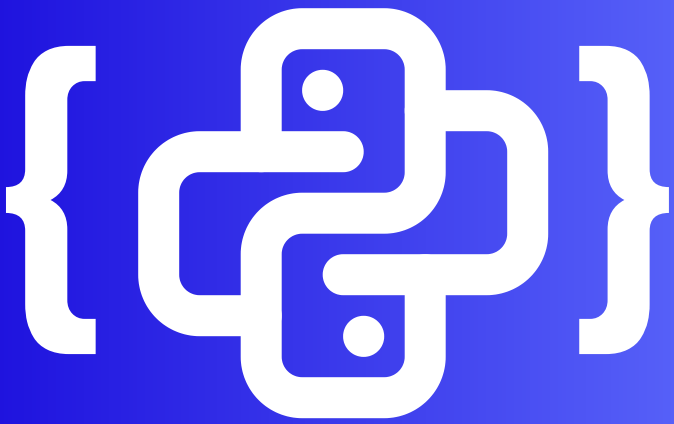
Code em um minuto

PYTHON INTERMEDIÁRIO

Conceitos e Exemplos e mais de 30 Projetos



Conceitos, exemplos e mais de 30 Projetos para praticar Python e rechear seu portifólio!

2023, Python 3.11.2

Introdução

Se você já tem algum conhecimento em Python e está procurando avançar em suas habilidades de programação, este eBook é perfeito para você. "Python Intermediário: Conceitos e Exemplos + 50 Projetos" é um guia completo que aborda conceitos e exemplos intermediários da linguagem Python e ainda oferece 50 projetos práticos para você aprimorar suas habilidades e enriquecer seu portfólio.
Ao longo deste eBook, você irá explorar diversos conceitos importantes, como listas, dicionários, manipulação de arquivos, funções, tratamento de exceções, dentre outros.

Não importa se você é um estudante de programação, um desenvolvedor iniciante ou um profissional experiente, este eBook irá ajudá-lo a aprimorar suas habilidades em Python e se destacar em um mercado cada vez mais competitivo. Então, pegue sua xícara de café e prepare-se para mergulhar no mundo do Python intermediário!

Lembrando que essa é a parte 2 do e-book. Então se você não entender bem, volte e verifique a parte 1!

Listas

As listas são uma das estruturas de dados mais básicas em Python e são usadas para armazenar um conjunto ordenado de valores em uma única variável. As listas são denotadas por colchetes e cada elemento dentro delas é separado por vírgulas.

As listas são úteis quando precisamos armazenar múltiplos valores e acessá-los ou modificá-los de maneira dinâmica durante a execução do programa. Por exemplo, podemos usar uma lista para armazenar os nomes de uma lista de alunos, as notas de cada um em uma disciplina, os números de uma sequência de Fibonacci, entre outros.

Aqui estão alguns exemplos de como usar listas em Python:

1. Armazenar os nomes de alunos em uma lista:

```
alunos = ['Ana', 'Bruno', 'Carla', 'Daniel', 'Eduardo']
```

2. Armazenar as notas de cada aluno em uma lista:

```
notas = [8.5, 7.2, 9.0, 6.5, 8.9]
```

3. Armazenar uma sequência de números em uma lista:

```
numeros = [1, 1, 2, 3, 5, 8, 13, 21]
```

Listas

Para acessar uma lista em Python, podemos usar o índice de cada elemento da lista. O índice é um número que indica a posição do elemento na lista, começando em 0 para o primeiro elemento, 1 para o segundo elemento e assim por diante.

Por exemplo, se temos a lista de alunos criada anteriormente:

```
alunos = ['Ana', 'Bruno', 'Carla', 'Daniel', 'Eduardo']
```

Podemos acessar o primeiro elemento da lista (Ana) usando o índice 0:

```
print(alunos[0])  # saída: Ana
```

Também podemos acessar outros elementos da lista usando seus respectivos índices:

```
print(alunos[1])  # saída: Bruno
print(alunos[2])  # saída: Carla
print(alunos[3])  # saída: Daniel
print(alunos[4])  # saída: Eduardo
```

Listas

O acesso a listas é importante para poder obter e modificar os elementos armazenados em uma lista.

Modificando elementos da lista:

```
# modificando o segundo elemento da lista (índice 1)
nomes[1] = 'Beatriz'

# exibindo a lista modificada
print(nomes)  # saída: ['Ana', 'Beatriz', 'Carla', 'Daniel', 'Eduardo']
```

Métodos Para Listas

Métodos de lista em Python são funções que podem ser aplicadas em objetos do tipo lista para realizar uma variedade de operações, como adicionar, remover ou modificar elementos, classificar a lista, inverter a ordem dos elementos, entre outras.

Esses métodos são muito úteis porque permitem que os desenvolvedores realizem operações comuns em listas de maneira fácil e eficiente, sem precisar escrever código complexo e repetitivo. Eles também podem ajudar a tornar o código mais legível e organizado, já que muitas operações em listas podem ser realizadas com apenas uma linha de código.

Alguns dos métodos de lista mais comuns em Python incluem:

- **append(): adiciona um elemento ao final da lista.**
- **insert(): adiciona um elemento em uma posição específica da lista.**
- **remove(): remove o primeiro elemento da lista que corresponde a um valor específico.**
- **pop(): remove e retorna o último elemento da lista.**
- **sort(): ordena os elementos da lista em ordem crescente.**
- **reverse(): inverte a ordem dos elementos na lista.**

Métodos Para Listas

Vamos ver alguns exemplos de como usar esses métodos:

```
# definindo uma lista
frutas = ['maçã', 'banana', 'laranja', 'abacaxi']
```

```
# adicionando um elemento ao final da lista
frutas.append('uva')
print(frutas)
# saída: ['maçã', 'banana', 'laranja', 'abacaxi', 'uva']

# removendo um elemento da lista
frutas.remove('banana')
print(frutas)
# saída: ['maçã', 'laranja', 'abacaxi', 'uva']

# ordenando a lista em ordem alfabética
frutas.sort()
print(frutas)
# saída: ['abacaxi', 'laranja', 'maçã', 'uva']

# invertendo a ordem dos elementos na lista
frutas.reverse()
print(frutas)
# saída: ['uva', 'maçã', 'laranja', 'abacaxi']
```

Esses são apenas alguns exemplos dos métodos de lista em Python. Há muitos outros métodos disponíveis que podem ser usados para realizar uma variedade de operações em listas, dependendo das necessidades do seu programa.

Percorrendo Lista

Em Python, podemos percorrer uma lista usando a estrutura de loop for. Esse loop é especialmente útil para iterar sobre todos os elementos da lista e realizar alguma operação com cada um deles.

Para percorrer uma lista com for, podemos usar a sintaxe básica:

```
for elemento in lista:
    # faça algo com elemento
```

Onde elemento é uma variável que assume o valor de cada elemento da lista durante cada iteração do loop, e lista é a lista que queremos percorrer.

Por exemplo, suponha que temos a seguinte lista de números:

```
numeros = [1, 2, 3, 4, 5]
```

Podemos percorrer essa lista usando um loop for e imprimir cada número na tela da seguinte maneira:

```
for numero in numeros:
    print(numero)
```

A saída desse código seria:

```
1
2
3
4
5
```

Além disso, podemos combinar o loop for com outros conceitos de Python, como condicionais e operações de listas.

Por exemplo, podemos usar o loop for para somar todos os números de uma lista:

```python
numeros = [1, 2, 3, 4, 5]

soma = 0
for numero in numeros:
    soma += numero

print(soma) # saída: 15
```

Nesse exemplo, a variável soma é inicializada com valor zero, e em cada iteração do loop, o valor do elemento atual da lista é adicionado a soma. No final do loop, o valor de soma é impresso na tela, resultando na soma de todos os números da lista.

Tuplas

Em Python, as tuplas são estruturas de dados semelhantes às listas, porém são imutáveis, ou seja, uma vez criadas, seus elementos não podem ser modificados.
As tuplas são declaradas de forma semelhante às listas, utilizando parênteses em vez de colchetes para delimitar seus elementos.

Por exemplo:

```
minha_tupla = (1, 2, 3, 4, 5)
```

Assim como nas listas, os elementos de uma tupla podem ser de qualquer tipo, e elementos de tipos diferentes podem ser combinados em uma única tupla.
A principal diferença entre tuplas e listas é que as tuplas são imutáveis, ou seja, uma vez criadas, seus elementos não podem ser alterados. Isso significa que não é possível adicionar, remover ou alterar elementos de uma tupla depois que ela é criada.

Outra diferença é que as tuplas são mais eficientes em termos de memória e processamento do que as listas, especialmente para grandes quantidades de dados que não precisam ser modificados.

As tuplas podem ser usadas em situações em que queremos garantir que os dados sejam imutáveis e para passagem de valores para funções sem o risco de eles serem modificados dentro da função.

Funções

Em Python, uma função é um bloco de código que realiza uma tarefa específica. Por exemplo, você pode escrever uma função para somar dois números, verificar se um número é par ou ímpar, etc.

Para definir uma função em Python, usamos a palavra-chave "def", seguida do nome da função e, em seguida, dos parênteses. O código da função é escrito abaixo, e tudo o que é necessário para chamá-la é digitar o nome da função seguido dos parênteses.

Aqui está um exemplo simples de função em Python que imprime "Olá, mundo!" na tela:

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

Para chamar essa função no programa principal, basta digitar seu nome seguido dos parênteses:

```
hello()
```

Isso imprimirá "Olá, mundo!" na tela.

Exemplo completo:

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

hello()
```

Outro exemplo de função em Python seria uma função que retorna o dobro de um número:

```
def dobro(numero):
    resultado = numero * 2
    return resultado
```

Nesse exemplo, a função "dobro" recebe um número como parâmetro, multiplica esse número por 2 e retorna o resultado. Para chamar essa função e atribuir seu resultado a uma variável, fazemos o seguinte:

```
resultado = dobro(5)
print(resultado)  # Saída: 10
```

Isso chama a função "dobro" com o número 5 como parâmetro, atribui o resultado (10) à variável "resultado" e, em seguida, imprime o resultado na tela usando a função "print".

Exemplo completo:

```python
def dobro(numero):
    resultado = numero * 2
    return resultado

resultado = dobro(5)
print(resultado)  # Saída: 10
```

Função em Python que verifica se um número é par ou ímpar:

```python
def par_ou_impar(numero):
    if numero % 2 == 0:
        print("O número é par.")
    else:
        print("O número é ímpar.")

par_ou_impar(4)  # Saída: O número é par.
par_ou_impar(7)  # Saída: O número é ímpar.
```

Tratamento de Exceções

O tratamento de exceções em Python é uma técnica que permite ao programador lidar com erros que possam ocorrer durante a execução de um programa. Em vez de simplesmente deixar o programa travar ou emitir uma mensagem de erro genérica, o tratamento de exceções oferece uma maneira mais elegante de lidar com essas situações.
A estrutura básica do tratamento de exceções é feita usando as palavras-chave try e except.

O bloco try contém o código que pode gerar uma exceção, enquanto o bloco except contém o código que será executado se uma exceção for gerada. Veja o exemplo abaixo:

```
try:
    valor = int(input("Digite um número inteiro: "))
    print("O número digitado foi:", valor)
except ValueError:
    print("O valor digitado não é um número inteiro válido.")
```

Tratamento de Exceções

Neste exemplo, o usuário é solicitado a digitar um número inteiro. O bloco try tenta converter a entrada do usuário em um número inteiro usando a função int(). Se o usuário digitar algo que não pode ser convertido em um número inteiro, uma exceção do tipo ValueError será gerada. O bloco except captura essa exceção e exibe uma mensagem de erro mais específica para o usuário.

É possível especificar diferentes tipos de exceções a serem capturadas e tratadas de maneira diferente. Por exemplo:

```
try:
    # código que pode gerar uma exceção
except TypeError:
    # código que trata a exceção do tipo TypeError
except ValueError:
    # código que trata a exceção do tipo ValueError
except:
    # código que trata qualquer outra exceção que não
    # foi especificada anteriormente
```

Tratamento de Exceções

Em resumo, o tratamento de exceções é uma técnica útil para lidar com situações em que erros podem ocorrer durante a execução de um programa. Com a estrutura try-except, é possível capturar exceções específicas e tratar esses erros de maneira mais apropriada para a situação em questão.

Algumas exceções pré-definidas:

- **ValueError: esta exceção é gerada quando o Python tenta converter um valor para um tipo de dado específico, mas o valor não é compatível com esse tipo de dado.**

- **IndexError: esta exceção é gerada quando o Python tenta acessar um elemento de uma lista, tupla ou outro objeto indexado usando um índice que está fora dos limites do objeto.**

- **TypeError: esta exceção é gerada quando o Python tenta executar uma operação em um tipo de dado que não é compatível com essa operação.**

- **ZeroDivisionError: esta exceção é gerada quando o Python tenta dividir um número por zero.**

Dicionários

Dicionários são uma estrutura de dados em Python que permitem armazenar e acessar valores usando uma chave. Eles são semelhantes às listas, mas ao invés de acessar valores por meio de um índice numérico, os valores são acessados por uma chave única associada a cada valor.

A principal diferença entre dicionários e listas é que dicionários são estruturados em pares chave-valor, enquanto as listas são estruturadas em uma sequência ordenada de valores indexados numericamente. Em outras palavras, os elementos de um dicionário são acessados por uma chave, que pode ser de qualquer tipo imutável, enquanto os elementos de uma lista são acessados por um índice numérico.

Em Python, dicionários são declarados utilizando-se chaves {}. Cada item do dicionário é composto por uma chave e um valor separados por dois pontos :. Vamos ver um exemplo simples:

```python
# Declarando um dicionário
dicionario = {'chave1': 'valor1', 'chave2': 'valor2', 'chave3': 'valor3'}

# Acessando valores do dicionário
print(dicionario['chave1']) # saída: 'valor1'
print(dicionario['chave3']) # saída: 'valor3'
```

Neste exemplo, declaramos um dicionário com três pares chave-valor e acessamos os valores associados às chaves 'chave1' e 'chave3'. É importante ressaltar que, se tentarmos acessar uma chave que não existe, o Python levantará uma exceção KeyError.

```
dicionario = {'a': 1, 'b': 2, 'c': 3}
print(dicionario['d'])
```

Nesse exemplo, tentamos acessar a chave 'd', que não existe no dicionário. Como resultado, ocorre o seguinte erro:

```
KeyError: 'd'
```

Ou seja, o Python nos informa que a chave 'd' não existe no dicionário.

Percorrendo um Dicionários

Para percorrer um dicionário com um loop for em Python, podemos utilizar o método items(), que retorna um iterável com pares chave-valor do dicionário.

O formato geral é o seguinte:

```
for chave, valor in dicionario.items():
    # código para executar dentro do loop
```

Por exemplo, suponha que temos o seguinte dicionário:

```
dicionario = {'nome': 'João', 'idade': 30, 'cidade': 'São Paulo'}
```

Podemos percorrê-lo com um loop for da seguinte forma:

```
for chave, valor in dicionario.items():
    print(f"{chave}: {valor}")
```

Isso imprimirá cada chave e valor do dicionário, separados por dois pontos (:), em uma nova linha. O resultado será:

```
nome: João
idade: 30
cidade: São Paulo
```

Percorrendo um Dicionários

Podemos também acessar apenas as chaves ou apenas os valores do dicionário utilizando os métodos keys() e values(), respectivamente. Nesse caso, o formato geral do loop for seria:

```
# percorrendo apenas as chaves
for chave in dicionario.keys():
    # código para executar dentro do loop

# percorrendo apenas os valores
for valor in dicionario.values():
    # código para executar dentro do loop
```

Manipulação de Arquivos

Manipulação de arquivo em Python se refere à capacidade de criar, ler, escrever e modificar arquivos usando a linguagem de programação Python. Essa funcionalidade é muito importante para muitos programas que precisam interagir com arquivos, como leitores de arquivos CSV, geradores de relatórios e programas que precisam armazenar dados em um arquivo.

Aqui está um exemplo de como abrir um arquivo:

```
arquivo = open("nome_arquivo.extensao","modo")
```

Modos de abertura de um arquivo: 'r', 'a' e 'w'

- **'r' significa "read" (ler) e é usado para abrir um arquivo para leitura. Usando 'r', você só pode ler o conteúdo do arquivo, mas não pode escrever nele.**
- **'a' significa "append" (acrescentar) e é usado para abrir um arquivo para escrita. Se o arquivo já existe, o conteúdo novo será acrescentado ao final do arquivo existente. Se o arquivo não existir, ele será criado.**
- **'w' significa "write" (escrever) e é usado para abrir um arquivo para escrita. Se o arquivo já existir, seu conteúdo será substituído pelo novo conteúdo que você escrever. Se o arquivo não existir, ele será criado.**

Se fizermos:

```
arquivo = open("teste.txt","a")
```

Significa que, se o arquivo "teste.txt" já existir, o Python irá abrir o arquivo em modo de acrescentar e você poderá adicionar novas informações ao final do arquivo. Se o arquivo "teste.txt" não existir, o Python criará um novo arquivo com o nome "teste.txt".

Também é importante chamar o método close() após terminar de utilizar o arquivo. Isso garante que todas as operações de gravação ou leitura em um arquivo foram concluídas e que todos os recursos associados ao arquivo, como memória e conexões com o sistema de arquivos, são liberados.

```
arquivo = open("teste.txt","a")
# Operações com o arquivo aqui
arquivo.close()
```

Escrevendo em um arquivo

Existem várias formas de escrever em um arquivo. Aqui estão algumas das principais formas:

Usando o método write():

```
# Abre o arquivo no modo de escrita
arquivo = open("exemplo.txt", "a")

# Escreve uma linha no arquivo
arquivo.write("Exemplo de texto usando o método write()\n")

# Escreve outra linha no arquivo
arquivo.write("Essa é outra linha\n")

# Fecha o arquivo
arquivo.close()
```

Usando o método writelines():

```
# Abre o arquivo no modo de escrita
arquivo = open("exemplo.txt", "a")

# Define uma lista com várias linhas de texto
linhas = ["Exemplo de texto usando o método writelines()\n",
"Essa é outra linha\n", "Mais uma linha\n"]

# Escreve todas as linhas no arquivo
arquivo.writelines(linhas)

# Fecha o arquivo
arquivo.close()
```

Lendo conteúdo do arquivo

Existem várias formas de ler um arquivo. Aqui estão algumas das principais formas:

Usando o método read():

```
# Abre o arquivo no modo de leitura
arquivo = open("exemplo.txt", "r")

# Lê o conteúdo completo do arquivo
conteudo = arquivo.read() # (Tem como retorno, uma string)

# Imprime o conteúdo na tela
print(conteudo)

# Fecha o arquivo
arquivo.close()
```

Usando o método read():

```
# Abre o arquivo no modo de leitura
arquivo = open("exemplo.txt", "r")

# Lê o conteúdo completo do arquivo
conteudo = arquivo.readlines() # (Tem como retorno, uma lista)

# Imprime a lista na tela
print(conteudo)

# Fecha o arquivo
arquivo.close()
```

With

Ao abrir um arquivo em Python, é possível utilizar a função open() para criar um objeto de arquivo que pode ser usado para ler ou escrever em um arquivo. Porém, há uma diferença importante entre abrir um arquivo com with e sem with.

Sem with, você precisa explicitamente fechar o arquivo depois de terminar de usá-lo. Se você não fechar o arquivo, isso pode resultar em problemas, como recursos do sistema operacional sendo desperdiçados, o que pode levar a erros ou comportamento inesperado.

Por outro lado, usando with, o Python irá garantir que o arquivo seja fechado automaticamente depois que você terminar de usá-lo, mesmo que ocorra uma exceção durante a execução do código. Isso significa que você não precisa se preocupar com o fechamento do arquivo explicitamente.

Vejamos um exemplo que mostra a diferença entre abrir um arquivo com e sem with:
Exemplo sem with:

```
arquivo = open('exemplo.txt', 'w')
arquivo.write('Hello, world!')
arquivo.close()
```

With

Neste exemplo, abrimos o arquivo 'exemplo.txt' em modo de escrita, escrevemos a string "Hello, world!" no arquivo e, em seguida, fechamos o arquivo usando o método close().
O problema com esse código é que se ocorrer uma exceção antes de chegarmos ao método close(), o arquivo não será fechado, o que pode levar a problemas.

Exemplo com with:

```
with open('exemplo.txt', 'w') as arquivo:
    arquivo.write('Hello, world!')
```

Neste exemplo, usamos with para abrir o arquivo em modo de escrita e escrevemos a string "Hello, world!" nele. Quando o bloco with é encerrado, o arquivo é automaticamente fechado pelo Python, mesmo que ocorra uma exceção durante a execução do código.

Em resumo, a principal diferença entre abrir um arquivo com e sem with é que with garante que o arquivo seja fechado corretamente, mesmo que ocorra uma exceção durante a execução do código.

Para melhor experiência, os projetos estarão na página abaixo em um formato de PDF diferente.

01 – Calculadora

```python
# define a função 'calculadora' que recebe a operação e dois números
def calculadora(operacao, num1, num2):
    # verifica se a operação escolhida é uma soma
    if operacao == "+":
        # se for, retorna a soma dos números
        return num1 + num2
    # verifica se a operação escolhida é uma subtração
    elif operacao == "-":
        # se for, retorna a subtração dos números
        return num1 - num2
    # verifica se a operação escolhida é uma multiplicação
    elif operacao == "*":
        # se for, retorna a multiplicação dos números
        return num1 * num2
    # verifica se a operação escolhida é uma divisão
    elif operacao == "/":
        # se for, retorna a divisão dos números
        return num1 / num2
    # se a operação escolhida não for nenhuma das opções acima
    else:
        # retorna uma mensagem de erro
        return "Operação inválida"

# imprime uma mensagem de boas-vindas
print("Calculadora em Python")
print("Insira os números e a operação desejada")

# solicita o primeiro número ao usuário e converte para float
num1 = float(input("Digite o primeiro número: "))
# solicita a operação ao usuário
operacao = input("Digite a operação (+, -, *, /): ")
# solicita o segundo número ao usuário e converte para float
num2 = float(input("Digite o segundo número: "))

# chama a função 'calculadora' passando os parâmetros necessários e guarda o
resultado
resultado = calculadora(operacao, num1, num2)

# imprime a operação e o resultado formatados com duas casas decimais
print(f"{num1} {operacao} {num2} = {round(resultado,2)}")
```

02 – Lista de Tarefas

```python
# define uma lista vazia para armazenar as tarefas
```

```python
tarefas = []

# inicia um loop infinito
while True:
    # imprime o menu com as opções disponíveis
    print("======= LISTA DE TAREFAS =======")
    print("1 - Adicionar tarefa")
    print("2 - Ver lista de tarefas")
    print("3 - Remover tarefa")
    print("0 - Sair")

    # solicita ao usuário que escolha uma opção
    opcao = input("Escolha uma opção: ")

    # verifica se o usuário escolheu a opção de adicionar tarefa
    if opcao == "1":
        # solicita ao usuário que digite a nova tarefa e adiciona à lista
        tarefa = input("Digite a nova tarefa: ")
        tarefas.append(tarefa)
        # informa ao usuário que a tarefa foi adicionada com sucesso
        print("Tarefa adicionada com sucesso!")
    # verifica se o usuário escolheu a opção de ver a lista de tarefas
    elif opcao == "2":
        # verifica se a lista de tarefas está vazia
        if not tarefas:
            # se estiver, informa ao usuário que não há tarefas na lista
            print("Não há tarefas na lista.")
        else:
            # se não estiver, imprime a lista de tarefas numeradas
            print("Lista de tarefas:")
            for i, tarefa in enumerate(tarefas):
                print(f"{i+1}. {tarefa}")
    # verifica se o usuário escolheu a opção de remover tarefa
    elif opcao == "3":
        # verifica se a lista de tarefas está vazia
        if not tarefas:
            # se estiver, informa ao usuário que não há tarefas para remover
            print("Não há tarefas para remover.")
        else:
            # solicita ao usuário que digite o número da tarefa a ser removida
            tarefa = int(input("Digite o número da tarefa que deseja remover: "))
            try:
                # verifica se o número digitado é válido
                if tarefa > 0 and tarefa <= len(tarefas):
                    # se for, remove a tarefa da lista e informa ao usuário que a
tarefa foi removida com sucesso
                    tarefas.pop(tarefa-1)
                    print("Tarefa removida com sucesso!")
                else:
```

```
                    # se não for, informa ao usuário que a opção é inválida
                    print("Opção inválida.")
            # trata a exceção em caso de valor inválido digitado
            except ValueError:
                print("Opção inválida.")
    # verifica se o usuário escolheu a opção de sair
    elif opcao == "0":
        # se sim, informa ao usuário que está saindo e encerra o loop
        print("Saindo...")
        break
    # se o usuário escolheu uma opção inválida
    else:
        # informa ao usuário que a opção é inválida
        print("Opção inválida.")
```

03 – Conversor de Unidades

```
# Define a função "converter" que recebe três argumentos: "valor",
"unidade_origem" e "unidade_destino"
def converter(valor, unidade_origem, unidade_destino):
    # Verifica se as unidades de origem e destino são metros e centímetros
    if unidade_origem == "m" and unidade_destino == "cm":
        resultado = valor * 100  # Realiza a conversão de metros para centímetros
        return f"{valor} metros equivalem a {resultado:.2f} centímetros."  #
Retorna o resultado da conversão em uma string formatada
    # Verifica se as unidades de origem e destino são centímetros e metros
    elif unidade_origem == "cm" and unidade_destino == "m":
        resultado = valor / 100  # Realiza a conversão de centímetros para metros
        return f"{valor} centímetros equivalem a {resultado:.2f} metros."  #
Retorna o resultado da conversão em uma string formatada
    # Verifica se as unidades de origem e destino são polegadas e centímetros
    elif unidade_origem == "pol" and unidade_destino == "cm":
        resultado = valor * 2.54  # Realiza a conversão de polegadas para
centímetros
        return f"{valor} polegadas equivalem a {resultado:.2f} centímetros."  #
Retorna o resultado da conversão em uma string formatada
    # Verifica se as unidades de origem e destino são centímetros e polegadas
    elif unidade_origem == "cm" and unidade_destino == "pol":
        resultado = valor / 2.54  # Realiza a conversão de centímetros para
polegadas
        return f"{valor} centímetros equivalem a {resultado:.2f} polegadas."  #
Retorna o resultado da conversão em uma string formatada
    # Se as unidades de origem e destino não forem reconhecidas, retorna uma
mensagem de erro
    else:
        return "Unidades não reconhecidas. Use 'm' para metros, 'cm' para
centímetros e 'pol' para polegadas."
```

```python
# Código principal
print("Conversor de Unidades")
valor = float(input("Digite o valor a ser convertido: "))
unidade_origem = input("Digite a unidade de origem (m, cm ou pol): ")
unidade_destino = input("Digite a unidade de destino (m, cm ou pol): ")

# Chama a função "converter" e imprime o resultado na tela
print(converter(valor, unidade_origem, unidade_destino))
```

04 – Analisador de Texto

```python
# define a função 'analisador_texto' que recebe um parâmetro 'texto'
def analisador_texto(texto):
    # separa o texto em palavras (palavras são definidas por espaços em branco)
    palavras = texto.split()
    # conta quantas palavras existem no texto
    num_palavras = len(palavras)

    # cria uma lista 'letras' contendo todas as letras do texto (excluindo
espaços, pontuações etc)
    letras = [letra for letra in texto if letra.isalpha()]
    # conta quantas letras existem no texto
    num_letras = len(letras)

    # separa o texto em frases (frases são definidas por pontos '.')
    frases = texto.split('.')
    # conta quantas frases existem no texto
    num_frases = len(frases)

    # calcula a média de palavras por frase (se houver pelo menos uma frase)
    if num_frases > 0:
        media_palavras = num_palavras / num_frases
    else:
        media_palavras = 0

    # cria um dicionário 'resultados' contendo os resultados da análise do texto
    resultados = {
        'palavras': num_palavras,
        'letras': num_letras,
        'frases': num_frases,
        'media_palavras_por_frase': media_palavras
    }

    # retorna o dicionário 'resultados'
    return resultados
```

```
# imprime uma mensagem para o usuário
print("Analisador de Texto")
# pede para o usuário digitar um texto
texto = input("Digite o texto a ser analisado: ")

# chama a função 'analisador_texto' com o texto digitado pelo usuário como
parâmetro
resultados = analisador_texto(texto)

# imprime os resultados da análise do texto
print(f"O texto contém {resultados['palavras']} palavras.")
print(f"O texto contém {resultados['letras']} letras.")
print(f"O texto contém {resultados['frases']} frases.")
print(f"A média de palavras por frase é de
{resultados['media_palavras_por_frase']:.2f}.")
```

05 – Jogo da Forca

```
# importa o módulo 'random' para escolher uma palavra aleatória
import random

# define a função 'jogar_forca'
def jogar_forca():
    # cria uma lista de palavras
    palavras = ["abacaxi", "banana", "laranja", "uva", "melancia", "morango",
"limao"]
    # escolhe uma palavra aleatória da lista
    palavra = random.choice(palavras)
    # cria uma lista vazia para armazenar as letras erradas digitadas pelo usuário
    letras_erradas = []
    # cria uma lista vazia para armazenar as letras corretas digitadas pelo
usuário
    letras_certas = []
    # define o número máximo de tentativas
    tentativas = 6

    # inicia o loop do jogo
    while True:
        # verifica se o número de tentativas chegou a zero
        if tentativas == 0:
            print(f"Você perdeu! A palavra era {palavra}")
            break

        # pede para o usuário digitar uma letra (em minúsculas)
        letra = input("Digite uma letra: ").lower()

        # verifica se o usuário já tentou essa letra
```

```
        if letra in letras_certas or letra in letras_erradas:
            print("Você já tentou essa letra! Tente outra.")
            continue

        # verifica se a letra está na palavra
        if letra in palavra:
            letras_certas.append(letra)
        else:
            letras_erradas.append(letra)
            tentativas -= 1

        # cria a palavra escondida (com '_' no lugar das letras não descobertas)
        palavra_escondida = ""
        for letra_palavra in palavra:
            if letra_palavra in letras_certas:
                palavra_escondida += letra_palavra
            else:
                palavra_escondida += "_"

        # exibe a palavra escondida, o número de tentativas restantes e as letras
erradas já digitadas
        print(f"Palavra: {palavra_escondida}")
        print(f"Tentativas restantes: {tentativas}")
        print(f"Letras erradas: {letras_erradas}")

        # verifica se o usuário já descobriu todas as letras da palavra
        if "_" not in palavra_escondida:
            print("Parabéns! Você venceu!")
            break

# chama a função 'jogar_forca'
jogar_forca()
```

06 – Simulador de dados

```
import random

# Inicia um loop infinito
while True:
    # Pausa o programa e espera o usuário pressionar Enter
    input("Pressione Enter para jogar o dado...")
    # Gera um número aleatório entre 1 e 6 e armazena na variável resultado
    resultado = random.randint(1, 6)
    # Imprime o resultado na tela
    print(f"O dado caiu em {resultado}")
    # Pergunta ao usuário se ele deseja jogar novamente e armazena a resposta na
variável continua
```

```
    continua = input("Deseja jogar novamente? (s/n): ")
    # Se a resposta não for 's' (sim), encerra o loop
    if continua.lower() != 's':
        break
```

07 – Contador de Votos

```
candidatos = ["João", "Maria", "José", "Antônio"]
votos = [0, 0, 0, 0]

# Inicia um loop infinito
while True:
    print("Digite o número do seu candidato (ou 0 para sair):")
    # Exibe a lista de candidatos na tela, numerando-os
    for i in range(len(candidatos)):
        print(f"{i+1} - {candidatos[i]}")
    # Lê a escolha do usuário e armazena na variável escolha
    escolha = int(input("Escolha: "))
    # Se o usuário digitar 0, encerra o loop
    if escolha == 0:
        break
    # Se o usuário digitar um número inválido, exibe uma mensagem de erro e
continua o loop
    elif escolha < 1 or escolha > len(candidatos):
        print("Opção inválida. Tente novamente.")
    # Se o usuário digitar um número válido, registra o voto e exibe uma mensagem
de confirmação
    else:
        votos[escolha-1] += 1
        print(f"Você votou em {candidatos[escolha-1]}.\n")

# Exibe o resultado da votação
print("Resultado da votação:")
for i in range(len(candidatos)):
    print(f"{candidatos[i]}: {votos[i]} votos")
```

08 – Jogo de Poker

```
import random

# Define as cartas e naipes do baralho
cartas = ['A', '2', '3', '4', '5', '6', '7', '8', '9', '10', 'J', 'Q', 'K']
naipes = ['♠', '♥', '♦', '♣']
```

```
# Cria o baralho como uma lista de tuplas, cada tupla contendo uma carta e um
naipe
baralho = [(valor, naipe) for valor in cartas for naipe in naipes]

# Embaralha o baralho aleatoriamente
random.shuffle(baralho)

# Distribui as cartas entre os jogadores, removendo-as do baralho
jogador1 = []
jogador2 = []
for i in range(5):
    jogador1.append(baralho.pop())
    jogador2.append(baralho.pop())

# Exibe as cartas de cada jogador
print(f"Jogador 1: {jogador1}")
print(f"Jogador 2: {jogador2}")
```

09 – Adivinhe o número

```
from random import randint

# inicializa as variáveis
continua = True
wins = 0
loss = 0

# imprime o título do jogo
print("=== ADIVINHE O NÚMERO === ")

# loop principal do jogo
while continua:
    # gera um número aleatório entre 1 e 5
    aleatorio = randint(1,5)
    # define o número máximo de tentativas
    tentativas = 3
    # loop para as tentativas do jogador
    while tentativas > 0:
        # solicita que o jogador digite um número
        escolha = int(input("Digite um número: "))
        # verifica se o número escolhido é igual ao número aleatório
        if escolha == aleatorio:
            # caso acerte, incrementa o placar de vitórias, exibe mensagem e
pergunta se deseja continuar
            wins += 1
            print("Parabéns! Você acertou!")
            x = input("Deseja continuar para o próximo número? S/N: ")
```

```
            continua = True if x.upper() == "S" else False
            break
        else:
            # caso erre, decrementa o número de tentativas e exibe mensagem de
erro
            tentativas -= 1
            if tentativas == 0:
                # caso o número de tentativas chegue a zero, incrementa o placar
de derrotas, pergunta se deseja continuar
                # e define a variável continua com base na resposta
                loss += 1
                x = input("Acabaram as tentativas, deseja continuar? S/N: ")
                continua = True if x.upper() == "S" else False
            else:
                # caso ainda hajam tentativas, exibe mensagem de erro e o número
de tentativas restantes
                print(f"Você errou :(. {tentativas} tentativas restantes")
    # exibe o placar atual
    print(f"PLACAR = WINS: {wins} X LOSSES: {loss}")
```

10 – Temporizador

```
from time import sleep

# Definição da função "temporizador"
def temporizador(tempo):
    while tempo >= 0:
        print(tempo)     # imprime o tempo atual
        sleep(1)         # aguarda 1 segundo
        tempo -= 1       # decrementa o tempo em 1 segundo
    print("O tempo acabou!!!")   # imprime mensagem de aviso quando o tempo acaba

# Solicitação do tempo para o usuário
tempo = int(input("Tempo em segundos: "))

# Chamada da função "temporizador" com o tempo fornecido pelo usuário
temporizador(tempo)
```

11 – Gerador de Senhas

```
# Importa as bibliotecas string e random
import string
import random

# Define as variáveis que serão utilizadas na geração de senhas
```

```
letras_minusculas = string.ascii_lowercase
letras_maiusculas = string.ascii_uppercase
numeros = string.digits
caracteres_especiais = "@#$%^&*()!"

# Concatena todas as variáveis em uma única string
todos_caracteres = letras_minusculas + letras_maiusculas + numeros +
caracteres_especiais

# Define o tamanho padrão da senha como 10 caracteres
tamanho = 10

# Define a variável "continua" como 'S' para iniciar o loop while
continua = 'S'

# Inicia o loop while que irá gerar senhas aleatórias até que o usuário decida
parar
while continua == 'S':

    # Gera a senha aleatória utilizando a função sample da biblioteca random
    senha = "".join(random.sample(todos_caracteres, tamanho))

    # Imprime a senha gerada na tela
    print(f"Senha gerada: {senha}")

    # Pede ao usuário para digitar se deseja gerar outra senha ou não
    continua = str(input("Deseja gerar outra? S/N: ")).upper()
```

12 – Agenda de Contatos

```
# Define a função para adicionar um novo contato na agenda
def adicionar_contato(agenda, nome, telefone, email):
    # Adiciona um novo elemento ao dicionário agenda
    # O nome será a chave e o valor será um novo dicionário com telefone e email
como chaves
    agenda[nome] = {'telefone': telefone, 'email': email}

# Define a função para remover um contato existente da agenda
def remover_contato(agenda, nome):
    # Verifica se o nome está presente na agenda
    if nome in agenda:
        # Remove o contato correspondente à chave nome
        del agenda[nome]

# Define a função para procurar um contato na agenda
def procurar_contato(agenda, nome):
    # Verifica se o nome está presente na agenda
```

```
    if nome in agenda:
        # Retorna o valor correspondente à chave nome (que é um dicionário com
telefone e email)
        return agenda[nome]
    else:
        # Caso o nome não esteja presente na agenda, retorna None
        return None

# Define a função para exibir todos os contatos da agenda
def exibir_contatos(agenda):
    # Percorre todos os elementos do dicionário agenda utilizando o método items()
    for nome, contato in agenda.items():
        # Imprime o nome, telefone e email de cada contato
        print(f'{nome} ({contato["telefone"]}, {contato["email"]})')

# Cria um dicionário vazio que será a agenda
agenda = {}

# Adiciona dois contatos na agenda
adicionar_contato(agenda, 'Lucas', '123456', 'lucas@email.com')
adicionar_contato(agenda, 'João', '654321', 'joao@email.com')

# Exibe todos os contatos da agenda
exibir_contatos(agenda)

# Remove um contato da agenda
remover_contato(agenda, 'Lucas')

# Procura um contato na agenda e armazena o resultado na variável "contato"
contato = procurar_contato(agenda, 'Maria')

# Verifica se o contato foi encontrado e imprime uma mensagem correspondente
if contato:
    print(f'Contato encontrado: {contato}')
else:
    print('Contato não encontrado.')
```

13 – Gerador de Nomes Aleatórios

```
# Importa o módulo random
import random

# Define a função para gerar um nome aleatório
def gerar_nome():
    # Define duas strings com vogais e consoantes
    vogais = 'aeiou'
    consoantes = 'bcdfghjklmnpqrstvwxyz'
```

```
    # Define uma string vazia para armazenar o nome gerado
    nome = ''
    # Gera um tamanho aleatório para o nome entre 4 e 10 letras
    tamanho = random.randint(4, 10)
    # Percorre cada posição do nome
    for i in range(tamanho):
        # Se o índice for par, adiciona uma consoante
        if i % 2 == 0:
            nome += random.choice(consoantes)
        # Se o índice for ímpar, adiciona uma vogal
        else:
            nome += random.choice(vogais)
    # Retorna o nome gerado com a primeira letra maiúscula
    return nome.capitalize()

# Define uma variável "continua" com o valor "S"
continua = 'S'

# Enquanto o usuário desejar gerar novos nomes
while continua == 'S':
    # Chama a função gerar_nome() e imprime o resultado na tela
    print(gerar_nome())
    # Pergunta se o usuário deseja gerar outro nome e armazena a resposta em
"continua"
    continua = input('Gerar outro nome? S/N').upper()
```

14 – Decodificador de Código Morse

```
# Dicionário que associa cada letra ou número em código Morse
# com sua representação em pontos e traços.
MORSE_CODE = {
    'A': '.-', 'B': '-...', 'C': '-.-.', 'D': '-..', 'E': '.', 'F': '..-.',
    'G': '--.', 'H': '....', 'I': '..', 'J': '.---', 'K': '-.-', 'L': '.-..',
    'M': '--', 'N': '-.', 'O': '---', 'P': '.--.', 'Q': '--.-', 'R': '.-.',
    'S': '...', 'T': '-', 'U': '..-', 'V': '...-', 'W': '.--', 'X': '-..-',
    'Y': '-.--', 'Z': '--..',
    '0': '-----', '1': '.----', '2': '..---', '3': '...--', '4': '....-',
    '5': '.....', '6': '-....', '7': '--...', '8': '---..', '9': '----.'
}

# Função que recebe uma sequência de pontos e traços em código Morse
# e retorna a mensagem decodificada em caracteres normais.
def decodificar_morse(texto):
    # Divide a sequência de entrada em uma lista de símbolos em código Morse
    codigo_morse = texto.split(' ')
    # Cria uma variável para armazenar a mensagem decodificada
    decodificado = ''
```

```
    # Loop sobre cada símbolo na lista de código Morse
    for simbolo in codigo_morse:
        # Se o símbolo existe como um valor no dicionário MORSE_CODE
        if simbolo in MORSE_CODE.values():
            # Adiciona a letra ou número correspondente ao símbolo decodificado
            decodificado +=
list(MORSE_CODE.keys())[list(MORSE_CODE.values()).index(simbolo)]
        else:
            # Se o símbolo não existir no dicionário, assume que é um espaço em
branco
            decodificado += ' '
    # Retorna a mensagem decodificada
    return decodificado

# Teste da função com uma sequência de entrada em código Morse
print(decodificar_morse('.... . .-.. .-.. --- / .-- --- .-. .-.. -..'))
```

15 – Gerador de Números para Loteria

```
import random

# Função que gera uma lista aleatória de números da loteria, sem repetição,
# com base no número de elementos desejado.
def gerar_numeros_loteria(num):
    return random.sample(range(1, 61), num)

# Função que gera e imprime uma série de jogos da loteria, cada um com
# um número específico de números gerados aleatoriamente.
def jogar_loteria(num_de_jogos, num_de_numeros):
    # Loop sobre o número de jogos desejados
    for i in range(num_de_jogos):
        # Gera uma lista de números da loteria para este jogo
        numeros = gerar_numeros_loteria(num_de_numeros)
        # Imprime a lista de números para este jogo
        print(f"Jogo {i+1}: {numeros}")

# Teste da função jogar_loteria, gerando 5 jogos com 6 números cada
jogar_loteria(5, 6)
```

16 - Calculadora de consumo de energia elétrica

```
# Função que calcula o consumo diário de um aparelho com base na sua potência e
tempo de uso diário.
def calcular_consumo_aparelho(potencia, horas_uso_diario):
    consumo_diario = (potencia * horas_uso_diario) / 1000  # Consumo em kWh
```

```
    return consumo_diario

# Função que calcula o consumo mensal com base no consumo diário e número de dias
no mês.
def calcular_consumo_mensal(consumo_diario, dias_mes):
    consumo_mensal = consumo_diario * dias_mes  # Consumo em kWh
    return consumo_mensal

# Define as variáveis necessárias para calcular o consumo
potencia_aparelho = 100 # Watts
horas_uso_diario = 4
dias_mes = 30

# Calcula o consumo diário e mensal com as funções acima
consumo_diario = calcular_consumo_aparelho(potencia_aparelho, horas_uso_diario)
consumo_mensal = calcular_consumo_mensal(consumo_diario, dias_mes)

# Imprime o resultado para o usuário
print(f"O consumo diário do aparelho é de {consumo_diario:.2f} kWh.")
print(f"O consumo mensal do aparelho é de {consumo_mensal:.2f} kWh.")
```

17 – Simulador de Roleta

```
# Importando o módulo random para geração de números aleatórios
import random

# Definindo uma lista com as possíveis cores da roleta
cores = ['vermelho', 'preto', 'verde']

# Definindo as probabilidades de cada cor ser sorteada
probabilidades = [18/38, 18/38, 2/38]

# Solicitando ao usuário que digite a cor na qual deseja apostar, convertendo-a
para letras minúsculas
aposta = input("Digite a cor que deseja apostar (vermelho, preto ou verde):
").lower()

# Verificando se a cor apostada está na lista de cores válidas
if aposta not in cores:
    print("Aposta inválida!")
else:
    # Sorteando a cor com base nas probabilidades definidas e selecionando o
primeiro elemento da lista retornada
    cor_sorteada = random.choices(cores, probabilidades)[0]

    # Verificando se a cor sorteada é a mesma da aposta e exibindo uma mensagem
correspondente
```

```
    if cor_sorteada == aposta:
        print(f"A roleta parou na cor {cor_sorteada}. Você ganhou!")
    else:
        print(f"A roleta parou na cor {cor_sorteada}. Você perdeu!")
```

18 – Quiz

```
# Exibe a mensagem de boas-vindas ao quiz
print("Bem-vindo ao Quiz!")

# Lista com as perguntas do quiz
perguntas = [
    "Qual é a capital do Brasil? ",
    "Qual é o maior planeta do sistema solar? ",
    "Qual é o maior país do mundo em área territorial? "
]

# Lista com as respostas corretas para cada pergunta
respostas = [
    "brasília",
    "júpiter",
    "rússia"
]

# Variável para armazenar a pontuação do usuário
pontuacao = 0

# Loop que percorre todas as perguntas e realiza a validação da resposta do
usuário
for i in range(len(perguntas)):
    resposta_usuario = input(perguntas[i]).lower() # Recebe a resposta do usuário
e converte para letras minúsculas
    if resposta_usuario == respostas[i]: # Verifica se a resposta do usuário é
igual à resposta correta
        print("Resposta correta!")
        pontuacao += 1 # Adiciona 1 ponto na pontuação do usuário se a resposta
estiver correta
    else:
        print("Resposta incorreta!")

# Exibe a mensagem de fim do quiz e a pontuação final do usuário
print("Fim do quiz!")
print(f"Sua pontuação final é: {pontuacao}")
```

```python
# Dicionário que armazena o estoque dos produtos
estoque = {"Notebook": 5, "Mouse": 10, "Teclado": 11}

# Função que adiciona um produto ao estoque
def adicionar_produto(produto, quantidade):
    if produto in estoque: # Verifica se o produto já está no estoque
        estoque[produto] += quantidade # Se estiver, adiciona a quantidade informada ao estoque atual
    else:
        estoque[produto] = quantidade # Se não estiver, adiciona o produto e a quantidade informada ao estoque

# Função que remove um produto do estoque
def remover_produto(produto):
    if produto in estoque: # Verifica se o produto está no estoque
        del estoque[produto] # Se estiver, remove o produto do estoque
    else:
        print("Produto não encontrado.") # Se não estiver, exibe uma mensagem de erro

# Função que exibe todos os produtos do estoque
def exibir_produtos():
    for produto, quantidade in estoque.items(): # Percorre todos os itens do dicionário
        print(f"{produto} : {quantidade}") # Exibe o nome do produto e a quantidade em estoque

# Loop principal que permite que o usuário interaja com o estoque
while True:
    print("Monitor de Estoque!")
    print("O que deseja fazer?")
    print("1. Exibir Estoque")
    print("2. Adicionar Produto")
    print("3. Remover Produto")
    print("0. Sair")
    opcao = int(input("Escolha: "))

    if opcao == 1:
        print("\nExibindo Estoque...")
        exibir_produtos() # Chama a função que exibe os produtos do estoque
    elif opcao == 2:
        print("\nAdicionando Produto...")
        nome_produto = str(input("Nome do produto: "))
        qnt_estoque = int(input("Quantidade em Estoque: "))
        adicionar_produto(nome_produto, qnt_estoque) # Chama a função que adiciona um produto ao estoque
```

```
    elif opcao == 3:
        print("\nRemovendo Produto...")
        nome_produto = str(input("Nome do produto: "))
        remover_produto(nome_produto) # Chama a função que remove um produto do
estoque
    elif opcao == 0:
        break # Sai do loop principal se o usuário escolher a opção 0
    else:
        print("\nOpção inválida!") # Exibe uma mensagem de erro se o usuário
escolher uma opção inválida
```

20 – Lista de Compras com arquivo

```
# Função para adicionar um item na lista de compras
def adicionar_item(item):
    # Abre o arquivo no modo "append" e adiciona o item
    with open("lista_de_compras.txt", "a") as arquivo:
        arquivo.write(item + "\n")
    print("Item adicionado com sucesso!")

# Função para remover um item da lista de compras
def remover_item(item):
    # Abre o arquivo no modo "read" e lê todas as linhas
    with open("lista_de_compras.txt", "r") as arquivo:
        linhas = arquivo.readlines()
    # Abre o arquivo no modo "write" e escreve todas as linhas que não são o item
a ser removido
    with open("lista_de_compras.txt", "w") as arquivo:
        for linha in linhas:
            if linha.strip() != item:
                arquivo.write(linha)
    print("Item removido com sucesso!")

# Função para exibir a lista de compras
def exibir_lista():
    # Abre o arquivo no modo "read" e lê todas as linhas
    with open("lista_de_compras.txt", "r") as arquivo:
        linhas = arquivo.readlines()
    # Verifica se a lista está vazia
    if not linhas:
        print("Lista vazia.")
    else:
        # Exibe os itens da lista
        print("Lista de compras:")
        for linha in linhas:
            print("- " + linha.strip())
```

```
# Loop principal
while True:
    # Exibe as opções disponíveis
    opcao = input("O que você gostaria de fazer?\n1. Adicionar um item à lista\n2.
Remover um item da lista\n3. Exibir a lista de compras\n4. Sair\n")

    # Executa a opção escolhida pelo usuário
    if opcao == "1":
        item = input("Digite o nome do item que deseja adicionar: ")
        adicionar_item(item)
    elif opcao == "2":
        item = input("Digite o nome do item que deseja remover: ")
        remover_item(item)
    elif opcao == "3":
        exibir_lista()
    elif opcao == "4":
        print("Saindo do programa...")
        break
    else:
        print("Opção inválida. Tente novamente.")
```

21 – Sistema de Login e cadastro com arquivo txt

```
# Função para cadastrar usuário no arquivo 'usuarios.txt'
def cadastrar_usuario(usuario, senha):
    # Abre o arquivo 'usuarios.txt' em modo de escrita e associa à variável
'arquivo'
    with open('usuarios.txt', 'a') as arquivo:
        # Escreve o nome de usuário e senha no arquivo, seguido de uma quebra de
linha
        arquivo.write(usuario + ' ' + senha + '\n')
    # Imprime mensagem informando que o usuário foi cadastrado com sucesso
    print('Usuário cadastrado com sucesso!')

# Função para realizar login de usuário a partir do arquivo 'usuarios.txt'
def realizar_login(usuario, senha):
    # Abre o arquivo 'usuarios.txt' em modo de leitura e associa à variável
'arquivo'
    with open('usuarios.txt', 'r') as arquivo:
        # Lê todas as linhas do arquivo e associa à lista 'usuarios'
        usuarios = arquivo.readlines()

    # Verifica se o nome de usuário e senha informados estão presentes na lista de
usuários
    if usuario + ' ' + senha + '\n' in usuarios:
        # Se estiverem, imprime mensagem informando que o login foi realizado com
sucesso
```

```
        print('Login realizado com sucesso!')
    else:
        # Caso contrário, imprime mensagem informando que usuário ou senha estão
inválidos
        print('Usuário ou senha inválidos!')

# Exibe as opções de escolha para o usuário
print("O que deseja fazer?")
print("1. Cadastro")
print("2. Login")

# Recebe a escolha do usuário e converte para o tipo inteiro
opcao = int(input("Escolha: "))

# Se a opção escolhida for 1, solicita o nome de usuário e senha para cadastrar
if opcao == 1:
    usuario = input('Digite o nome de usuário: ')
    senha = input('Digite a senha: ')
    cadastrar_usuario(usuario, senha)
# Se a opção escolhida for 2, solicita o nome de usuário e senha para realizar
login
elif opcao == 2:
    usuario = input('Digite o nome de usuário: ')
    senha = input('Digite a senha: ')
    realizar_login(usuario, senha)
# Caso contrário, exibe mensagem de opção inválida
else:
    print('Opção inválida!')
```

22 – Criptografador de Mensagens

```
# Define o alfabeto e a chave de substituição
alfabeto = "abcdefghijklmnopqrstuvwxyz"
chave = "qwertyuiopasdfghjklzxcvbnm"

# Função para criptografar uma mensagem
def criptografar(mensagem):
    # Cria uma string vazia para armazenar a mensagem criptografada
    mensagem_criptografada = ""
    # Percorre cada letra da mensagem
    for letra in mensagem:
        # Se a letra estiver no alfabeto, realiza a substituição pela letra
correspondente na chave
        if letra in alfabeto:
            indice = alfabeto.index(letra)
            nova_letra = chave[indice]
            mensagem_criptografada += nova_letra
```

```
        # Se a letra não estiver no alfabeto, adiciona à mensagem criptografada
sem fazer substituição
        else:
            mensagem_criptografada += letra
    # Retorna a mensagem criptografada
    return mensagem_criptografada

# Solicita a mensagem a ser criptografada e a converte para minúsculas
mensagem = input("Digite a mensagem a ser criptografada: ").lower()
# Chama a função criptografar e armazena o resultado na variável
mensagem_criptografada
mensagem_criptografada = criptografar(mensagem)
# Imprime a mensagem criptografada
print("Mensagem criptografada:", mensagem_criptografada)
```

23 – Jogo da Velha

```
def imprimir_tabuleiro(tabuleiro):
    print("  0 1 2")
    for i in range(3):
        print(f"{i} {' '.join(tabuleiro[i])}")

def jogada_valida(tabuleiro, linha, coluna):
    return linha in range(3) and coluna in range(3) and tabuleiro[linha][coluna]
== " "

def jogo_acabou(tabuleiro):
    # Verifica linhas
    for i in range(3):
        if tabuleiro[i][0] == tabuleiro[i][1] == tabuleiro[i][2] and
tabuleiro[i][0] != " ":
            return True

    # Verifica colunas
    for i in range(3):
        if tabuleiro[0][i] == tabuleiro[1][i] == tabuleiro[2][i] and
tabuleiro[0][i] != " ":
            return True

    # Verifica diagonais
    if tabuleiro[0][0] == tabuleiro[1][1] == tabuleiro[2][2] and tabuleiro[0][0]
!= " ":
        return True
    if tabuleiro[0][2] == tabuleiro[1][1] == tabuleiro[2][0] and tabuleiro[0][2]
!= " ":
        return True
```

```
    # Verifica se há jogadas possíveis
    for i in range(3):
        for j in range(3):
            if tabuleiro[i][j] == " ":
                return False

    # Caso não haja jogadas possíveis, o jogo acabou empatado
    return True

def jogar(tabuleiro, jogador):
    imprimir_tabuleiro(tabuleiro)

    print(f"Jogador {jogador}")
    linha = int(input("Linha: "))
    coluna = int(input("Coluna: "))

    if jogada_valida(tabuleiro, linha, coluna):
        tabuleiro[linha][coluna] = jogador
    else:
        print("Jogada inválida, tente novamente")
        jogar(tabuleiro, jogador)

    if jogo_acabou(tabuleiro):
        imprimir_tabuleiro(tabuleiro)
        print("Fim de jogo")
    else:
        proximo_jogador = "O" if jogador == "X" else "X"
        jogar(tabuleiro, proximo_jogador)

tabuleiro = [[" ", " ", " "] for i in range(3)]

jogar(tabuleiro, "X")
```

24 – Conversão de moedas

```
import requests  # Importa a biblioteca requests para fazer requisições HTTP
import json  # Importa a biblioteca json para manipular arquivos JSON

def converter(valor, de, para):
    # Faz uma requisição GET para a API de conversão de moedas com os códigos de
moeda desejados
    api = requests.get(f"https://economia.awesomeapi.com.br/{de}-{para}/")

    # Verifica se a requisição foi bem sucedida (código 200)
    if api.status_code == 200:
        response = api.json()  # Converte a resposta em JSON
        return float(response[0]['bid']) * valor  # Retorna o valor convertido
    else:
```

```
        return None  # Retorna None se a requisição falhou

print("=== Conversor de moedas ===")
print("Principais moedas disponíveis: USD, BRL, EUR, JPY, BTC, ETH, DOGE, ETC.\n")

valor = float(input("Valor: "))  # Recebe o valor a ser convertido do usuário
de = input("Converter de (código): ")  # Recebe o código da moeda de origem do
usuário
para = input("Converter para (código): ")  # Recebe o código da moeda de destino
do usuário

cotacao = converter(valor, de, para)  # Chama a função de conversão

if cotacao is not None:
    print(f"{valor} {de} é equivalente a {cotacao} {para}")  # Imprime o resultado
da conversão
else:
    print("Erro: moeda inválida.")  # Imprime uma mensagem de erro se a conversão
falhou
```

25 – Buscador de CEP

```
import requests  # Importa a biblioteca requests para fazer requisições HTTP

def buscar_cep(cep):
    url = f'https://viacep.com.br/ws/{cep}/json/'  # Monta a URL da API do ViaCEP
com o CEP fornecido
    response = requests.get(url)  # Faz uma requisição GET para a API

    if response.status_code == 200:  # Verifica se a requisição foi bem sucedida
(código 200)
        endereco = response.json()  # Converte a resposta em JSON
        if 'erro' not in endereco:  # Verifica se o CEP foi encontrado
            return endereco  # Retorna o dicionário com as informações do endereço
    return None  # Retorna None se o CEP não foi encontrado

cep = input('Digite o CEP: ')  # Recebe o CEP do usuário
resultado = buscar_cep(cep)  # Chama a função de busca de CEP

if resultado:  # Se o CEP foi encontrado
    print(f'CEP: {resultado["cep"]}')  # Imprime o CEP
    print(f'Logradouro: {resultado["logradouro"]}')  # Imprime o logradouro
    print(f'Complemento: {resultado.get("complemento", "")}')  # Imprime o
complemento (se existir)
    print(f'Bairro: {resultado["bairro"]}')  # Imprime o bairro
    print(f'Cidade: {resultado["localidade"]}')  # Imprime a cidade
    print(f'Estado: {resultado["uf"]}')  # Imprime o estado
```

```
else:
    print('CEP não encontrado.')  # Imprime uma mensagem de erro se o CEP não foi encontrado
```

26 – Gerador de QR Code

```
# Importa a biblioteca qrcode
import qrcode

# Cria um QR Code com o conteúdo "https://www.tiktok.com"
imagem = qrcode.make("https://www.tiktok.com")

# Salva a imagem do QR Code em um arquivo JPG com o nome "meuqrcode.jpg"
imagem.save("meuqrcode.jpg")
```

27 – Calcular Tempo de Viagem

```
# Define a função "calcular_tempo_viagem" que recebe a distância e a velocidade média como parâmetros
def calcular_tempo_viagem(distancia, velocidade_media):
    tempo = distancia / velocidade_media  # Calcula o tempo de viagem em horas
    return tempo  # Retorna o tempo de viagem

distancia = float(input("Digite a distância da viagem (em km): "))  # Lê a distância da viagem do usuário
velocidade_media = float(input("Digite a velocidade média (em km/h): "))  # Lê a velocidade média do usuário
tempo_viagem = calcular_tempo_viagem(distancia, velocidade_media)  # Chama a função "calcular_tempo_viagem"

# Imprime o tempo de viagem formatado com duas casas decimais
print(f"O tempo de viagem é de {tempo_viagem:.2f} horas.")
```

28 – Gerador de Gráficos de Pizza

```
import matplotlib.pyplot as plt

dados = [20, 30, 40, 10]
labels = ['Grupo A', 'Grupo B', 'Grupo C', 'Grupo D']
plt.pie(dados, labels=labels)
plt.show()
```

```python
import os  # Importa o módulo os para trabalhar com arquivos e diretórios
import hashlib  # Importa o módulo hashlib para calcular o hash dos arquivos

def verificar_duplicatas(diretorio):
    # Cria um dicionário vazio para armazenar os hashes dos arquivos
    hash_dict = {}

    # Percorre o diretório e calcula o hash de cada arquivo
    for raiz, diretorios, arquivos in os.walk(diretorio):
        for arquivo in arquivos:
            # Calcula o hash do arquivo
            caminho_arquivo = os.path.join(raiz, arquivo)  # Junta o caminho da raiz com o nome do arquivo
            with open(caminho_arquivo, 'rb') as f:  # Abre o arquivo em modo de leitura binária
                arquivo_bytes = f.read()  # Lê o conteúdo do arquivo em bytes
                arquivo_hash = hashlib.md5(arquivo_bytes).hexdigest()  # Calcula o hash MD5 do conteúdo do arquivo

            # Verifica se o hash já existe no dicionário
            if arquivo_hash in hash_dict:
                # Se já existir, adiciona o caminho do arquivo à lista de arquivos duplicados
                hash_dict[arquivo_hash].append(caminho_arquivo)
            else:
                # Se não existir, adiciona o hash e o caminho do arquivo como uma nova entrada no dicionário
                hash_dict[arquivo_hash] = [caminho_arquivo]

    # Imprime os arquivos duplicados
    duplicatas = [arquivos for arquivos in hash_dict.values() if len(arquivos) > 1]  # Filtra apenas as entradas do dicionário que têm mais de um caminho de arquivo
    if duplicatas:
        print('Os seguintes arquivos são duplicados:')
        for arquivos in duplicatas:
            for arquivo in arquivos:
                print(f'- {arquivo}')  # Imprime o caminho de cada arquivo duplicado
            print()  # Imprime uma linha em branco entre os grupos de arquivos duplicados
    else:
        print('Não foram encontrados arquivos duplicados.')  # Imprime uma mensagem informando que não foram encontrados arquivos duplicados
```

```
# Chama a função verificar_duplicatas, passando o caminho do diretório a ser verificado como argumento
verificar_duplicatas('C:meu/caminho/para/diretorio')
```

30 – Sistema de Backup de Arquivos

```
import shutil  # Importa o módulo shutil para copiar arquivos
import os  # Importa o módulo os para trabalhar com arquivos e diretórios
import time  # Importa o módulo time para obter a data e hora atual

diretorio_origem = 'diretorioparafazer/backup'  # Define o diretório de origem dos arquivos a serem copiados
diretorio_destino = 'caminhoparaarmazenar/backup'  # Define o diretório de destino para o backup

data_hora = time.strftime('%Y-%m-%d-%H-%M-%S')  # Obtém a data e hora atual no formato 'ano-mês-dia-hora-minuto-segundo'
nome_diretorio_backup = f'backup-{data_hora}'  # Define o nome do diretório de backup a partir da data e hora atual

caminho_backup = os.path.join(diretorio_destino, nome_diretorio_backup)  # Junta o caminho do diretório de destino com o nome do diretório de backup
os.makedirs(caminho_backup)  # Cria o diretório de backup no diretório de destino

for nome_arquivo in os.listdir(diretorio_origem):  # Percorre a lista de arquivos no diretório de origem
    caminho_arquivo_origem = os.path.join(diretorio_origem, nome_arquivo)  # Junta o caminho do diretório de origem com o nome do arquivo
    caminho_arquivo_destino = os.path.join(caminho_backup, nome_arquivo)  # Junta o caminho do diretório de backup com o nome do arquivo
    shutil.copy(caminho_arquivo_origem, caminho_arquivo_destino)  # Copia o arquivo da origem para o destino

print(f'Backup completo: {caminho_backup}')  # Imprime uma mensagem informando o caminho do diretório de backup criado.
```

31 – Alarme Básico

```
import datetime  # Importa o módulo datetime para trabalhar com datas e horas
import time  # Importa o módulo time para gerenciar o tempo

# Solicita que o usuário insira a hora para o alarme no formato HH:MM
horario_alarme = input("Insira a hora para o alarme no formato HH:MM: ")

while True:  # Inicia um loop infinito
```

```
    agora = datetime.datetime.now().strftime("%H:%M")  # Obtem a hora atual no
formato HH:MM

    # Verifica se a hora atual é igual ao horário do alarme
    if agora == horario_alarme:
        print("Acionando alarme!")  # Exibe a mensagem de que o alarme foi
acionado
        break  # Encerra o loop infinito e finaliza o programa

    time.sleep(1)  # Espera 1 segundo antes de verificar novamente se é hora de
acionar o alarme
```