```
import tkinter as tk
from tkinter import messagebox
import json
import os
```

Esse trecho importa módulos do Python usados para criar interfaces gráficas e manipular arquivos JSON

- **import tkinter as tk** - importa a biblioteca Tkinter, usada para criar interfaces gráficas no Python, o 'alias' tk facilita a referência ao módulo
- **from tkinter import messagebox** - importa a funcionalidade messagebox do Tkinter, que exibe caixas de diálogo com mensagens - alertas e confirmações
- Import json - importa o módulo JSON, que permite salvar e carregar arquivos no formato JSON
- **import os** - importa o módulo os - usado para interagir com o sistema operacional, como manipular arquivos e diretórios

```
#Nome do arquivo onde os dados serão salvos
ARQUIVO_JSON = "lista_compras.json"

#Lista de compras (dicionário: item -> [quantidade, preço])
lista_compras = {}
```

- **ARQUIVO_JSON** = "lista_compras.json" - define o nome do arquivo onde os dados da lista de compras serão armazenados em formato JSON
- **lista_compras = {}** - inicializa um dicionário vazio para armazenar os itens da lista de compras

```
#Função para salvar a lista em JSON
def salvar_lista():
    with open(ARQUIVO_JSON, "w") as file:
        json.dump(lista_compras,file,indent=4)

#Função para carregar a lista do arquivo JSON
def carregar_lista():
    global lista_compras
    if os.path.exists(ARQUIVO_JSON): #Verifica se o arquivo existe
        with open(ARQUIVO_JSON, "r") as file:
            lista_compras = json.load(file)
    atualizar_lista()
```

- **def salvar_lista()**
  - Salvar a lista de compras no arquivo lista_compras.json
  - Abre o arquivo em modo de escrita ("w")
  - Usa json.dump() para converter o dicionário lista_compras em JSON e gravá-lo no arquivo

- O parâmetro indent=4 deixa o JSON formatado com indentação de 4 espaços para melhor leitura.

- **def carregar_lista()**
  - Carregar a lista de compras do arquivo lista_compras.json para o programa
  - Usa os.path.exists(ARQUIVO_JSON) para verificar se o arquivo existe antes de tentar lê-lo
  - Se o arquivo existir:
    - Abre o arquivo em modo de leitura (“r”)
    - Usa json.load() para carregar os dados do JSON de volta para o dicionário lista_compras
  - Chama atualizar_lista(), que provavelmente irá atualizar a interface gráfica com os dados carregados.

```
#Função de atualizar a exibição da lista na interface
def atualizar_lista():
    lista_box.delete(0,tk.END) #Limpa a exibição
    for item, (quantidade, preco) in lista_compras.items():
        lista_box.insert(tk.END, f"{quantidade} x {item} - R$
{preco:.2f}")
```

A função atualizar_lista(), atualiza a exibição da lista de compras na interface gráfica:

- **lista_box.delete(0, tk.END)**
  - Remove todos os itens da exibição da lista antes de adicionar os novos
  - 0, tk. END significa que ele vai apagar do primeiro até o último item da lista_box
- Loop for **item, (quantidade, preco) in lista_compras.items():**
  - Percorre o dicionário lista_compras
  - item - nome do produto
  - (quantidade, preco) - lista contendo a quantidade e o preço do item
- **lista_box.insert(tk.END, f”{quantidade} x {item} - R$ {preco:.2f}”)**
  - Adiciona um novo item à lista_box, que é o widget de exibição da lista
  - tk.END insere o item no final da lista
  - O texto formatado f”{quantidade} x {item} - R$ {preco:.2f}” exibe:
    - A quantidade
    - O nome do produto
    - O preço formatado com duas casas decimais

```
#Função de adicionar item
def adicionar_item():
    item = entrada_item.get().strip().lower()
    if not item:
        messagebox.showerror("Aviso","Digite o nome do item!")
        return

    try:
        quantidade = int(entrada_quantidade.get())
        preco = float(entrada_preco.get())
        if quantidade <= 0 or preco <= 0:
            raise ValueError
    except ValueError:
        messagebox.showerror("Erro","Quantidade e Preço devem ser números
positivos!")
        return


    #Lógica de verificação e adição do item à lista
    if item in lista_compras:
        lista_compras[item][0] += quantidade #Atualiza quantidade
    else:
        lista_compras[item] = [quantidade,preco]

    print(f"{quantidade}x {item} adicionado(s) à lista de compras.")
    salvar_lista() #Salva automaticamente após adicionar um item
    atualizar_lista()
    entrada_item.delete(0, tk.END)
    entrada_quantidade.delete(0, tk.END)
    entrada_preco.delete(0, tk.END)
```

Essa função permite adicionar itens à lista de compras, garantindo que os dados inseridos sejam válidos. Vamos analisar cada parte:

1) Captura e validação do nome do item

```
item = entrada_item.get().strip().lower()
    if not item:
        messagebox.showerror("Aviso","Digite o nome do item!")
        return
```

- Obtém o nome do item do campo entrada_item e remove espaços extras por meio do strip()

- Converte para letras minúsculas lower() para evitar duplicatas com variações maiúsculas/minúsculas
- Se o nome estiver vazio, exibe uma mensagem de erro e interrompe a função com return

2) Captura e validação de quantidade e preço

```
try:
        quantidade = int(entrada_quantidade.get())
        preco = float(entrada_preco.get())
        if quantidade <= 0 or preco <= 0:
            raise ValueError
    except ValueError:
        messagebox.showerror("Erro","Quantidade e Preço devem ser números
positivos!")
        return
```

- Tenta converter os valores de quantidade e preço para números int e float
- Se forem negativos ou zero, levanta um erro e exibe uma mensagem de erro com messagebox.showerror()
- Se a conversão falhar (se o usuário digitar texto em vez de números), o except captura o erro e exibe um aviso.

3) Adiciona ou atualiza o item na lista

```
#Lógica de verificação e adição do item à lista
    if item in lista_compras:
        lista_compras[item][0] += quantidade #Atualiza quantidade
    else:
        lista_compras[item] = [quantidade,preco]
```

- Se o item já existir na lista, apenas soma a nova quantidade à quantidade já existente
- Se o item não estiver na lista, ele é adicionado com a quantidade e preço informados

4) Exibe uma confirmação no console

```
print(f"{quantidade}x {item} adicionado(s) à lista de compras.")
```

5) Salva e atualiza a interface

```
salvar_lista() #Salva automaticamente após adicionar um item
atualizar_lista()
```

- Salva a lista de compras no arquivo JSON
- Atualiza e interface gráfica para refletir as mudanças

6) Limpa os campos de entrada

```
entrada_item.delete(0, tk.END)
entrada_quantidade.delete(0, tk.END)
entrada_preco.delete(0, tk.END)
```

- Apaga os valores digitados nos campos de entrada para facilitar a inserção de novos itens

Resumo da função **adicionar_item():**

- Captura o nome, quantidade e preço do item
- Valida se os dados estão corretos
- Atualiza a lista de compras (adicionando ou somando à quantidade)
- Salva no JSON e atualiza a interface
- Limpa os campos de entrada após a adição

```
def remover_item():
    selecionado = lista_box.curselection()
    if not selecionado:
        messagebox.showwarning("Aviso","Selecione um item para remover!")
        return

    item_texto = lista_box.get(selecionado[0])
    item_nome = item_texto.split("x ")[1].split(" - ")[0]

    if item_nome in lista_compras:
        del lista_compras[item_nome]
        salvar_lista()
        atualizar_lista()
```

A função **remover_item()** permite remover um item da lista de compras através da interface gráfica, vamos analisar o que cada parte faz:

1) Captura o item selecionado

```
selecionado = lista_box.curselection()
    if not selecionado:
        messagebox.showwarning("Aviso","Selecione um item para remover!")
        return
```

- **lista_box.curselection()** retorna o índice do item selecionado na interface
- Se nenhum item estiver selecionado, exibe um aviso com **messagebox.showwarning()** e interrompe a função com **return**

2) Extrai o nome do item

```
item_texto = lista_box.get(selecionado[0])
item_nome = item_texto.split("x ")[1].split(" - ")[0]
```

- **lista_box.get(selecionado[0])** pega o texto do item exibido na interface, que tem o formato: 3 x Maçã - R$ 2.50
- **split(“x “)[1]** pega o que vem depois de “x”, resultando em: Maçã - R$ 2.50
- **Split (“ - “)[0]** pega apenas o nome do item, removendo o preço, resultando em: Maçã

3) Remove o item da lista

```
if item_nome in lista_compras:
        del lista_compras[item_nome]
        salvar_lista()
        atualizar_lista()
```

- Verifica se o **item_nome** está no dicionário **lista_compras.**
- Se estiver, usa **del lista_compras[item_nome]** para removê-lo da lista
- Salva a lista atualizada no JSON - **salvar_lista()**
- Atualiza a interface gráfica **atualizar_lista()**

Resumo da função **remover_item()**:

- Verifica se um item foi selecionado
- Extrai o nome do item da interface
- Remove o item do dicionário lista_compras
- Salva a lista e atualiza a interface

```
def preencher_campos():
    selecionado = lista_box.curselection()
    if not selecionado:
        messagebox.showwarning("Aviso","Selecione um item para editar!")
        return

    item_texto = lista_box.get(selecionado[0])
    item_nome = item_texto.split("x ")[1].split(" - ")[0]

    entrada_item.delete(0, tk.END)
    entrada_quantidade.delete(0, tk.END)
    entrada_preco.delete(0, tk.END)

    entrada_item.insert(0, item_nome)
    entrada_quantidade.insert(0, lista_compras[item_nome][0])
    entrada_preco.insert(0, lista_compras[item_nome][1])
```

A função **preencher_campos()** permite ao usuário selecionar um item na interface e alterá-lo, preenchendo os campos dos inputs, vamos analisar passo à passo:

1) Captura o item selecionado

```
selecionado = lista_box.curselection()
    if not selecionado:
        messagebox.showwarning("Aviso","Selecione um item para editar!")
        return
```

- **lista_box.curselection()** retorna o índice do item selecionado na interface
- Se nenhum item estiver selecionado, exibe um aviso com **messagebox.showwarning()** e interrompe a função com **return**

2) Extrai o nome do item

```
item_texto = lista_box.get(selecionado[0])
item_nome = item_texto.split("x ")[1].split(" - ")[0]
```

- **lista_box.get(selecionado[0])** pega o texto do item exibido na interface
- **split("x ")[1]** pega o que vem depois de "x", resultando em: Maçã - R$ 2.50

3) Limpa e reinsere os dados

```
entrada_item.delete(0, tk.END)
entrada_quantidade.delete(0, tk.END)
entrada_preco.delete(0, tk.END)

entrada_item.insert(0, item_nome)
entrada_quantidade.insert(0, lista_compras[item_nome][0])
entrada_preco.insert(0, lista_compras[item_nome][1])
```

- Remove qualquer texto que esteja nos campos de entrada
    - entrada_item
    - entrada_quantidade
    - entrada_preco
- Isso evita que valores antigos interfiram em uma nova adição ou edição

- Reinsere os dados do item removido
    - entrada_item.insert(0, item_nome) - insere o nome do item no campo de nome
    - entrada_quantidade.insert(0, lista_compras[item_nome][0]) - insere a quantidade do item
    - entrada_preco.insert(0, lista_compras[item_nome][1]) - insere o preço do item

Permite que, ao remover um item, os valores sejam automaticamente preenchidos nos campos, facilitando uma possível edição e re-adição do item sem precisar digitá-lo novamente.

```
def editar_item():
    item = entrada_item.get().strip().lower()
    if not item or item not in lista_compras:
        messagebox.showerror("Erro","O item não está na lista!")
        return

    try:
        quantidade = int(entrada_quantidade.get())
        preco = float(entrada_preco.get())
        if quantidade <= 0 or preco <= 0:
            raise ValueError
    except ValueError:
        messagebox.showerror("Erro","Quantidade e Preço devem ser números
positivos!")
        return

    lista_compras[item] = [quantidade, preco]

    salvar_lista()
    atualizar_lista()
    limpar_campos()
```

A função **editar_item()** permite editar um item existente na lista de compras. Vamos analisar o que cada parte faz:

1) Captura e validação do nome do item

```
item = entrada_item.get().strip().lower()
    if not item or item not in lista_compras:
        messagebox.showerror("Erro","O item não está na lista!")
        return
```

- Obtém o nome do item digitado no campo **entrada_item()**
  - Remove espaços extras **strip()**
  - Converte para minúsculas **lower()**
- Se o campo estiver vazio ou se o item não estiver na lista de compras, exibe uma mensagem de erro e interrompe a função com **return()**

2) Captura e validação da quantidade e preço

```
try:
        quantidade = int(entrada_quantidade.get())
        preco = float(entrada_preco.get())
        if quantidade <= 0 or preco <= 0:
            raise ValueError
    except ValueError:
        messagebox.showerror("Erro","Quantidade e Preço devem ser números
positivos!")
        return
```

- Tenta transformar a quantidade e o preço para números int e float
- Se o usuário digitar algo inválido (como texto) um **ValueError** será gerado e tratado
- Se a quantidade ou o preço forem menores ou iguais a zero, levanta um **ValueError** manualmente para garantir que apenas valores positivos sejam aceitos.
    - Em caso de erro, exibe uma mensagem de erro e interrompe a função.

3) Atualiza os valores do item

```
lista_compras[item] = [quantidade, preco]
```

4) Salva e atualiza a interface

```
salvar_lista()
atualizar_lista()
```

5) Limpa os campos de entrada

```
limpar_campos()
```

- Chama a função limpar_campos(), que é definida por:

```
#Limpar os campos de entrada
def limpar_campos():
    entrada_item.delete(0, tk.END)
    entrada_quantidade.delete(0, tk.END)
    entrada_preco.delete(0, tk.END)
```

- Remove qualquer texto que esteja nos campos de entrada
    - entrada_item
    - entrada_quantidade
    - entrada_preco
- Isso evita que valores antigos interfiram em uma nova adição ou edição

```
#Criar a janela principal
janela = tk.Tk()
janela.title("Lista de Compras")
janela.geometry("400x700")
janela.config(bg="#f4f4f9")
```

Essas linhas criam e configuram a janela principal da interface gráfica da aplicação usando a biblioteca Tkinter

- janela = tk.Tk() - cria uma nova instância da janela principal Tk() que servirá como a interface principal do programa
- janela.title(“Lista de Compras”) - define o título da janela, que será exibido na barra superior
- janela.geometry(“400x700”) - define o tamanho inicial da janela para 400px de largura e 700x de altura
- janela.config(bg=”f4f4f9”) - define a cor de fundo da janela para um tom claro

```
#Criar menu superior
menu_bar = tk.Menu(janela)
janela.config(menu=menu_bar)
```

- **menu_bar** cria uma barra de menu (Menu) associada à janela
- **janela.config(menu=menu_bar)** associa a barra de menu (menu_bar) à janela, fazendo com que o menu apareça na interface

```
#Adicionar opções ao menu
arquivo_menu = tk.Menu(menu_bar, tearoff=0)
arquivo_menu.add_command(label="Salvar",command=salvar_lista)
arquivo_menu.add_command(label="Carregar",command=carregar_lista)
arquivo_menu.add_separator()
arquivo_menu.add_command(label="Sair",command=janela.quit)

menu_bar.add_cascade(label="Arquivo",menu=arquivo_menu)
```

```
arquivo_menu = tk.Menu(menu_bar, tearoff=0)
```

- Cria um submenu chamado **arquivo_menu** dentro da **menu_bar**
- O parêmetro **tearoff=0** impede que o menu possa ser destacado e movido separadamente da janela principal

```
arquivo_menu.add_command(label="Salvar",command=salvar_lista)
arquivo_menu.add_command(label="Carregar",command=carregar_lista)
```

- Adiciona a opção “Salvar” que chama a função **salvar_lista()** para guardar os dados no JSON
- Adiciona a opção “Carregar” que chama a função **carregar_lista()** para recuperar os dados salvos

```
arquivo_menu.add_separator()
```

- Adiciona uma linha divisória entre os itens do menu

```
arquivo_menu.add_command(label="Sair",command=janela.quit)
```

- Adiciona a opção “Sair”, que fecha a aplicação ao ser clicada **janela.quit()**

```
menu_bar.add_cascade(label="Arquivo",menu=arquivo_menu)
```

- Adiciona o submenu “Arquivo” à barra de menus principal menu_bar

```
#Widgets de entrada
tk.Label(janela, text="Item: ",
bg="#f4f4f9",font=("Arial",15)).pack(pady=5)
entrada_item = tk.Entry(janela, font=("Arial",14),bd=2,relief="solid")
entrada_item.pack()

tk.Label(janela, text="Quantidade: ", bg="#f4f4f9",font=("Arial",15),
width=20).pack(pady=5)
entrada_quantidade = tk.Entry(janela,
font=("Arial",14),bd=2,relief="solid")
entrada_quantidade.pack()

tk.Label(janela, text="Preço unitário (R$):",
bg="#f4f4f9",font=("Arial",15)).pack(pady=5)
entrada_preco = tk.Entry(janela,  font=("Arial",14),bd=2,relief="solid")
entrada_preco.pack()
```

Aqui, estamos criando os campos de entrada - widgets de inputs para que o usuário possa adicionar itens à lista de compras.

```
tk.Label(janela, text="Item: ",
bg="#f4f4f9",font=("Arial",15)).pack(pady=5)
entrada_item = tk.Entry(janela, font=("Arial",14),bd=2,relief="solid")
entrada_item.pack()
```

- Cria um rótulo (label) chamado “Item”
- Define a cor de fundo
- Define a fonte
- O **pady=5** adiciona um pequeno espaçamento vertical

E assim, sucessivamente…

```
Botões de ação
btn_adicionar = tk.Button(janela, text="Adicionar",
command=adicionar_item, bg="#4CAF50", fg="white", font=("Arial", 15),
width=20, height=2, relief="solid")
btn_adicionar.pack(pady=5)

btn_editar = tk.Button(janela, text="Preencher para Edição",
command=preencher_campos, bg="#FFC107", fg="black", font=("Arial", 15),
width=20, height=2, relief="solid")
btn_editar.pack(pady=5)

btn_atualizar = tk.Button(janela, text="Atualizar",command=editar_item,
bg="#2196F3", fg="white", font=("Arial", 15), width=20, height=2,
relief="solid")
btn_atualizar.pack(pady=5)

btn_remover = tk.Button(janela, text="Remover",command=remover_item,
bg="#F44336", fg="white", font=("Arial", 15), width=20, height=2,
relief="solid")
btn_remover.pack(pady=5)
```

Criando os botões de ação para adicionar, editar, atualizar, remover itens à lista de compras.

- **tk.Button(janela, …)** - cria um botão dentro da janela principal
- **text = “Adicionar”** - Define o texto exibido no botão
- **command=adicionar_item** - quando o botão for clicado, a função **adicionar_item()** será executada
- **bg=”#4CAF50”** - define a cor de fundo para verde
- **fg=”white”** - define a cor do texto como branco
- **font=(“Arial”,15)** - configurações de fonte
- **width=20, height=2** - ajusta o tamanho do botão
- **relief=”solid”** - dá efeito de borda sólida ao botão
- **btn_adicionar.pack(pady=5)** - adiciona o botão à interface com espaçamento vertical de 5px para não ficar colado em outros elementos

```
#Lista de Compras
lista_box = tk.Listbox(janela, width=50, height=10, font=("Arial", 17),
bd=2, relief="solid")
lista_box.pack()
```

- **tk.Listbox(janela, …)** - cria uma lista dentro da janela principal
- **lista_box.pack()** - posiciona o componente na interface

```
#Carregar dados e atualizar lista ao iniciar
carregar_lista()

#Executar a interface gráfica
janela.mainloop()
```