS	Apr 20, 2023	Apr 20, 2023	 
<a href="#">+ Create new secret key</a>				

### Default organization

If you belong to multiple organizations, this setting controls which organization is used by default when making requests with the API keys above.

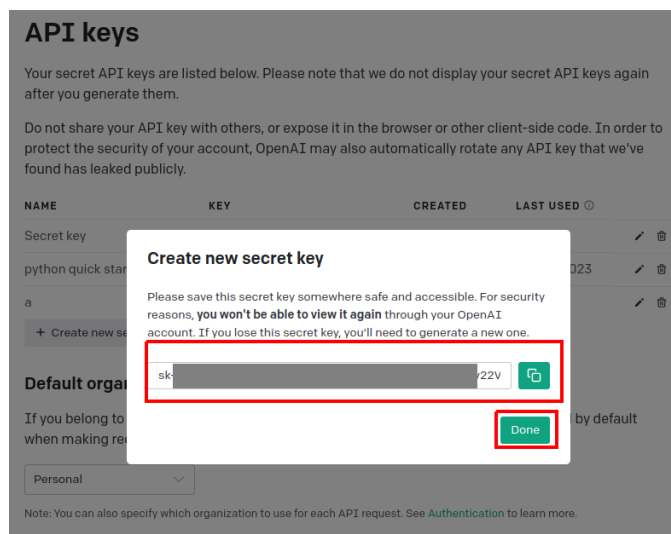
Personal

Note: You can also specify which organization to use for each API request. See [Authentication](#) to learn more.

Clique nele e invente um nome para a sua chave. A seguir, basta copiá-la e clicar **Done**, como na Figura 2.3.3.



Figura 2.3.3



**Nota. Você precisa copiar a chave agora.** Depois de fechar a janela não será mais possível visualizá-la. Se isso acontecer, você poderá criar outra.

Evidentemente, nosso programa fará uso da chave. Porém, é fundamental que ela não fique escrita explicitamente junto com o código. Isso por, pelo menos, duas razões:

- o controle de versão pode ser feito em repositórios públicos
- a chave a ser utilizada varia em função do ambiente (desenvolvimento, homologação, produção etc). Por isso, ela é chamada “**variável de ambiente**” e desejamos uma maneira simples de fazer a troca de chave conforme o ambiente muda.

Diferentes ambientes utilizam diferentes mecanismos para isso. Em Python, vamos criar um arquivo chamado **.env** (o nome começa com “.” para que ele seja oculto e “env” é de “environment”, ou seja, ambiente, um arquivo capaz de abrigar variáveis de ambiente) e armazenar a chave nele, como no Código 2.3.1.

Código 2.3.1

```
OPENAI_API_KEY=coloque a sua chave aqui
```

Agora é necessário tornar a chave disponível como uma variável de ambiente. Para tal, utilizaremos o módulo do Python chamado **python-dotenv**. Ele será uma **dependência** de nosso projeto. Ou seja, para que a aplicação funcione, ela depende de sua existência. Em geral, módulos dos quais uma aplicação Python depende são declarados num arquivo chamado **requirements.txt**. Por isso crie um arquivo com este nome na raiz do seu projeto e adicione o conteúdo do Código 2.3.2 a ele.

## Código 2.3.2

```
python-dotenv
```

No terminal, execute

```
pip install -r requirements.txt
```

para fazer a instalação deste módulo.

A seguir, vamos realizar um teste e verificar se já é possível acessar as variáveis de ambiente. Para tal, crie um arquivo chamado **app.py**. Vamos importar a função **load\_dotenv** pois ela é capaz de ler o conteúdo do arquivo `.env` e disponibilizar seu conteúdo como variáveis de ambiente. Importamos também o módulo **os** para utilizar a sua função **getenv**. A ela entregamos o nome de uma variável de ambiente e ela nos devolve o valor de interesse. Veja o Código 2.3.3.

## Código 2.3.3

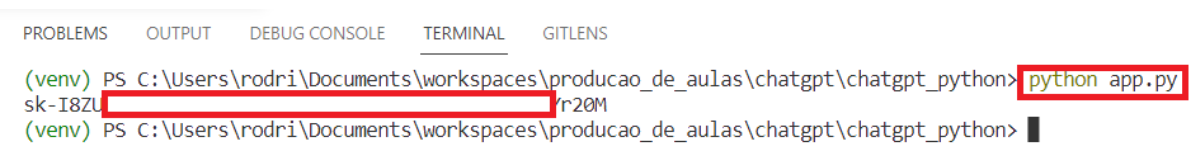
```
import os
from dotenv import load_dotenv
load_dotenv()
#exibindo o valor associado à chave OPENAI_API_KEY
print (os.getenv('OPENAI_API_KEY'))
```

Use

```
python app.py
```

no terminal. Se tudo deu certo, você deverá visualizar a sua chave de acesso ao ChatGPT. Veja a Figura 2.3.4.

Figura 2.3.4



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  GITLENS
(venv) PS C:\Users\rodri\Documents\workspaces\producao_de_aulas\chatgpt\chatgpt_python> python app.py
sk-I8ZU...r20M
(venv) PS C:\Users\rodri\Documents\workspaces\producao_de_aulas\chatgpt\chatgpt_python> █
```

Agora podemos deixar de exibir a chave. Como mostra o Código 2.3.4, basta comentar a linha. Ela pode ser útil no futuro.

Código 2.3.4

```
import os
from dotenv import load_dotenv
load_dotenv()
#exibindo o valor associado à chave OPENAI_API_KEY
#print (os.getenv('OPENAI_API_KEY'))
```

**2.4 (Módulo para o acesso ao ChatGPT)** Criaremos um módulo Python a fim de isolar o código de acesso à API do ChatGPT, promovendo seu nível de reusabilidade e facilidade de manutenção. Comece criando um arquivo chamado **chatgpt.py** na raiz do projeto. Nosso novo módulo utilizará o módulo do ChatGPT e, por isso, este precisa ser descrito no arquivo **requirements.txt**, como no Código 2.4.1.

Código 2.4.1

```
python-dotenv
openai
```

Instale a nova dependência com

**pip install -r requirements.txt**

A seguir, vamos escrever uma função cuja finalidade **será interagir com o ChatGPT a fim de obter uma pergunta** de acordo com os seguintes parâmetros:

- Chave de acesso à API do ChatGPT
- Assunto desejado
- Tipo desejado (dissertativa ou alternativa)
- Dificuldade (Fácil, Médio, Difícil)
- Pergunta exemplo (opcional) (uma pergunta em que o ChatGPT pode se basear)

A função começa produzindo um texto que explica mais detalhadamente o que desejamos. Ou seja, ela tenta deixar o nosso **prompt** o mais compreensível possível para o ChatGPT. Veja o Código 2.4.2.

## Código 2.4.2

```
import openai

def criar_pergunta(
    OPENAI_API_KEY,
    assunto,
    tipo,
    dificuldade,
    pergunta_exemplo = None
):
    openai.api_key = OPENAI_API_KEY
    assunto = f'Elabore uma pergunta sobre {assunto}'
    tipo = f'Ela deve ser {tipo}' + ' com 4 alternativas' if tipo ==
'Alternativa' else ''
    dificuldade = f'Seu nível de dificuldade deve ser {dificuldade}'
    pergunta_exemplo = f'Utilize esta pergunta como exemplo:
{pergunta_exemplo}' if pergunta_exemplo != None and pergunta_exemplo
!= '\n' else ''
    prompt = f'{assunto}.{tipo}.{dificuldade}.{pergunta_exemplo}'
```

Agora que temos o prompt em mãos, podemos enviá-lo ao ChatGPT na expectativa de obter um **completion** interessante.

Estamos utilizando os seguintes parâmetros da API

- **engine**: O modelo de inteligência artificial que desejamos utilizar. Veja uma descrição daquele que escolhemos, extraída da documentação:

*text-davince-003: Can do any language task with better quality, longer output, and consistent instruction-following than the curie, babbage, or ada models. Also supports inserting completions within text.*

- **max\_tokens**: O número máximo de tokens que o completion produzido pode conter. É importante configurar este parâmetro pois seu valor padrão é 16, ou seja, apenas 16 tokens. Algo em torno de  $16 * 4 = 64$  letras.

- **prompt**: o prompt a partir do qual o completion será gerado.

Veja mais informações sobre os modelos visitando o Link 2.4.1.

## Link 2.4.1

<https://platform.openai.com/docs/models/gpt-3-5>

Observe que o resultado é probabilístico. Diferentes resultados são possíveis. Uns com maior probabilidade e outros com menor probabilidade. Assim, o resultado é uma coleção de valores chamada **choices**. Para pegar aquela de maior probabilidade, acessamos a sua posição zero. A coleção é repleta de objetos do tipo **OpenAIObject**, os quais possuem propriedades como **logprobs**, **text** etc. No momento, o que nos interessa é a propriedade **text**. Utilizamos a função **strip** do Python para remover eventuais espaços em branco, tabs etc do começo e do final da resposta. Veja o Código 2.4.3.

## Código 2.4.3

```
import openai
def criar_pergunta(
    OPENAI_API_KEY,
    assunto,
    tipo,
    dificuldade,
    pergunta_exemplo = None
):
    openai.api_key = OPENAI_API_KEY
    assunto = f'Elabore uma pergunta sobre {assunto}'
    tipo = f'Ela deve ser {tipo}' + ' com 4 alternativas' if tipo ==
'Alternativa' else ''
    dificuldade = f'Seu nível de dificuldade deve ser {dificuldade}'
    pergunta_exemplo = f'Utilize esta pergunta como exemplo:
{pergunta_exemplo}' if pergunta_exemplo != None and pergunta_exemplo
!= '\n' else ''
    prompt = f'{assunto}.{tipo}.{dificuldade}.{pergunta_exemplo}'

    resposta = openai.Completion.create(
        engine='text-davinci-003',
        prompt = prompt,
        max_tokens=150
    )
    return resposta.choices[0].text.strip()
```

Vá ao arquivo **app.py** e faça um teste como no Código 2.4.4.

Código 2.4.4

```
import os
from dotenv import load_dotenv
import chatgpt
load_dotenv()
OPENAI_API_KEY = os.getenv('OPENAI_API_KEY')
print(
    chatgpt.criar_pergunta(
        OPENAI_API_KEY,
        'java',
        'alternativa',
        'médio'
    )
)
```

Execute com

**python app.py**

Teste também usando uma pergunta de exemplo, como no Código 2.4.5.

## Código 2.4.5

```

import os
from dotenv import load_dotenv
import chatgpt
load_dotenv()
OPENAI_API_KEY = os.getenv('OPENAI_API_KEY')

pergunta_exemplo = 'import javax.swing.JOptionPane; public class
SomatorioWhile{ public static void main (String [] args){ int n =
Integer.parseInt(JOptionPane.showInputDialog("Digite um número"));
int i = 1; int soma = 0; while (i <= n){ soma = soma + i % 2 == 0 ? i
: 0; i = i + 1; } JOptionPane.showMessageDialog(null, soma); } } I> 0
programa calcula a soma dos números inteiros pares no intervalo
fechado [0, n]. II. Se o usuário digitar um valor ímpar qualquer, o
programa produzirá o valor zero. III. Se o usuário digitar o valor 2,
o programa produzirá o valor 2. É correto apenas o que se afirma em
a) I b) II c) III d) I e II e) II e III'

print(
    chatgpt.criar_pergunta(
        OPENAI_API_KEY,
        'java',
        'alternativa',
        'difícil',
        pergunta_exemplo
    )
)

```

**Nota.** A Figura 2.4.1 mostra o código Java utilizando como exemplo corretamente indentado, caso deseje visualizar a pergunta.

Figura 2.4.1

```

1  import javax.swing.JOptionPane;
2  public class SomatorioWhile{
3      public static void main (String [] args){
4          int n = Integer.parseInt(JOptionPane.showInputDialog("Digite um número"));
5          int i = 1;
6          int soma = 0;
7          while (i <= n){
8              soma = soma + i % 2 == 0 ? i : 0;
9              i = i + 1;
10         }
11         JOptionPane.showMessageDialog(null, soma);
12     }
13 }

```

Depois de fazer seus testes, **apague o código de teste.**

A Figura 2.4.2 mostra um exemplo de resultado esperado.

Figura 2.4.2

```
(venv) (base) rodrigo@bossini-insp-5502:~/workspaces/producao_de_aulas/python/chatgpt_tkinter_geracao_e_correcao_
• de_questoes$ python app.py
Qual dos seguintes componentes da linguagem Java é usado para criar variáveis e métodos?
a) Classe
b) Objeto
c) Pacote
d) Interface
```

De volta ao arquivo **chatgpt.py**, vamos escrever uma função que solicita ao ChatGPT que responda uma pergunta. Seus parâmetros são:

- Chave de acesso à API do ChatGPT
- Pergunta a ser respondida

Observe no Código 2.4.3 como ajustamos o texto para tornar o prompt mais apropriado e compreensível.



## Código 2.4.3

```

import openai

def criar_pergunta(
    OPENAI_API_KEY,
    assunto,
    tipo,
    dificuldade,
    pergunta_exemplo = None
):
    openai.api_key = OPENAI_API_KEY
    assunto = f'Elabore uma pergunta sobre {assunto}'
    tipo = f'Ela deve ser {tipo}' + ' com 4 alternativas' if tipo ==
'Alternativa' else ''
    dificuldade = f'Seu nível de dificuldade deve ser {dificuldade}'
    pergunta_exemplo = f'Utilize esta pergunta como exemplo:
{pergunta_exemplo}' if pergunta_exemplo != None and pergunta_exemplo
!= '\n' else ''
    prompt = f'{assunto}.{tipo}.{dificuldade}.{pergunta_exemplo}'
    resposta = openai.Completion.create(
        engine='text-davinci-003',
        prompt = prompt,
        max_tokens=150
    )
    return resposta.choices[0].text.strip()

def responder_pergunta(
    OPENAI_API_KEY,
    pergunta_a_ser_respondida
):
    openai.api_key = OPENAI_API_KEY
    prompt = f'Responda a seguinte pergunta:{pergunta_a_ser_respondida}'

```

A seguir, apenas enviamos o prompt ao ChatGPT e tratamos o completion gerado por ele. Veja o Código 2.4.4.

## Código 2.4.4

```

...
def responder_pergunta(
    OPENAI_API_KEY,
    pergunta_a_ser_respondida
):
    openai.api_key = OPENAI_API_KEY
    prompt = f'Responda a seguinte pergunta:{pergunta_a_ser_respondida}'
    resposta = openai.Completion.create(
        engine='text-davinci-003',
        prompt = prompt,
        max_tokens = 150
    )
    return resposta.choices[0].text.strip()

```

No arquivo **app.py**, façamos um teste da nova função. Veja o Código 2.4.5.

## Código 2.4.5

```

import os
from dotenv import load_dotenv
import chatgpt
load_dotenv()
OPENAI_API_KEY = os.getenv('OPENAI_API_KEY')

pergunta = 'import javax.swing.JOptionPane; public class
SomatorioWhile{ public static void main (String [] args){ int n =
Integer.parseInt(JOptionPane.showInputDialog("Digite um número"));
int i = 1; int soma = 0; while (i <= n){ soma = soma + i % 2 == 0 ? i
: 0; i = i + 1; } JOptionPane.showMessageDialog(null, soma); } } I> O
programa calcula a soma dos números inteiros pares no intervalo
fechado [0, n]. II. Se o usuário digitar um valor ímpar qualquer, o
programa produzirá o valor zero. III. Se o usuário digitar o valor 2,
o programa produzirá o valor 2. É correto apenas o que se afirma em
a) I b) II c) III d) I e II e) II e III'

print(chatgpt.responder_pergunta(OPENAI_API_KEY, pergunta))

```

**Apague seus testes quando terminar.**

**2.5 (Desenvolvendo a interface gráfica com Tkinter: Introdução)** Tkinter é uma biblioteca que permite a criação de interfaces gráficas em Python. Trata-se de uma interface para que possamos acessar a biblioteca Tk em Python. Por sua vez, Tk é uma biblioteca gráfica que pode ser utilizada com diferentes linguagens de programação como

- Tcl (originalmente desenvolvida para essa linguagem) <https://www.tcl.tk/>
- Python (Tkinter) <https://docs.python.org/3/library/tkinter.html>
- Java (Jython, permite a execução de código Python em Java) <https://www.jython.org/>
- Lisp (LispWorks) <http://www.lispworks.com/>

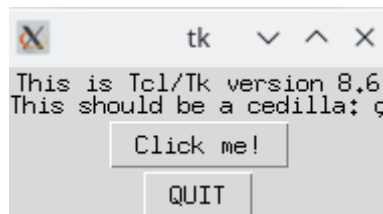
**Nota.** O nome Tkinter vem de **Tk Interface**.

Em seu terminal, execute

**python -m tkinter**

para certificar-se de que a biblioteca está instalada em seu ambiente. O resultado esperado se parece com aquele exibido pela Figura 2.5.1.

Figura 2.5.1



**2.6 (Hello World com Tkinter)** Começamos escrevendo uma aplicação Hello, World com a Tkinter. Para tal, crie um arquivo chamado **tela.py** na raiz de seu projeto.

Começamos importando a **tkinter**, pois ela dá **acesso aos componentes visuais de interesse (widgets)**.

A seguir, importamos o módulo **ttk**. Segundo a sua documentação, ele permite que o comportamento de um componente seja separado de sua aparência. Por meio deste módulo, **podemos trocar o tema** (coleção de cores) utilizado na exibição dos componentes. Em particular, podemos acessar os novos componentes disponibilizados a partir da **versão 8.5** da Tkinter.

A função **Tk** constrói um componente **“top level”**. Ele representa a tela a que adicionaremos novos widgets.

Por meio do módulo **ttk**, construímos um componente **Button** e, usando o método **grid**, o adicionamos à raiz. Observe que o método **Button** recebe o componente de quem o botão será filho, além de propriedades de interesse para o botão.

O método **mainloop** dispara a “thread principal de execução”. Ela é responsável por lidar com os eventos gerados pelo usuário.

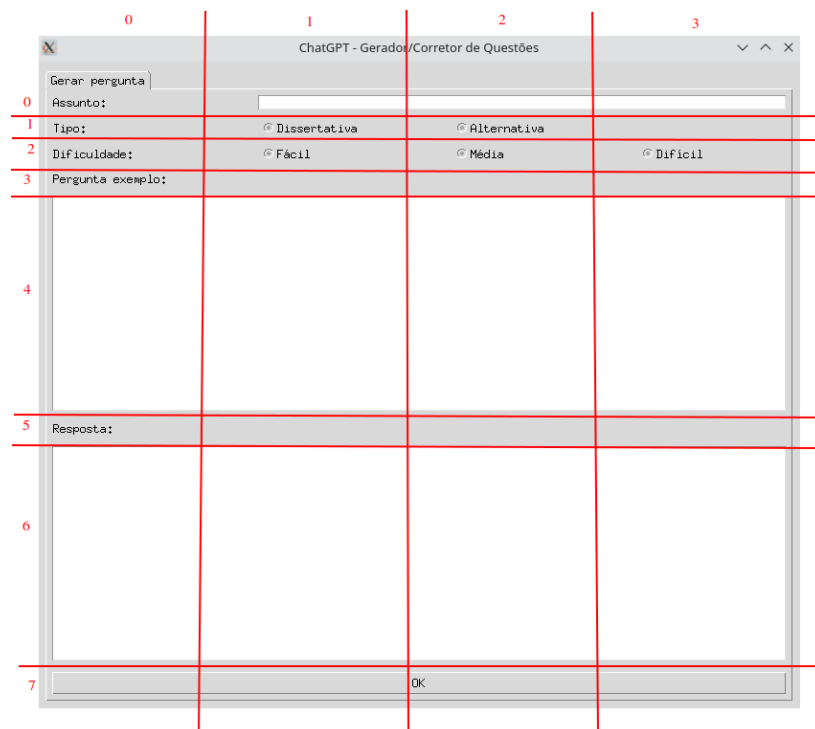
Veja o Código 2.6.1.

Código 2.6.1

```
from tkinter import *
from tkinter import ttk
root = Tk()
ttk.Button(root, text="Hello, World").grid()
root.mainloop()
```

**2.7 (Interface gráfica da aplicação: disposição dos Widgets com um grid)** Os Widgets de nosso aplicativo serão dispostos na tela usando o gerenciador de layout “grid” da tkinter. Veja a Figura 2.7.1.

Figura 2.7.1



**Nota.** Muitos Widgets farão uso do parâmetro **sticky**. Ele representa em quais cantos o widget “cola” quando a tela for expandida. N, S, E, W significam, respectivamente, North, South, East e West. Veja a Figura 2.8.4.

Figura 2.7.2

ChatGPT - Gerador/Corretor de Questões

Gerar pergunta

Assunto:

Tipo: ☐ Dissertativa ☐ Alternativa

Dificuldade: ☐ Fácil ☐ Média ☐ Difícil

Pergunta exemplo:

Resposta:

OK

N

W

E

S

Observe que **eles farão mais sentido depois de configurarmos o peso de cada linha e de cada coluna, o que será feito no final da criação da aba**. Depois disso, vamos realizar alguns testes alterando os valores.

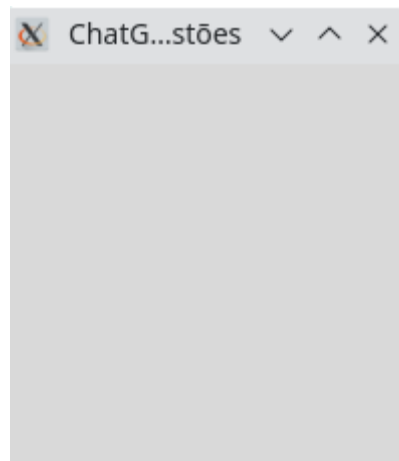
**2.8 (Interface da aplicação: a primeira aba)** Começamos criando a tela principal e dando a ela um título, como no Código 2.8.1. Estamos no arquivo **tela.py**.

Código 2.8.1

```
import tkinter as tk
# Cria a janela principal
window = tk.Tk()
window.title('ChatGPT - Gerador/Corretor de Questões')
# Inicia o loop principal
window.mainloop()
```

Como a Figura 2.8.1 mostra, o resultado ainda não tem muita graça.

Figura 2.8.1



Para adicionar abas à aplicação, utilizamos um Widget do tipo **Notebook**. Um **notebook** serve de contêiner para múltiplos Frames e, neste contexto, cada Frame representa uma **aba**. Há dois parâmetros que merecem explicação:

- **expand=True**: caso o usuário expanda a tela, novo espaço será alocado ao Notebook para que ele possa aumentar de tamanho também, muito embora ainda não faça isso
- **fill=both**: o Notebook deve ocupar espaço nos dois eixos. Tente as opções 'x', 'y' e 'none' também.

Veja o Código 2.8.2.

## Código 2.8.2

```
import tkinter as tk
from tkinter import ttk

# Cria a janela principal
window = tk.Tk()
window.title('ChatGPT - Gerador/Corretor de Questões')

# # Cria o notebook (o container para as abas)
# #padding é a medida entre as bordas da janela e o notebook
notebook = ttk.Notebook(window, padding=10)

# Coloca o notebook na janela
#expand serve que mais espaço seja reservado para o notebook caso o usuário expanda a
tela
# fill both serve para dizer que o componente deve ocupar o espaço novo tanto na
horizontal quanto na vertical. Também poderia ser x, y ou none
notebook.pack(expand=True, fill='both')

# Inicia o loop principal
window.mainloop()
```

Como mostra a Figura 2.8.2, o resultado continua sem graça.

Figura 2.8.2



Agora vamos escrever uma função cuja finalidade será criar os componentes para a primeira aba. Ela começa criando um Frame que representa justamente a primeira aba. A função recebe o Notebook, já que a aba precisa ser colocada dentro dele. Veja o Código 2.8.3.

**Nota.** Depois desse ajuste, teste novamente variando os valores de expand e fill.

Código 2.8.3

```
import tkinter as tk
from tkinter import ttk

#função para criar a primeira aba
def criar_abal(notebook):
    tab1 = ttk.Frame(notebook)
    notebook.add(tab1, text='Gerar pergunta')
    return tab1

window = tk.Tk()
window.title('ChatGPT - Gerador/Corretor de Questões')

notebook = ttk.Notebook(window, padding=10)

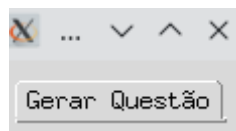
#cria a primeir aba
tab1 = criar_abal(notebook)

notebook.pack(expand=True, fill='both')

window.mainloop()
```

Agora o resultado visual já está mais divertido. Veja a Figura 2.8.3.

Figura 2.8.3





A **linha 0** do grid possui dois Widgets.

**Label:** para exibir o texto não editável “Assunto:”

**Entry:** para permitir que o usuário digite algo.

Veja o Código 2.8.4.

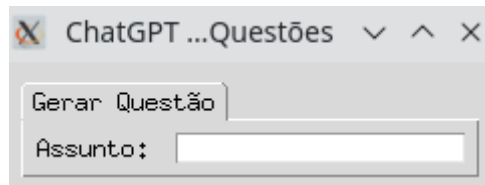
Código 2.8.4

```
...
#função para criar a primeira aba
def criar_abal(notebook):
    tab1 = ttk.Frame(notebook)
    notebook.add(tab1, text='Gerar pergunta')

    #linha 0
    tk.Label(tab1, text='Assunto:').grid(row=0, column=0, sticky='W', padx=5, pady=5)
    assunto_entry = tk.Entry(tab1)
    assunto_entry.grid(row=0, column=1, columnspan=3, sticky='WE', padx=5, pady=5)
    return tab1
...
```

O resultado esperado aparece na Figura 2.8.4.

Figura 2.8.4



Na **linha 1**, teremos

**Label:** para exibir o texto não editável “Tipo”.

**Radiobutton:** são dois, para exibir “Dissertativa” e “Alternativa”

Observe que também declaramos um **StringVar**. Ele servirá para armazenar a opção selecionada para o usuário.

Os parâmetros **text** e **value** de um Radiobutton representam, respectivamente, o texto apresentado na tela e o valor que será de fato utilizado caso o usuário selecione aquela opção.

Veja o Código 2.8.5.

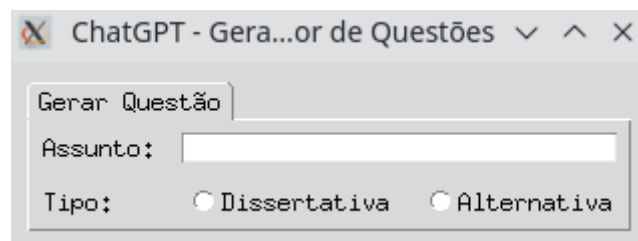
Código 2.8.5

```
def criar_abal(notebook):
    tab1 = ttk.Frame(notebook)
    notebook.add(tab1, text='Gerar pergunta')
    #linha 0
    tk.Label(tab1, text='Assunto:').grid(row=0, column=0, sticky='W', padx=5, pady=5)
    assunto_entry = tk.Entry(tab1)
    assunto_entry.grid(row=0, column=1, columnspan=3, sticky='WE', padx=5, pady=5)

    #linha 1
    tk.Label(tab1, text='Tipo:').grid(row=1, column=0, sticky='W', padx=5, pady=5)
    tipo_var = tk.StringVar()
    tk.Radiobutton(tab1, text='Dissertativa', variable=tipo_var, value='Dissertativa').grid(row=1,
column=1, sticky='W', padx=5, pady=5)
    tk.Radiobutton(tab1, text='Alternativa', variable=tipo_var, value='Alternativa').grid(row=1,
column=2, sticky='W', padx=5, pady=5)
    return tab1
```

A Figura 2.8.5 mostra o resultado esperado.

Figura 2.8.5



Na **linha 2**, teremos

**Label:** para exibir o texto não editável **Dificuldade**

**Radiobutton:** São três, para exibir “Fácil”, “Média” e “Difícil”

Observe que também temos um **StringVar**, também responsável por armazenar o que o usuário escolher.

Veja o Código 2.8.6 e a Figura 2.8.6.

Código 2.8.6

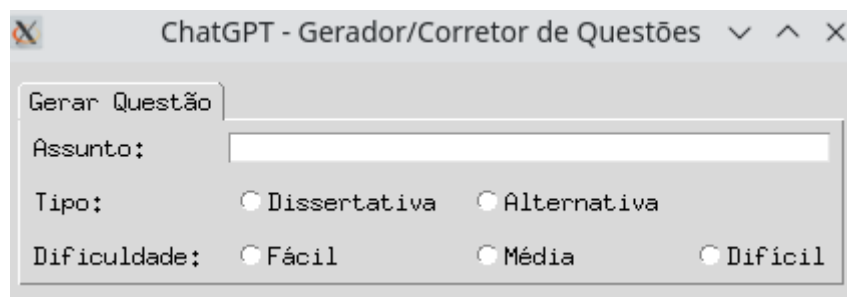
```
#função para criar a primeira aba
def criar_abal(notebook):
    tab1 = ttk.Frame(notebook)
    notebook.add(tab1, text='Gerar pergunta')
    #linha 0
    tk.Label(tab1, text='Assunto:').grid(row=0, column=0, sticky='W', padx=5,
pady=5)
    assunto_entry = tk.Entry(tab1)
    assunto_entry.grid(row=0, column=1, columnspan=3, sticky='WE', padx=5, pady=5)

    #linha 1
    tk.Label(tab1, text='Tipo:').grid(row=1, column=0, sticky='W', padx=5, pady=5)
    tipo_var = tk.StringVar()
    tk.Radiobutton(tab1, text='Dissertativa', variable=tipo_var,
value='Dissertativa').grid(row=1, column=1, sticky='W', padx=5, pady=5)
    tk.Radiobutton(tab1, text='Alternativa', variable=tipo_var,
value='Alternativa').grid(row=1, column=2, sticky='W', padx=5, pady=5)

    #linha 2
    tk.Label(tab1, text='Dificuldade:').grid(row=2, column=0, sticky='W', padx=5,
pady=5)
    dificuldade_var = tk.StringVar()
    tk.Radiobutton(tab1, text='Fácil', variable=dificuldade_var,
value='Fácil').grid(row=2, column=1, sticky='W', padx=5, pady=5)
    tk.Radiobutton(tab1, text='Média', variable=dificuldade_var,
value='Média').grid(row=2, column=2, sticky='W', padx=5, pady=5)
    tk.Radiobutton(tab1, text='Difícil', variable=dificuldade_var,
value='Difícil').grid(row=2, column=3, sticky='W', padx=5, pady=5)

    return tab1
```

Figura 2.8.6



A linha 3 terá

**Label:** para exibir o texto fixo “Pergunta exemplo”.

Veja o Código 2.8.7 e a Figura 2.8.7.

Código 2.8.7

```
...
#linha 2
tk.Label(tab1, text='Dificuldade:').grid(row=2, column=0, sticky='W',
padx=5, pady=5)
dificuldade_var = tk.StringVar()
tk.Radiobutton(tab1, text='Fácil', variable=dificuldade_var,
value='Fácil').grid(row=2, column=1, sticky='W', padx=5, pady=5)
tk.Radiobutton(tab1, text='Média', variable=dificuldade_var,
value='Média').grid(row=2, column=2, sticky='W', padx=5, pady=5)
tk.Radiobutton(tab1, text='Difícil', variable=dificuldade_var,
value='Difícil').grid(row=2, column=3, sticky='W', padx=5, pady=5)

#linha 3
tk.Label(tab1, text='Pergunta exemplo:').grid(row=3, column=0, sticky='W',
padx=5, pady=5)
return tab1
```

Figura 2.8.7

ChatGPT - Gerador/Corretor de

Gerar pergunta

Assunto:

Tipo: ☐ Dissertativa ☐ Alternativa

Dificuldade: ☐ Fácil ☐ Média ☐ Difícil

Pergunta exemplo:

A linha 4 terá

**Text:** para permitir que o usuário informe uma pergunta de exemplo.

Veja o Código 2.8.8 e a Figura 2.8.8.

Código 2.8.8

```
...
#linha 3
tk.Label(tab1, text='Pergunta exemplo:').grid(row=3, column=0, sticky='W',
padx=5, pady=5)

#linha 4
pergunta_exemplo = tk.Text(tab1)
#20 linhas de texto
pergunta_exemplo.configure(height=20)
pergunta_exemplo.grid(row=4, column=0, columnspan=4, sticky='WE', padx=5,
pady=5)
return tab1
```

Figura 2.8.8

ChatGPT - Gerador/Corretor de Questões

Gerar pergunta

Assunto:

Tipo: ☐ Dissertativa ☐ Alternativa

Dificuldade: ☒ Fácil ☐ Média ☐ Difícil

Pergunta exemplo:

A linha 5 terá

**Label:** para exibir o texto não editável "Resposta:"

Veja o Código 2.8.9 e a Figura 2.8.9.

Código 2.8.9

```
...
#linha 4
pergunta_exemplo = tk.Text(tab1)
#20 linhas de texto
pergunta_exemplo.configure(height=20)
pergunta_exemplo.grid(row=4, column=0, columnspan=4, sticky='WE', padx=5,
pady=5)

#linha 5
tk.Label(tab1, text='Resposta:').grid(row=5, column=0, sticky='W', padx=5,
pady=5)
return tab1
...
```

Figura 2.8.9

The screenshot shows a window titled "ChatGPT - Gerador/Corretor de Questões". It features a tab labeled "Gerar pergunta". Below the tab, there are several input fields and radio buttons:

- Assunto:** A text input field.
- Tipo:** Two radio buttons labeled "Dissertativa" and "Alternativa".
- Dificuldade:** Three radio buttons labeled "Fácil", "Média", and "Difícil".
- Pergunta exemplo:** A large text area for input.
- Resposta:** A text input field at the bottom.

A linha 6 terá

**Text:** utilizado para exibir a resposta do ChatGPT. Veja o Código 2.8.10 e a Figura 2.8.10.

Código 2.8.10

```
#linha 5
tk.Label(tab1, text='Resposta:').grid(row=5, column=0, sticky='W', padx=5,
pady=5)

#linha 6
resposta_text = tk.Text(tab1)
#20 linhas de texto
resposta_text.configure(height=20)
resposta_text.grid(row=6, column=0, columnspan=4, sticky='WE', padx=5,
pady=5)
return tab1
```

Figura 2.8.10

The screenshot shows a window titled "ChatGPT - Gerador/Corretor de Questões". It features a tabbed interface with the "Gerar pergunta" tab selected. The form includes the following elements:

- Assunto:** A text input field.
- Tipo:** Radio buttons for "Dissertativa" and "Alternativa".
- Dificuldade:** Radio buttons for "Fácil", "Média", and "Difícil".
- Pergunta exemplo:** A large text area for input.
- Resposta:** A large text area for output.

A linha 7 terá

**Button:** quando clicado, ele acionará o chatgpt a fim de obter uma pergunta de acordo com as especificações do usuário. Veja o Código 2.8.11 e a Figura 2.8.11.

Código 2.8.11

```
...  
  
#linha 6  
resposta_text = tk.Text(tab1)  
#20 linhas de texto  
resposta_text.configure(height=20)  
resposta_text.grid(row=6, column=0, columnspan=4, sticky='WE', padx=5,  
pady=5)  
  
#linha 7  
ok_button = tk.Button(tab1, text='OK')  
ok_button.grid(row=7, column=0, columnspan=4, sticky='WE', padx=5, pady=5)  
return tab1  
  
...
```



Figura 2.8.11

ChatGPT - Gerador/Corretor de Questões

Gerar pergunta

Assunto:

Tipo: ☐ Dissertativa ☐ Alternativa

Dificuldade: ☐ Fácil ☐ Média ☐ Difícil

Pergunta exemplo:

Resposta:

OK

Expanda um pouco a tela e verifique que os componentes permanecem fixos, como na Figura 2.8.12.

Figura 2.8.12

ChatGPT - Gerador/Corretor de Questões

Gerar pergunta

Assunto:

Tipo: ☐ Dissertativa ☐ Alternativa

Dificuldade: ☐ Fácil ☐ Média ☐ Difícil

Pergunta exemplo:

Resposta:

OK

Para ajustar isso, basta indicarmos qual a proporção a ser atribuída **a cada linha e a cada coluna do grid**. Atribuímos peso igual a 1 para todas. Como temos 4 colunas, cada uma terá 1/4 da tela. Como temos 8 linhas, cada uma terá 1/8 da tela.

Veja o Código 2.8.12.

Código 2.8.12

```

...

#linha 7
ok_button = tk.Button(tab1, text='OK')
ok_button.grid(row=7, column=0, columnspan=4, sticky='WE', padx=5, pady=5)

#peso 1 para cada linha conforme o usuário expande
for i in range(8): # para 8 linhas (0-7)
    tab1.grid_rowconfigure(i, weight=1) # expande com peso 1
#peso 1 para cada coluna conforme o usuário expande
for i in range(4): # para 4 colunas (0-3)
    tab1.grid_columnconfigure(i, weight=1) # expande com peso 1
return tab1

...

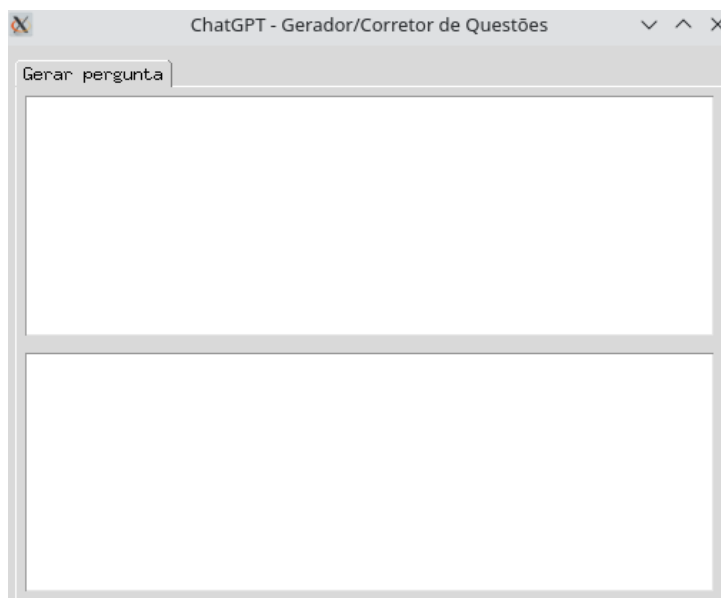
```

Expanda novamente a tela e veja que os componentes agora a acompanham.

Agora faz sentido testar os valores atribuídos ao parâmetro **sticky**. Troque alguns de EW para W etc. Faça seus experimentos e se familiarize com a biblioteca!

Observe também que, se o usuário reduzir muito o tamanho da tela, os widgets somem, como na Figura 2.8.13.

Figura 2.8.13



Para evitar isso, configure um tamanho mínimo para a janela como no Código 2.8.13.

## Código 2.8.13

```
...
window = tk.Tk()
window.title('ChatGPT - Gerador/Corretor de Questões')
window.minsize(800, 600)
...
```

Por fim, vamos ajustar o módulo tela.py para que ele apenas defina uma função que cria a tela. Assim, ele pode ser importado e seu cliente pode chamar a função quando desejar. Observe que ela recebe Veja o Código 2.8.14.

## Código 2.8.14

```
import tkinter as tk
from tkinter import ttk

#função para criar a primeira aba
def criar_abal(notebook):
    ...
    return tab1

def criar_tela():
    window = tk.Tk()
    window.title('ChatGPT - Gerador/Corretor de Questões')
    window.minsize(800, 600)

    notebook = ttk.Notebook(window, padding=10)

    #cria a primeir aba
    tab1 = criar_abal(notebook)
    notebook.pack(expand=True, fill='both')
    window.mainloop()
```

**2.9 Integrando tudo** O módulo principal da aplicação deve agora importar os módulos **chatgpt** e **tela** e realizar a integração.

Para criar a tela, ele espera passar uma função como parâmetro. Ela será associada ao botão que, quando clicado, deve acionar o chatgpt. Veja o Código 2.9.1. Estamos no arquivo **app.py**.

Código 2.9.1

```
import os
from dotenv import load_dotenv
import chatgpt
import tela
load_dotenv()
OPENAI_API_KEY = os.getenv('OPENAI_API_KEY')

def criar_pergunta(assunto, tipo, dificuldade, pergunta_exemplo):
    return chatgpt.criar_pergunta(
        OPENAI_API_KEY,
        assunto,
        tipo,
        dificuldade,
        pergunta_exemplo
    )

tela.criar_tela(criar_pergunta)
```

Claro, a assinatura da função **criar\_tela** deve ser atualizada para que ela admita receber a função. Observe que, por sua vez, ela passa a função recebida como parâmetro para a função **criar\_aba1** a fim de que ela possa associá-la ao botão. Veja o Código 2.9.2. Estamos agora no arquivo **tela.py**.

## Código 2.9.2

```
def criar_tela(criar_pergunta):
    window = tk.Tk()
    window.title('ChatGPT - Gerador/Corretor de Questões')
    window.minsize(800, 600)

    notebook = ttk.Notebook(window, padding=10)

    #cria a primeir aba
    tab1 = criar_abal(notebook, criar_pergunta)
    notebook.pack(expand=True, fill='both')
    window.mainloop()
```

Ajustamos agora a função `criar_abal` para que ela receba a função e a associe ao botão. Observe que a função precisa ser chamada com alguns parâmetros, os quais devem ser extraídos dos componentes visuais com os quais o usuário interage. Por isso, vamos definir uma função que opera sobre eles pegando os dados e então chama a função recebida. Veja o Código 2.9.3. Ainda estamos no arquivo **tela.py**.

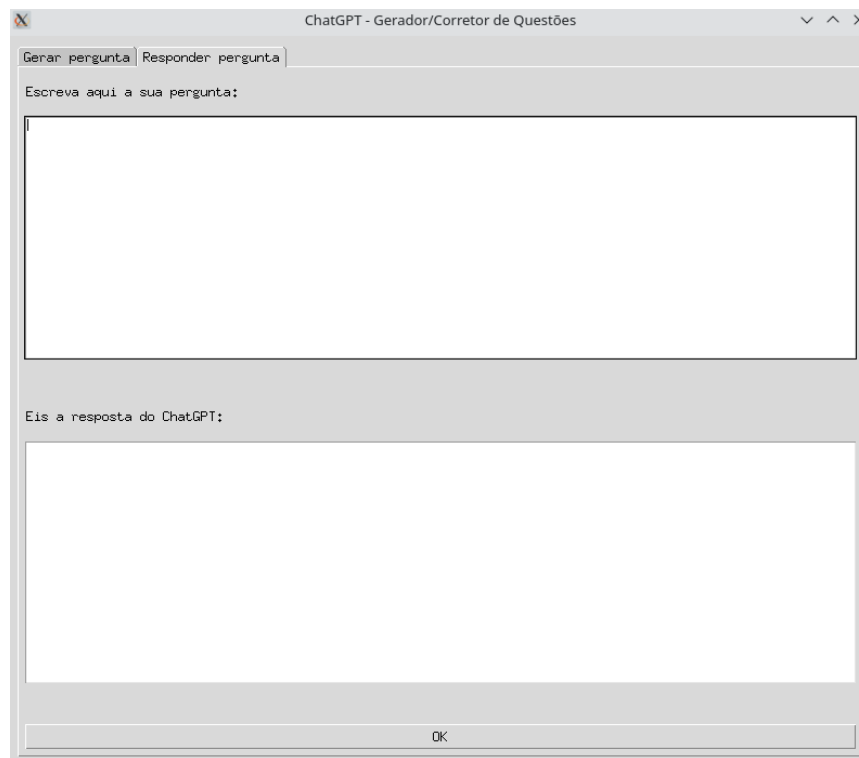
## Código 2.9.3

```
def criar_abal(notebook, criar_pergunta):
    ...
    #linha 7
    def executar():
        #num Text TKinter '1.0' representa número da linha (começa do 1) e da
        #coluna (começa do zero)
        #end é uma constante especial que representa "até o último caractere"
        #limpamos para casos de perguntas anteriores
        resposta_text.delete("1.0", 'end')
        resposta = criar_pergunta(assunto_entry.get(), tipo_var.get(),
        dificuldade_var.get(), pergunta_exemplo.get('1.0', 'end'))
        resposta_text.insert("1.0", resposta)
    ok_button = tk.Button(tab1, text='OK', command=executar)
    ok_button.grid(row=7, column=0, columnspan=4, sticky='WE', padx=5, pady=5)

    #peso 1 para cada linha conforme o usuário expande
    for i in range(8): # para 8 linhas (0-7)
        tab1.grid_rowconfigure(i, weight=1) # expande com peso 1
    #peso 1 para cada coluna conforme o usuário expande
    for i in range(4): # para 4 colunas (0-3)
        tab1.grid_columnconfigure(i, weight=1) # expande com peso 1
    return tab1
```

**2.10 (A segunda aba)** A segunda aba permitirá que enviemos perguntas ao ChatGPT para que ele as responda. Veja a Figura 2.10.1.

Figura 2.10.1



A função que define a segunda aba é semelhante àquela que define a primeira. Veja o Código 2.10.1. Estamos no arquivo **tela.py**.

## Código 2.10.1

```

import tkinter as tk
from tkinter import ttk

def criar_aba2 (notebook, responder_pergunta):
    tab2 = ttk.Frame(notebook)
    notebook.add(tab2, text='Responder pergunta')
    #linha 0
    tk.Label(tab2, text='Escreva aqui a sua pergunta:').grid(row=0,
column=0, sticky='W', padx=5, pady=0)
    #linha 1
    pergunta_text = tk.Text(tab2)
    #20 linhas de texto
    pergunta_text.configure(height=20)
    pergunta_text.grid(row=1, column=0, sticky='WEN', padx=5, pady=0)

    #linha 2
    tk.Label(tab2, text='Eis a resposta do ChatGPT:').grid(row=2,
column=0, sticky='W', padx=5, pady=0)
    #linha 3
    resposta_text = tk.Text(tab2)
    #20 linhas de texto
    resposta_text.configure(height=20)
    resposta_text.grid(row=3, column=0, sticky='WEN', padx=5, pady=0)

    #linha 4
    def executar():
        resposta = responder_pergunta( pergunta_text.get('1.0', 'end'))
        resposta_text.delete('1.0', 'end')
        resposta_text.insert('1.0', resposta)
    ok_button = tk.Button(tab2, text='OK', command=executar)
    ok_button.grid(row=4, column=0, sticky='WE', padx=5, pady=5)
    ...

```

Ainda no arquivo **tela.py**, a função **criar\_tela** a utiliza como mostra o Código 2.10.2.



## Código 2.10.2

```
def criar_tela(criar_pergunta, responder_pergunta):
    window = tk.Tk()
    window.title('ChatGPT - Gerador/Corretor de Questões')
    window.minsize(800, 600)

    notebook = ttk.Notebook(window, padding=10)

    #cria a primeir aba
    tab1 = criar_aba1(notebook, criar_pergunta)
    #cria a segunda aba
    tab2 = criar_aba2(notebook, responder_pergunta)
    notebook.pack(expand=True, fill='both')
    window.mainloop()
```

No arquivo **app.py**, definimos uma função que utiliza o ChatGPT e a entregamos como parâmetro para a função `criar_tela`, como no Código 2.10.3.

## Código 2.10.3

```
import os
from dotenv import load_dotenv
import chatgpt
import tela

load_dotenv()
OPENAI_API_KEY = os.getenv('OPENAI_API_KEY')

def criar_pergunta(assunto, tipo, dificuldade, pergunta_exemplo):
    return chatgpt.criar_pergunta(
        OPENAI_API_KEY,
        assunto,
        tipo,
        dificuldade,
        pergunta_exemplo
    )

def responder_pergunta (pergunta_a_ser_respondida):
    return chatgpt.responder_pergunta(
        OPENAI_API_KEY,
        pergunta_a_ser_respondida
    )

tela.criar_tela(criar_pergunta, responder_pergunta)
```

### ***Referências***

**OpenAI.** OpenAI, 2023. Disponível em <<https://openai.com/>>. Acesso em maio de 2023.