

Guia de Python

Introdução

Este PDF tem o objetivo de explicar detalhadamente os arquivos do repositório. Para aproveitar ao máximo, leia-o com a atenção necessária e pratique os códigos várias vezes. Somente assim você conseguirá fluência na linguagem.

script.py

```
# criando uma variavel
nome = "Alberto" # variavel nome que recebe o valor
"Alberto"
print(nome) # método print fazendo a exibição da variavel

# Exibindo uma string(texto)
print("Alberto é professor de lógica")
```

Detalhes:

- 1.Cria uma variável chamada nome e armazena nela o valor "Alberto".
- 2.Uso o método print para mostrar o valor da variável nome na tela.
- 3.Mostra uma mensagem fixa na tela: "Alberto é professor de lógica".

Basicamente, ele armazena e exibe informações.

script1.py

```
# Tipos de dados simples:  
  
# string  
fruta = "Manga"  
  
# number  
preco = 4.10  
  
# boolean  
condicao = True # Madura  
  
# exibindo as 3 variaveis  
print(fruta, preco, condicao)
```

Detalhes:

- 1.Cria uma variável fruta e armazena a string "Manga".
- 2.Cria uma variável preco e armazena o número 4.10.
- 3.Cria uma variável condicao e armazena o valor booleano True.
- 4.Uso o método print para mostrar todas as três variáveis na tela, separadas por espaços.

script2.py

```
# tipos de dados complexos

# listas
frutas = ["Manga", "Uva", "Morango",
"Goiaba"]
# dicionario
manga = {
    "nome": "Manga",
    "preco": 5.10,
    "quantidade": 10,
    "disponivel": True
}

# exibindo a variavel frutas
print(frutas)

# exibindo a variavel manga
print(manga)
```

Detalhes:

- 1.Cria uma lista chamada frutas com quatro elementos: "Manga", "Uva", "Morango", e "Goiaba".
- 2.Cria um dicionário chamado manga com quatro pares chave-valor: "nome" (valor "Manga"), "preco" (valor 5.10), "quantidade" (valor 10), e "disponivel" (valor True).
- 3.Uso o método print para mostrar a lista frutas e o dicionário manga na tela.

script3.py

```
# Manipulando listas:  
# definição: listas são tipos de dados capaz de armazenar vários  
dados  
# listas  
frutas = ["Manga", "Uva", "Morango", "Goiaba"]  
  
# acessando o primeiro item da lista  
frutas[0] # teste um print(frutas[0]) para ver o resultado  
# acessando o último  
frutas[-1] # teste um print(frutas[-1]) para ver o resultado  
# adicionando um novo item na lista  
frutas.append("Melancia")  
# removendo um item da lista  
frutas.remove("Goiaba")  
  
# exibindo a variável frutas  
print(frutas)  
  
# alterando o valor de um item da lista  
frutas[1] = "Laranja" # Uva virou Laranja  
  
# exibindo a variável frutas  
print(frutas)
```

Detalhes:

- 1.Cria uma lista frutas com quatro itens: "Manga", "Uva", "Morango", e "Goiaba".
- 2.Acessa e mostra o primeiro item da lista ("Manga") e o último item ("Goiaba").
- 3.Adiciona "Melancia" ao final da lista.
- 4.Remove "Goiaba" da lista.
- 5.Mostra a lista atualizada.
- 6.Substitui o segundo item ("Uva") por "Laranja".
- 7.Mostra a lista com a alteração.

script4.py

```
# dicionário
# definição: é um tipo de dado que armazena pares de chave e
valor
# dicionário
fruta = {
    "chave_nome": "goiAba",
    "preco": 3.12
}

# Outro exemplo de dicionário:
fruta1 = {"nome": "Manga", "preco": 2.99}

# alterando um valor pela chave
fruta["chave_nome"] = "Goiaba" # novo valor 'Goiaba'

# acrescentando uma nova chave
fruta["disponivel"] = False

# Exibindo fruta
print(fruta)
```

Detalhes:

- 1.Cria um dicionário fruta com duas chaves: "chave_nome" com valor "goiAba" e "preco" com valor 3.12.
- 2.Cria outro dicionário fruta1 com "nome" igual a "Manga" e "preco" igual a 2.99.
- 3.Alterar o valor da chave "chave_nome" para "Goiaba".
- 4.Adiciona uma nova chave "disponivel" com valor False.
- 5.Mostra o dicionário fruta atualizado.



"""

Símbolo	Definição
""	string -> texto (representa texto entre aspas)
nome()	chamada de métodos ou funções (invoca funções)
.nome	acesso a um método ou propriedade de um objeto
[]	listas (também usado para acessar elementos por índice)
{}	dicionários (também usados para criar conjuntos)

- Observações:

- Números não possuem "", são representados sem aspas.
- Decimais são separados por . (ponto), não por , (vírgula).
- [] pode criar listas vazias ou acessar elementos específicos.
- {} pode criar dicionários ou conjuntos, dependendo do contexto.

Métodos mais comuns por tipos de dados:

Strings:

- .lower() - Converte para minúsculas
- .upper() - Converte para maiúsculas
- .strip() - Remove espaços extras
- .split() - Divide a string em uma lista
- .replace(old, new) - Substitui partes da string

Numbers:

- abs(x) - Retorna o valor absoluto
- round(x, ndigits) - Arredonda para 'ndigits' casas decimais
- pow(x, y) - Retorna 'x' elevado à potência 'y'
- int(x) - Converte para inteiro
- float(x) - Converte para float

Listas:

- .append(item) - Adiciona um item no final
- .pop(index) - Remove e retorna o item no índice
- .sort() - Ordena a lista
- .reverse() - Inverte a lista
- .extend(lista) - Estende a lista com outra

Dicionários:

- .get(key) - Retorna o valor de uma chave
- .keys() - Retorna as chaves
- .values() - Retorna os valores
- .items() - Retorna pares chave-valor
- .pop(key) - Remove e retorna o valor da chave

"""

script6.py

```
# variável com valor False
fruta_condicao = False

# condição verificando se a variável é
# se fruta_condicao == False:
    print("Está verde a fruta") # Exibindo

# mesma condição de forma mais ideal
if not fruta_condicao:
    print("Está verde também") # Exibindo
```

Detalhes:

- 1.Cria uma variável fruta_condicao com valor False.
- 2.Uso um if para verificar se fruta_condicao é False e, se for, mostra "Está verde a fruta".
- 3.Uso a forma mais simples com if not fruta_condicao para verificar se fruta_condicao é False e mostra "Está verde também".

script6.py

```
# variável com valor False
fruta_condicao = False

# condição verificando se a variável é
# se fruta_condicao == False:
    print("Está verde a fruta") # Exibindo

# mesma condição de forma mais ideal
if not fruta_condicao:
    print("Está verde também") # Exibindo
```

Detalhes:

- 1.Cria uma variável fruta_condicao com valor False.
- 2.Uso um if para verificar se fruta_condicao é False e, se for, mostra "Está verde a fruta".
- 3.Uso a forma mais simples com if not fruta_condicao para verificar se fruta_condicao é False e mostra "Está verde também".

script7.py

```
● ● ●

# variável com valor True
fruta_condicao = True

# condição verificando se a variável é
# se fruta_condicao == True:
    print("Está madura") # Exibindo

# mesma condição, porém ideal
if fruta_condicao:
    print("Está madura também") # Exibindo
```

Detalhes:

- 1.Cria uma variável fruta_condicao com valor True.
- 2.Uso um if para verificar se fruta_condicao é True e, se for, mostra "Está madura".
- 3.Uso a forma mais simples com if fruta_condicao para verificar se fruta_condicao é True e mostra "Está madura também".

script8.py

```
# Uma lista de dicionário
frutas = [
    {"nome": "Maçã", "cor": "Vermelha", "preço": 3.50},
    {"nome": "Banana", "cor": "Amarela", "preço": 2.00},
    {"nome": "Laranja", "cor": "Laranja", "preço": 2.50},
    {"nome": "Uva", "cor": "Roxa", "preço": 4.00},
    {"nome": "Manga", "cor": "Amarela", "preço": 5.00},
]

# Verificando se a fruta do index 1 e chave nome é igual a
if frutas[1]["nome"] == "banana":
    # Não vai exibir nada pois não atende a condição
    print("Encontrei a fruta Banana")
```

Detalhes:

- 1.Cria uma lista chamada frutas com cinco dicionários. Cada dicionário representa uma fruta com chaves "nome", "cor", e "preço".
- 2.Uso um if para verificar se o valor da chave "nome" do dicionário no índice 1 (que é "Banana") é igual a "banana".
- 3.Não exibe nada, pois a condição não é atendida (lembre-se que "Banana" não é igual a "banana" por causa da diferença de maiúsculas e minúsculas).



```
"""
```

Tipos de condições:

Operador		Descrição
==		Igual a
!=		Diferente de
>		Maior que
<		Menor que
>=		Maior ou igual a
<=		Menor ou igual a

```
"""
```

script10.py

```
frutas = ["Morango", "Laranja", "Melancia", "Manga",
"Goiaba"]
# Estrutura de repetição, que percorre a lista frutas
for item in frutas:
    print(item) # Exibe cada item da fruta
```

Detalhes:

1. Cria uma lista chamada frutas com cinco itens: "Morango", "Laranja", "Melancia", "Manga", e "Goiaba".
2. Usa um for para percorrer cada item da lista frutas.
3. Mostra cada item da lista um por um na tela.

script11.py

```
# indeces:  0      1      2      3      4
frutas = ["Morango", "Laranja", "Melancia", "Manga",
"Goiaba"]
# indices: 0  1  2  3  4
precos = [2.19, 3.0, 1.77, 3.5, 2.09]

# O enumerate gera o índice e o item da lista frutas
for index, item in enumerate(frutas):
    # exibe o item de frutas e o preço por index
    print(f"A fruta: {item} custa {precos[index]}")
```

Detalhes:

- 1.Cria duas listas: frutas com nomes de frutas e precos com os preços correspondentes.
- 2.Uso enumerate para obter o índice e o item de frutas.
- 3.Mostra o nome da fruta e o preço correspondente, usando o índice para acessar o preço na lista precos.

script12.py

```
# indices: 0      1      2      3      4
frutas = ["Morango", "Laranja", "Melancia", "Manga", "Goiaba"]

# indices: 0  1  2  3  4
precos = [2.19, 3.0, 1.77, 3.5, 2.09]

# 0 range(0, 2) vai do índice 0 até o índice 1 (não inclui o 2)
for index in range(0, 2, 1): # começa do 0, vai até a posição 2 de 1 em
    print(f"{frutas[index]} custa {precos[index]}")
```

Detalhes:

- 1.Cria duas listas: frutas e precos.
- 2.Uso range(0, 2, 1) para gerar índices de 0 a 1 (não inclui 2).
- 3.Mostra o nome da fruta e o preço correspondente apenas para os índices 0 e 1 das listas.

script13.py

```
condicao = True # Define a condição inicial como True

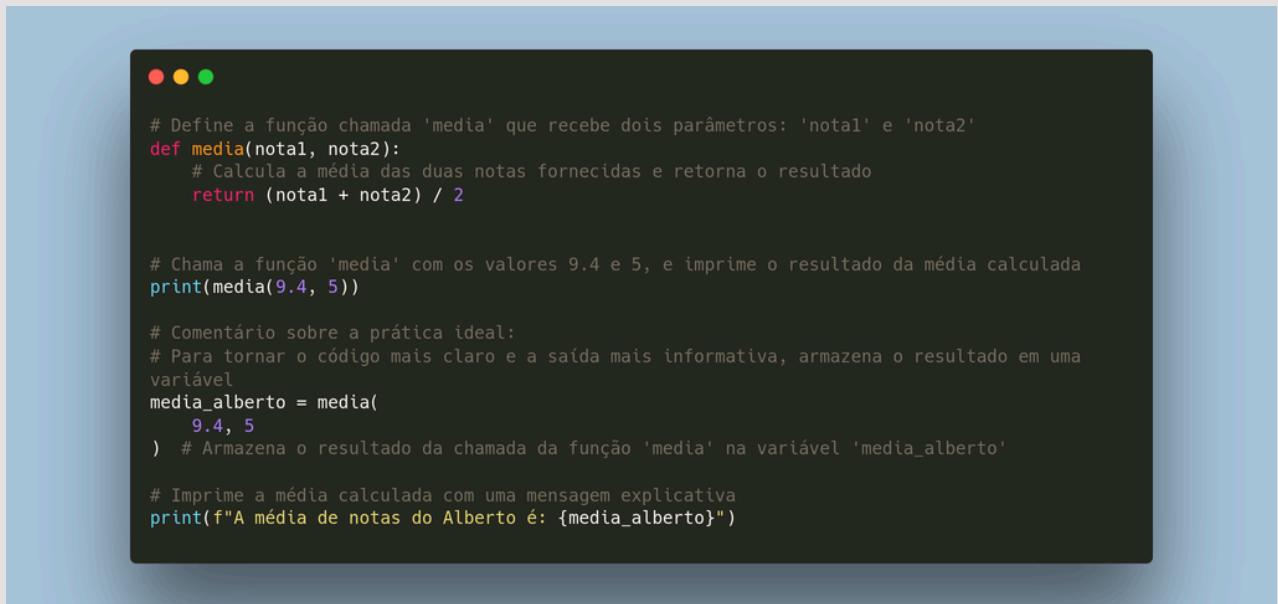
while condicao: # Enquanto a condição for True, o loop continuará
    print("Vou continuar até a condição ficar falsa.") # Exibe mensagem

    entrada = input("Digite X para encerrar: ") # Solicita input do usuário
    if entrada == "X": # Se o usuário digitar "X"
        condicao = False # Altera a condição para False, encerrando o loop
```

Detalhes:

1. Define a variável condicao como True.
2. Inicia um loop while que continua enquanto condicao for True.
3. Mostra a mensagem "Vou continuar até a condição ficar falsa.".
4. Pede para o usuário digitar algo.
5. Verifica se o usuário digitou "X". Se sim, muda condicao para False, encerrando o loop.

script14.py



```
# Define a função chamada 'media' que recebe dois parâmetros: 'nota1' e 'nota2'
def media(nota1, nota2):
    # Calcula a média das duas notas fornecidas e retorna o resultado
    return (nota1 + nota2) / 2

# Chama a função 'media' com os valores 9.4 e 5, e imprime o resultado da média calculada
print(media(9.4, 5))

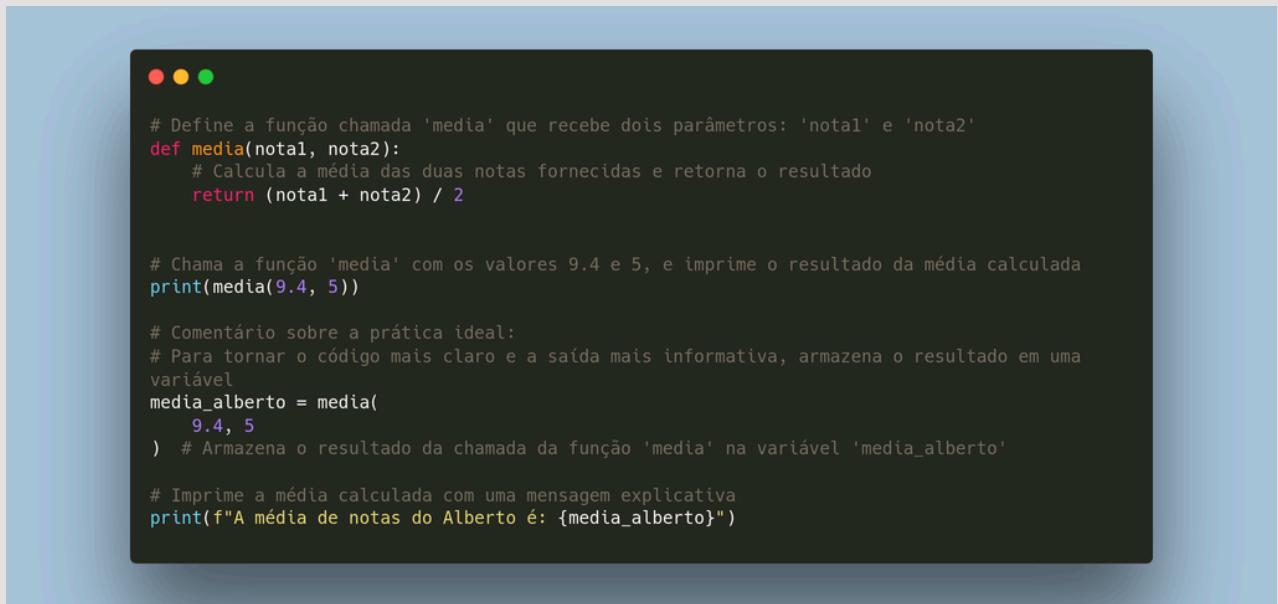
# Comentário sobre a prática ideal:
# Para tornar o código mais claro e a saída mais informativa, armazena o resultado em uma
variável
media_alberto = media(
    9.4, 5
) # Armazena o resultado da chamada da função 'media' na variável 'media_alberto'

# Imprime a média calculada com uma mensagem explicativa
print(f"A média de notas do Alberto é: {media_alberto}")
```

Detalhes:

1. Define uma função chamada `media` que recebe duas notas e calcula a média delas.
2. Chama a função com os valores 9.4 e 5, e mostra o resultado da média.
3. Armazena o resultado da função em uma variável chamada `media_alberto`.
4. Mostra a média com uma mensagem explicativa: "A média de notas do Alberto é: [resultado]".

script15.py



```
# Define a função chamada 'media' que recebe dois parâmetros: 'nota1' e 'nota2'
def media(nota1, nota2):
    # Calcula a média das duas notas fornecidas e retorna o resultado
    return (nota1 + nota2) / 2

# Chama a função 'media' com os valores 9.4 e 5, e imprime o resultado da média calculada
print(media(9.4, 5))

# Comentário sobre a prática ideal:
# Para tornar o código mais claro e a saída mais informativa, armazena o resultado em uma
variável
media_alberto = media(
    9.4, 5
) # Armazena o resultado da chamada da função 'media' na variável 'media_alberto'

# Imprime a média calculada com uma mensagem explicativa
print(f"A média de notas do Alberto é: {media_alberto}")
```

Detalhes:

1. Define uma função chamada `media` que recebe duas notas e calcula a média delas.
2. Chama a função com os valores 9.4 e 5, e mostra o resultado da média.
3. Armazena o resultado da função em uma variável chamada `media_alberto`.
4. Mostra a média com uma mensagem explicativa: "A média de notas do Alberto é: [resultado]".

script15.py

```
# Função para adicionar uma nova fruta e seu preço à lista
def adicionar(fruta, preco):
    frutas_precos.append(
        {"fruta": fruta, "preco": preco}
    ) # Adiciona um dicionário à lista

# Função para listar todas as frutas e seus preços
def listar():
    for item in frutas_precos: # Percorre cada dicionário na lista
        print(item["fruta"], item["preco"]) # Imprime a fruta e o
        preço
# Função principal que controla o menu e as opções do usuário
def main():
    while True: # Loop contínuo até que o usuário decida encerrar
        # Exibe as opções disponíveis para o usuário
        print("[1] - Para adicionar uma nova fruta")
        print("[2] - Para listar todas as frutas")
        print("[0] - Para encerrar \n")
        entrada = input("Sua opção: ") # Recebe a opção do usuário

        if entrada == "0":
            break # Encerra o loop e o programa

        if entrada == "1":
            # Recebe o nome e o preço da nova fruta
            entrada_fruta = input("Nome da fruta: ")
            entrada_preco = input("Preço da fruta:")
            # Adiciona a nova fruta à lista
            adicionar(entrada_fruta, entrada_preco)

        if entrada == "2":
            # Lista todas as frutas e preços
            listar()

    # Chama a função principal para iniciar o programa
main()
```

Detalhes:

- 1.Cria uma lista de dicionários frutas_precos com frutas e seus preços.
- 2.Define a função adicionar para adicionar uma nova fruta e seu preço à lista.
- 3.Define a função listar para mostrar todas as frutas e preços da lista.
- 4.Define a função main que:
- 5.Mostra um menu com opções.
- 6.Permite ao usuário adicionar uma nova fruta, listar todas as frutas ou encerrar o programa.
- 7.Uso de um loop contínuo até que o usuário escolha encerrar (0).
- 8.Chama a função main para iniciar o programa.

script16.py

```
import requests # Importa a biblioteca requests

# URL da API
url =
"https://www.themealdb.com/api/json/v1/1/categories.php"
# Faz a requisição GET para a API
response = requests.get(url)

# Verifica se a requisição foi bem-sucedida
if response.status_code == 200:
    # Converte a resposta em formato JSON
    data = response.json()
    # Imprime os dados recebidos
    print(data)
else:
    # Caso a requisição não tenha sido bem-sucedida
    print(f"Falha na requisição: {response.status_code}")
```

Detalhes:

1. Importa a biblioteca requests para fazer requisições HTTP.
2. Define a URL da API.
3. Faz uma requisição GET para a API usando requests.get(url).
4. Verifica se a requisição foi bem-sucedida (código de status 200).
5. Converte a resposta para JSON e mostra os dados recebidos.
6. Se a requisição falhar, mostra uma mensagem de erro com o código de status.

script17.py

```
import requests

def getApi():
    try:
        # URL da API
        url = "https://www.themealdb.com/api/json/v1/1/categories.php"

        # Faz a requisição GET para a API
        response = requests.get(url)

        # Verifica se a requisição foi bem-sucedida
        if response.status_code == 200:
            # Converte a resposta em formato JSON
            data = response.json()
            return data
        else:
            # Retorna uma mensagem de erro se a resposta não for bem-
            # sucedida
            return f"Falha na requisição: {response.status_code}"

    except requests.RequestException as e:
        # Retorna uma mensagem de erro em caso de exceção
        return f"Erro na requisição: {e}"

    # Exemplo de uso da função
    resultado = getApi()

    # Depois de testar a linha abaixo comente ela, por conta do próximo arquivo
    print(resultado)
```

Detalhes:

1. Importa a biblioteca requests.
2. Define a função getApi que:
3. Faz uma requisição GET para a URL da API.
4. Verifica se a requisição foi bem-sucedida (código de status 200).
5. Se bem-sucedida, converte a resposta para JSON e retorna os dados.
6. Se não bem-sucedida, retorna uma mensagem de erro com o código de status.
7. Se ocorrer uma exceção, retorna uma mensagem de erro com detalhes da exceção.
8. Chama a função getApi e armazena o resultado na variável resultado.
9. Mostra o resultado usando print.

script18.py

```
● ● ●  
from script17 import getApi # Importa a função getApi do módulo script17  
  
# Chama a função getApi e armazena o resultado na variável 'resultado'  
resultado = getApi()  
  
# Acessa o primeiro item da lista associada à chave 'categories' no dicionário  
resultado_1 = resultado['categories'][0]  
  
# Imprime o primeiro item da lista de categorias  
print(resultado_1)
```

Detalhes:

1. Importa a função getApi do módulo script17.
2. Chama a função getApi e armazena o resultado na variável resultado.
3. Acessa o primeiro item da lista associada à chave 'categories' no dicionário resultado.
4. Mostra esse primeiro item da lista usando print.

script19.py

```
● ● ●

# Nome do arquivo
arquivo = './6-arquivos/frutas.txt' # Caminho para o arquivo frutas.txt dentro do diretório
'6-arquivos'

# Abre o arquivo para leitura
with open(arquivo, 'r') as file: # Abre o arquivo no modo leitura ('r')
    # Itera sobre cada linha do arquivo
    for linha in file: # Percorre cada linha do arquivo
        print(linha.strip()) # Remove espaços extras e quebras de linha e exibe o conteúdo da
linha
```

Detalhes:

1. Define o caminho para o arquivo frutas.txt.
2. Abre o arquivo no modo leitura ('r') usando um bloco with, que garante que o arquivo será fechado corretamente após o uso.
3. Percorre cada linha do arquivo.
4. Remove espaços extras e quebras de linha de cada linha com strip() e mostra o conteúdo.

script20.py

```
# Nome do arquivo
arquivo = "./6-arquivos/frutas.txt" # Caminho para o arquivo frutas.txt localizado no
diretório '6-arquivos'

# Lista para armazenar os dados lidos
frutas_precos = [] # Inicializa uma lista vazia para armazenar os dados lidos do arquivo

# Abre o arquivo para leitura
with open(arquivo, "r") as file: # Abre o arquivo no modo leitura ('r')
    for linha in file: # Itera sobre cada linha do arquivo
        print(
            linha.strip().split(",")
        ) # Remove espaços extras e quebras de linha, divide a linha por vírgula e imprime a
        lista resultante

    print() # Adiciona uma linha em branco para separar a saída dos blocos de código

# Abre o arquivo para leitura
with open(arquivo, "r") as file: # Reabre o arquivo para leitura
    for linha in file: # Itera sobre cada linha do arquivo
        fruta, preco = linha.strip().split(
            ","
        ) # Divide a linha em duas partes: fruta e preço, removendo espaços extras
        print(fruta, preco) # Imprime a fruta e o preço separados por um espaço

    print() # Adiciona uma linha em branco para separar a saída dos blocos de código

# Abre o arquivo para leitura
with open(arquivo, "r") as file: # Reabre o arquivo para leitura
    for linha in file: # Itera sobre cada linha do arquivo
        fruta, preco = linha.strip().split(
            ","
        ) # Divide a linha em duas partes: fruta e preço, removendo espaços extras
        frutas_precos.append(
            {"fruta": fruta, "preco": float(preco)}
        ) # Adiciona um dicionário com a fruta e o preço (convertido para float) à lista
        frutas_precos

    print(frutas_precos) # Imprime a lista de dicionários contendo frutas e preços
```

Detalhes:

1. Define o caminho para o arquivo frutas.txt.
2. Cria uma lista vazia frutas_precos para armazenar os dados lidos do arquivo.
3. Abre o arquivo e:
4. Lê e mostra cada linha, dividida por vírgula.
5. Lê e mostra a fruta e o preço separados.
6. Lê e armazena os dados como dicionários na lista frutas_precos, convertendo o preço para float.
7. Mostra a lista frutas_precos contendo dicionários com frutas e preços.

script21.py

```
● ● ●

# Caminho para o arquivo
arquivo = "./6-arquivos/frutas_vazio.txt" # Define o caminho para o arquivo frutas_vazio.txt
dentro do diretório '6-arquivos'

# Lista de frutas e preços
frutas_precos = [
    {"fruta": "Morango", "preco": 2.19},
    {"fruta": "Laranja", "preco": 3.0},
    {"fruta": "Melancia", "preco": 1.77},
    {"fruta": "Manga", "preco": 3.5},
    {"fruta": "Goiaba", "preco": 2.09},
] # Define uma lista de dicionários com frutas e seus preços

# Abre o arquivo para escrita
with open(arquivo, "w") as file: # Abre o arquivo no modo escrita ('w'). Se o arquivo já
existir, ele será sobreescrito
    # Itera sobre cada item na lista frutas_precos
    for item in frutas_precos:
        # Escreve a fruta e o preço no arquivo, separados por vírgula, seguido de uma nova
        linha
        file.write(f"{item['fruta']},{item['preco']}\n")

# ATENÇÃO MANTENHA O ARQUIVO frutas_vazio sem nada.
```

Detalhes:

1. Define o caminho para o arquivo `frutas_vazio.txt`.
2. Cria uma lista de dicionários com frutas e preços.
3. Abre o arquivo no modo escrita ('w'), que sobrescreve o arquivo se ele já existir.
4. Escreve cada fruta e seu preço no arquivo, separados por vírgula e seguidos de uma nova linha.

script22.py

```
import pandas as pd # Importa a biblioteca pandas com o alias pd

# Dados das frutas e preços
dados = {
    "fruta": ["Morango", "Laranja", "Melancia", "Manga",
    "Goiápaéđo": [2.19, 3.0, 1.77, 3.5, 2.09],
} # Define um dicionário com frutas e preços

# Cria um DataFrame a partir dos dados
df = pd.DataFrame(dados) # Converte o dicionário em um DataFrame

# Exibe o DataFrame
print(df) # Mostra o DataFrame no console

# Salva o DataFrame em um arquivo CSV
df.to_csv(
    "./7-pandas/frutas.csv", index=False
) # Salva o DataFrame como 'frutas.csv', sem incluir o índice
```

Detalhes:

1. Importa a biblioteca pandas como pd.
2. Define um dicionário com frutas e preços.
3. Cria um DataFrame a partir do dicionário.
4. Mostra o DataFrame no console.
5. Salva o DataFrame em um arquivo CSV chamado frutas.csv, sem incluir o índice.

script23.py

```
● ● ●

import pandas as pd # Importa a biblioteca pandas com o alias pd

# Lê o DataFrame a partir do arquivo CSV
df = pd.read_csv("./7-pandas/frutas.csv") # Lê o arquivo CSV e cria um DataFrame

# Exibe o DataFrame
print(df) # Mostra o DataFrame no console

# Ordena o DataFrame por preço
df_ordenado = df.sort_values(by="preco") # Ordena o DataFrame pela coluna
# 'preco'
# Exibe o DataFrame ordenado
print("\nLista de Frutas e Preços (Ordenado por Preço):")
print(df_ordenado) # Mostra o DataFrame ordenado no console

# Exibe apenas os nomes das frutas
print("\nNomes das Frutas:")
print(df["fruta"]) # Mostra apenas a coluna 'fruta'
```

Detalhes:

1. Importa a biblioteca pandas como pd.
2. Lê o arquivo CSV frutas.csv e cria um DataFrame.
3. Mostra o DataFrame no console.
4. Ordena o DataFrame pela coluna preco e armazena o resultado em df_ordenado.
5. Mostra o DataFrame ordenado no console.
6. Mostra apenas a coluna fruta do DataFrame.

script24.py

```
● ● ●

import pandas as pd # Importa a biblioteca pandas com o alias pd

# Lê o DataFrame a partir do arquivo CSV
df = pd.read_csv("./7-pandas/frutas.csv") # Lê o arquivo CSV e cria um DataFrame

# Filtra frutas com preço menor que 3.0
frutas_baratas = df[df["preco"] < 3.0] # Filtra as frutas com preço menor que 3.0
# Exibe as frutas mais baratas
print("Frutas com preço menor que 3.0:")
print(frutas_baratas) # Mostra as frutas filtradas no console

# Adiciona uma coluna indicando se a fruta é barata ou cara
df["categoria"] = df["preco"].apply(lambda x: "Barato" if x < 3.0 else "Caro")

# Exibe o DataFrame com a nova coluna
print("\nFrutas com Categorias:")
print(df) # Mostra o DataFrame atualizado no console

# Calcula a média dos preços das frutas
media_precos = df["preco"].mean()

# Exibe a média dos preços
print(f"\nMédia dos Preços das Frutas: {media_precos:.2f}")
```

Detalhes:

1. Importa a biblioteca pandas como pd.
2. Lê o arquivo CSV frutas.csv e cria um DataFrame.
3. Filtra as frutas com preço menor que 3.0 e armazena o resultado em frutas_baratas.
4. Mostra as frutas com preço menor que 3.0.
5. Adiciona uma nova coluna categoria no DataFrame, que indica se a fruta é "Barato" ou "Caro" com base no preço.
6. Mostra o DataFrame atualizado com a nova coluna.
7. Calcula a média dos preços das frutas e armazena em media_precos.
8. Mostra a média dos preços das frutas.