Strings e Arquivos

Sumário

Strings

Como já visto, definimos uma string utilizando o caractere ‘ (aspas simples) ou “ (aspas duplas):

```
nome = "Lucas"
```

Strings

Elementos de uma string podem ser acessados de forma similar à listas:

```
nome = "Lucas"

letra = nome[2]
# A letra c será impressa
print(letra)
```

String

Importante! Os valores de uma string não podem ser modificados

```
nome = "Lucas"
nome[2] = "a"
```

```
Traceback (most recent call last):
  File "<string>", line 2, in <module>
TypeError: 'str' object does not support item assignment
```

Tamanho de uma string

Algumas funções utilizadas com listas também funcionam com strings. Um exemplo importante é a função **len()**, que retorna o número de caracteres da string.

No código abaixo, o valor 15 será impresso:

```
mensagem = "Seja bem vindo!"
tam = len(mensagem)
print(tam)
```

Métodos para manipular strings

- Existem diversos métodos para manipulação de strings.
- Por exemplo, para transformarmos todos os caracteres de uma string em maiúscula, utilizamos os comandos:

```
msg = "Seja Bem Vindo(a)!"

res = msg.upper()
```

- `.upper()` é um método associado a strings.
- Todo método de string retorna uma nova string. Portanto, precisamos definir uma nova variável que receba o resultado.

Métodos para manipular strings

Alguns métodos úteis:

```
# Torna a primeira letra maiúscula
res = msg.capitalize()
# Retorna o índice onde ocorre a palavra "Bem"
res = msg.find("Bem")
# Retorna True se todos os caracteres forem letras e números
res = msg.isalnum()
# Retorna True se todos os caracteres forem letras
res = msg.isalpha()
# Retorna True se todos os caracteres forem números
res = msg.isdecimal()
# Retorna True se o caractere for espaço
res = msg[4].isspace()
# Retorna uma nova string com todas as letras minúsculas
res = msg.lower()
# Substitui o caractere " " (espaço) pelo caractere "-"
msg.replace(" ", "-")
```

Sequências de escape (caracteres de controle)

Alguns comandos especiais podem ser utilizados em strings para controlar a posição do cursor de texto:

Comando	Descrição
\b	Backspace, retorna 1 caractere
\t	Tabulação, equivalente à tecla tab
\n	Alimentação de linha, cria uma nova linha
\r	Carriage return, retorna ao começo da linha

Por exemplo, para imprimir três nomes, um em cada linha:

```
print("Paulo\nLaura\nLucas")
```

Sequências de escape (caracteres de controle)

- Importante! Uma sequência de escape possui tamanho 1. Por exemplo, "\n" representa um único caractere.

- Outro exemplo. No comando

```
n = len("a\nb\nc\n")
```

a variável n possuirá o valor 6.

Exercício 1 - Contagem de caractere

- Faça uma função que receba como entrada uma string possuindo um tamanho qualquer e uma outra string possuindo um único caractere. A função retorna o número de vezes que o caractere ocorre na string.

- Por exemplo, se a função receber a string "Dezembro possui trinta e um dias" e o caractere "a", ela deve retornar o valor 2, pois o caractere "a" ocorre duas vezes na string.

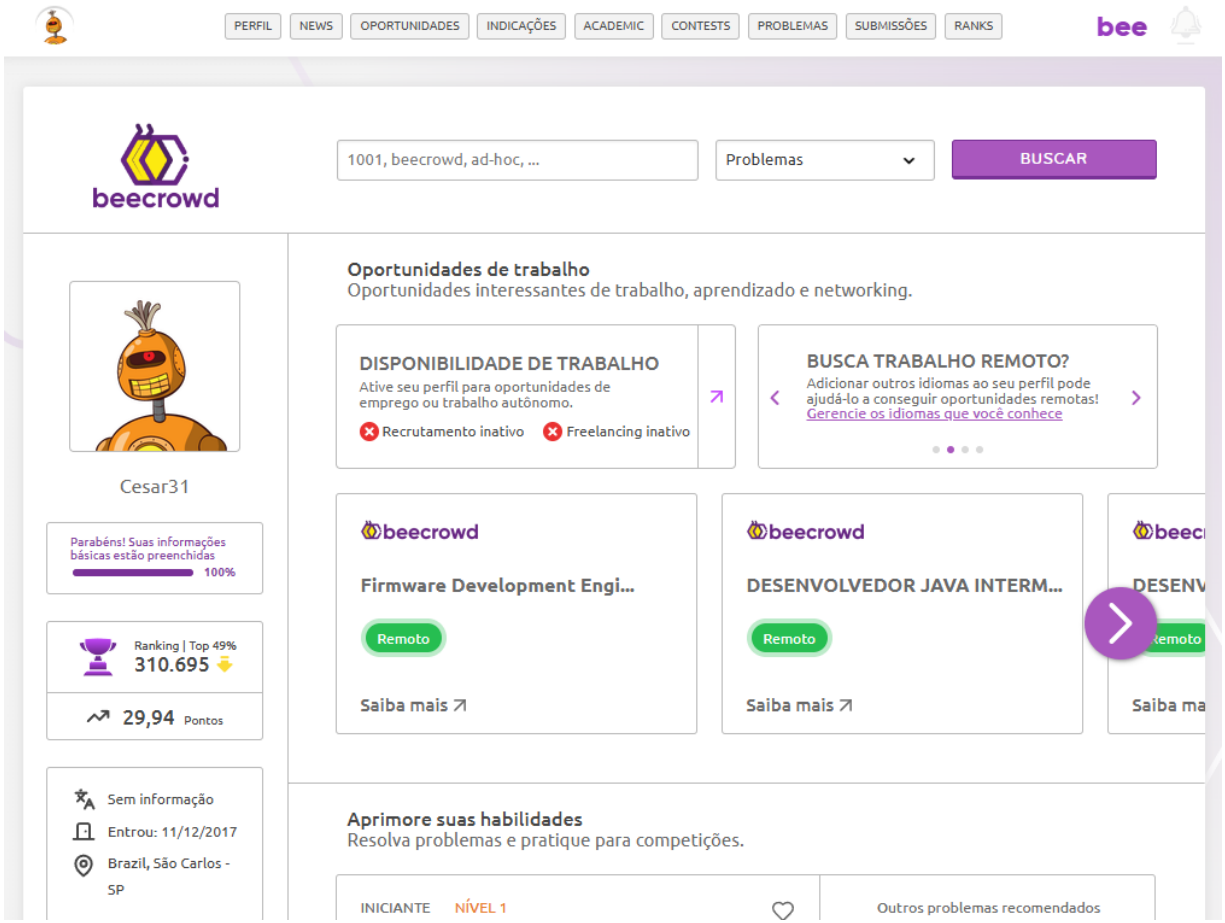
Solução

```
def contagem(texto, carac):

    n = len(texto)
    cont = 0
    for i in range(0, n):
        if texto[i]==carac:
            cont += 1

    return cont

def testes():

    texto = "Dezembro possui trinta e um dias"
    carac = "a"
    print(contagem(texto, carac))  # Valor 2 esperado

    texto = "Dezembro possui trinta e um dias"
    carac = "s"
    print(contagem(texto, carac))  # Valor 3 esperado

    texto = ""
    carac = "a"
    print(contagem(texto, carac))  # Valor 0 esperado

testes()
```

Formatação de strings para o Beecrowd

Beecrowd

- O site judge.beecrowd.com/pt é um lugar muito bom para praticar programação. Ele possui diversos problemas com correção automática



Beecrowd

- Uma dificuldade em utilizar o Beecrowd é que a entrada e saída do programa precisam estar exatamente no formato esperado pelo site.
- Veremos brevemente como formatar a entrada e saída de um programa no padrão esperado pelo site

String para tratamento de entrada de dados

- Até agora a entrada dos nossos programas sempre solicitou que o usuário digitasse um único valor de cada vez, usando a função input()
- É possível solicitar que o usuário digite vários valores de uma vez. Para isso, utilizamos o método split
- O método split é utilizado para separar os elementos de uma string de acordo com um caractere fornecido

String para tratamento de entrada de dados

- Por exemplo, suponha um programa que requisita três valores:

```
entrada = input("Digite três valores separados por espaço: ")
valores = entrada.split(" ")
```

O método .split(" ") separará os valores digitados pelo caractere espaço, gerando uma lista de três elementos

Se os valores 4 8 12 forem digitados, a variável valores será

```
["4", "8", "12"]
```

String para tratamento de entrada de dados

- Note que os valores retornados pelo método split são do tipo string

- Podemos converter os valores para o tipo adequado:

```
entrada = input("Digite o nome do produto, a quantidade e o preço: ")
valores = entrada.split(" ")

nome = valores[0]
quantidade = int(valores[1])
preco = float(valores[2])
```

Formatação da saída de dados

- Em relação à saída de dados, é possível formatar exatamente como os valores serão impressos na tela ao utilizar a função print()

- Considere uma variável valor definida como `valor = 4.5673`. Exemplos de impressão da variável:

Comando	Resultado
`print(f"{valor}")`	`4.5673`
`print(f"{valor:.1f}")`	`4.6`
`print(f"{valor:.3f}")`	`4.567`
`print(f"{valor:6.2f}")`	`  4.57`

- No formato `{valor:w.pf}`, `w` indica a largura do número impresso. Se o número possuir menos que `w` caracteres, espaços em branco são inseridos à esquerda. `p` indica o número de casas após a vírgula para arredondar o número

Leitura e escrita de arquivos

Leitura de arquivos

Podemos ler o conteúdo de um arquivo no computador através dos comandos:

```
# Abre o arquivo "texto.txt" em modo "r", que significa
# modo "read", ou modo leitura em português.
# A referência para o arquivo é armazenada na variável
# fp (acrônimo de file pointer)
fp = open("texto.txt", "r")

# Lê todo o conteúdo do arquivo e armazena o resultado na
# variável texto
texto = fp.read()

# Encerra o uso do arquivo
fp.close()
```

* É muito importante lembrar do comando fp.close() para fechar o arquivo. Se fp.close() não for usado, o conteúdo poderá não ser lido totalmente.

Leitura de arquivos

No comando

```
fp = open("texto.txt", "r")
```

`"texto.txt"` se refere ao nome e local do arquivo. Nesse caso, o arquivo precisará estar no mesmo diretório que o código.

Se o código estiver em outro local, precisamos passar o caminho completo da localização do arquivo. No Linux, o caminho será algo do tipo:

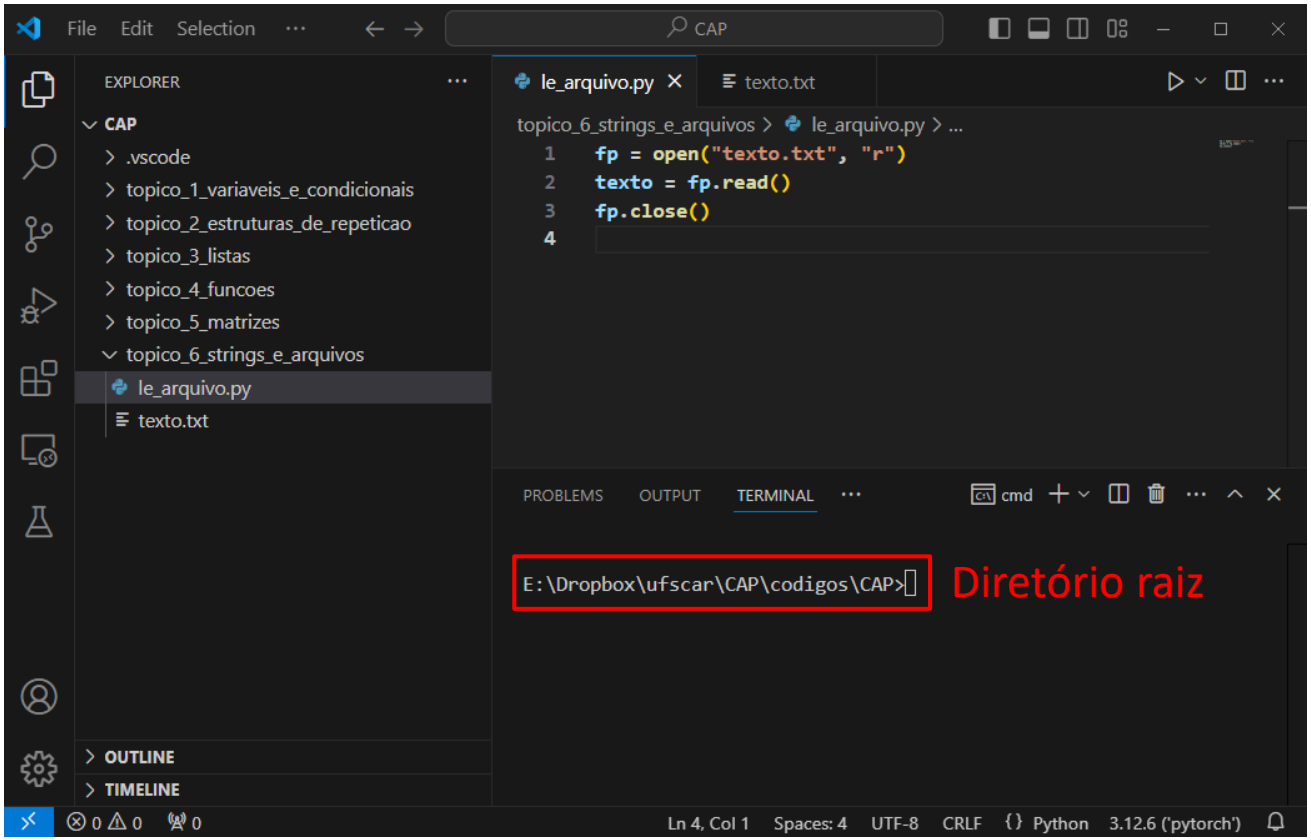
```
fp = open("/home/usuario/arquivos/texto.txt", "r")
```

No windows:

```
fp = open("C:/usuarios/usuario/documentos/texto.txt", "r")
```

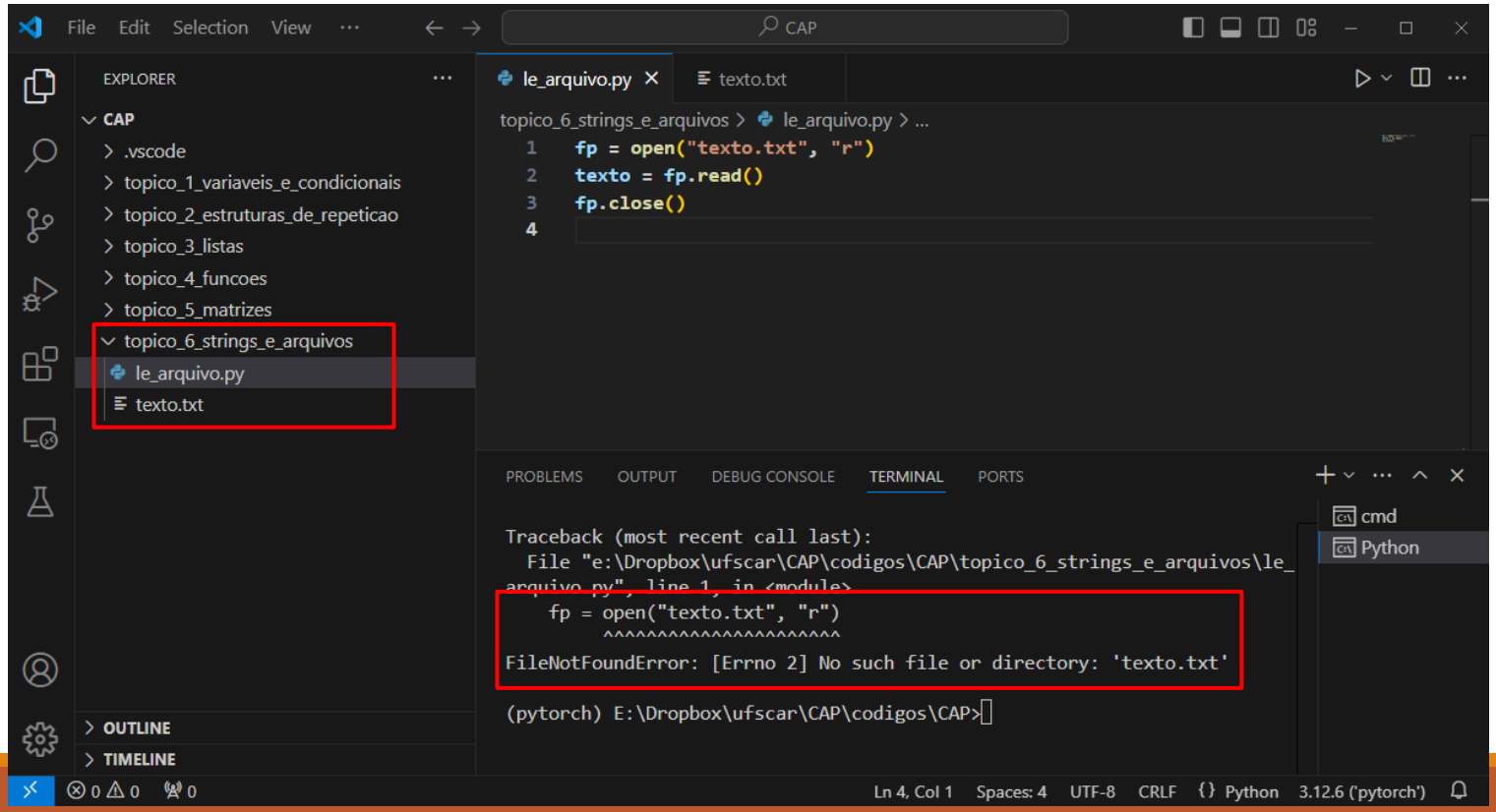
Leitura de arquivos no VSCode

Importante! O diretório padrão do VSCode é a raiz do espaço de trabalho (workspace) aberto no programa. Na imagem abaixo, a raiz do espaço de trabalho é "**E:\Dropbox\ufscar\CAP\codigos\CAP** "



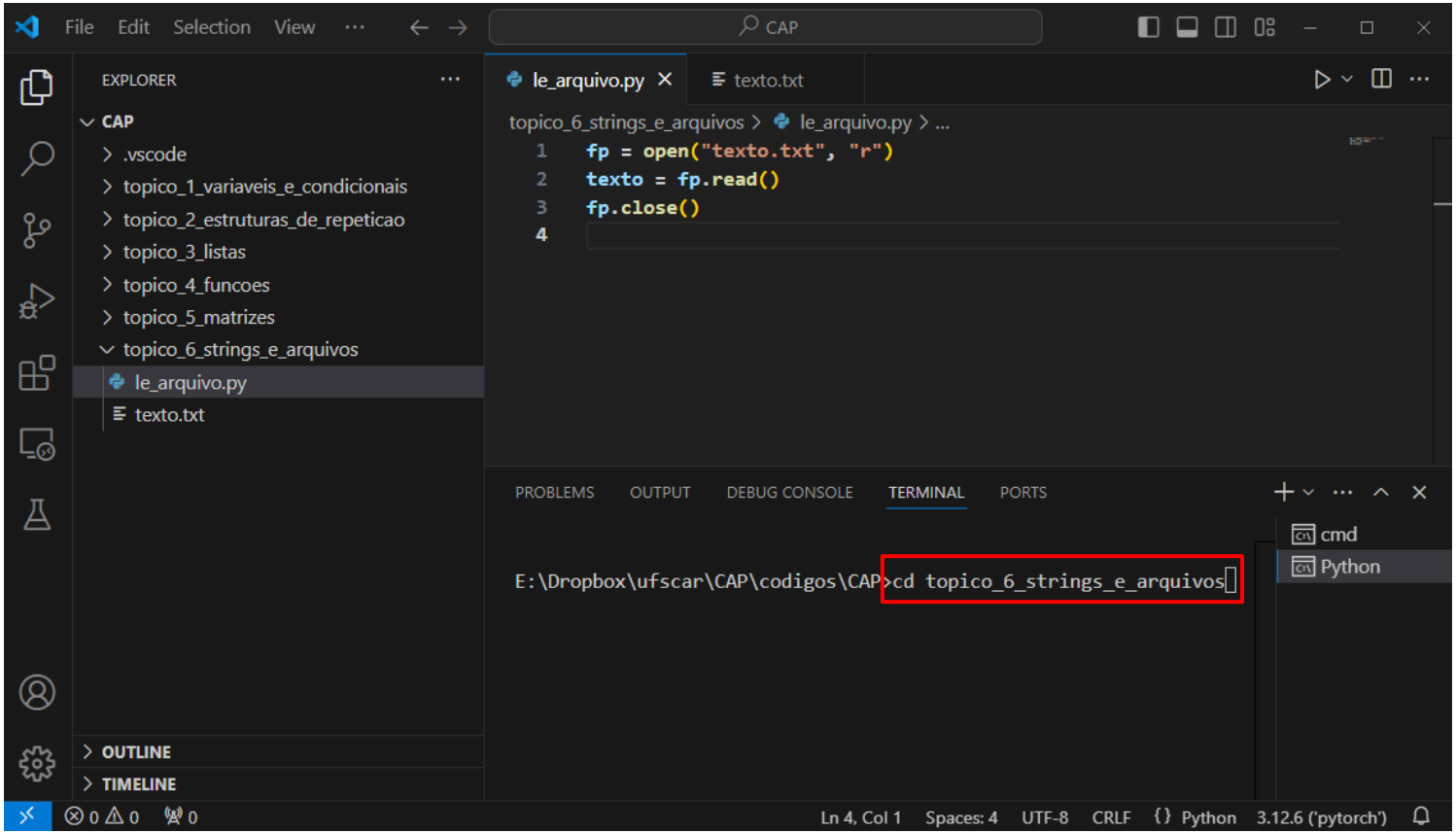
Leitura de arquivos no VSCode

Ao tentar executar o código para leitura de um arquivo, ele dará um erro de arquivo não encontrado, pois como o programa é executado no diretório raiz, o arquivo é lido no local "<**diretorio_raiz>\texto.txt**", sendo que o local correto é "<**diretorio_raiz>\topico_6_strings_e_arquivos\texto.txt**"



Leitura de arquivos no VSCode

Para resolver o problema, após executar o código uma vez, mudamos o diretório no terminal para o diretório onde o código está localizado utilizando o comando "cd"



Leitura de arquivos

- Em geral, não é adequado lermos todo o conteúdo de um arquivo.
- Se o conteúdo for muito grande, a leitura ocupará uma grande quantidade de memória
- A leitura de muito conteúdo também pode bloquear a execução de um programa, isto é, não é possível executar outras tarefas durante a leitura

Leitura de arquivos

- Temos três opções principais:
  - Leitura caractere a caractere
  - Leitura linha a linha
  - Leitura de *n* caracteres por vez, chamada de operação com *buffer* ou operação *bufferizada*

Leitura de arquivos

- Temos três opções principais:
  - Leitura caractere por caractere

```
texto = fp.read(1)
```

  - Leitura linha-a-linha

```
texto = fp.readline()
```

  - Leitura de *n* caracteres por vez, chamada de operação com *buffer* ou operação *bufferizada*

```
BUFFER_SIZE = 100

texto = fp.read(BUFFER_SIZE)
```

Leitura de arquivos

- Quando não houver mais texto a ser lido, fp.read() e fp.readline() retornam uma string vazia ""
- Portanto, para ler todo o conteúdo de um arquivo, os métodos fp.read() e fp.readline() são utilizados em uma estrutura de repetição até que a string retornada seja vazia. Por exemplo:

```
fp = open("test.txt", "r")
while True:
    caractere = fp.read(1)
    if caractere=="":
        break
fp.close()
```

Exercício 2 - Vogais

Faça uma função que receba o nome de um arquivo de texto e retorne o número de vogais minúsculas que ocorrem no texto. A função deve ler o arquivo caractere a caractere.

Por exemplo, para um arquivo contendo o texto

"Python é uma linguagem de programação de alto nível"

a função deve retornar o valor 16.

* Para o arquivo descricao.txt fornecido no AVA, a função deve retornar o valor 630

Solução

```
def vogais(nome):

    fp = open(nome, "r")
    cont = 0
    while True:
        c = fp.read(1)
        if c=="":
            break
        if c=="a" or c=="e" or c=="i" or c=="o" or c=="u":
            cont += 1
    fp.close()

    return cont
```

Escrita de arquivos

Uma string pode ser escrita em um arquivo através dos seguintes comandos:

```
texto = "Exemplo de texto em arquivo"

# Cria o arquivo texto.txt em modo "w", que significa
# modo "write", ou modo escrita em português.
# A referência para o arquivo é armazenada na variável
# fp (acrônimo de file pointer)
fp = open("texto.txt", "w")

# Escreve a string no arquivo
fp.write(texto)

# Encerra o uso do arquivo
fp.close()
```

* É muito importante lembrar do comando fp.close() para fechar o arquivo. Se fp.close() não for usado, o conteúdo poderá não ser escrito no arquivo.

Leitura e escrita de arquivos

Os modos mais comuns na função open() são:

Modo	Descrição
"r"	Modo leitura. Dá erro se o arquivo não existir
"w"	Modo escrita. Cria o arquivo se ele não existir. Apaga o conteúdo se o arquivo existir
"a"	Modo escrita. Cria o arquivo se ele não existir. Insere novos valores ao final do arquivo
"b"	Modo binário

Exercício 3 - Nomes

Faça um programa que receba do usuário nomes de pessoas, um nome por linha, e escreva cada nome em um arquivo de texto, um nome por linha.

* Use `fp.write(f"{nome}\n")` para escrever o nome no arquivo.

Solução

```
def recebe_nomes():

    fp = open("nomes.txt", "w")
    while True:
        nome = input("Digite um nome ou 's' para encerrar: ")
        if nome=="s":
            break
        fp.write(f"{nome}\n")
    fp.close()

recebe_nomes()
```

Exercício 4 - Base de dados

Faça um programa que receba informações sobre pessoas e armazene essas informações em um arquivo de texto. O programa também possibilita que as informações armazenadas sejam impressas na tela.

As informações a serem armazenadas para cada pessoa são: nome, idade, profissão e se a pessoa possui emprego atualmente. **A informação para cada pessoa é escrita em uma linha do arquivo.**

O programa possui o seguinte menu:

Menu
---
1. Inserir informação
2. Imprimir dados
3. Encerrar

* Use o modo "a" na função open para escrever no arquivo sem apagar o conteúdo.

Solução

Função principal:

```
def main():

    nome_arquivo = "dados.txt"
    # Cria o arquivo no início da execução do programa para garantir
    # que ele existe na chamada da função imprimir_dados
    cria_arquivo(nome_arquivo)

    print("Seja bem vindo(a)!")

    while True:

        print("Menu")
        print("-----")
        print("1. Inserir informação")
        print("2. Imprimir dados")
        print("3. Encerrar")
        opcao = input("Digite a opção: ")

        if opcao=="1":
            inserir_dados(nome_arquivo)
        elif opcao=="2":
            imprimir_dados(nome_arquivo)
        elif opcao=="3":
            print("Até mais!")
            break
        else:
            print("Opção não reconhecida")

main()
```

Solução

Funções auxiliares:

```
def cria_arquivo(nome_arquivo):
    """Cria um arquivo vazio."""

    fp = open(nome_arquivo, "a")
    fp.close()

def inserir_dados(nome_arquivo):
    """Requisita valores do usuário e os insere em um arquivo."""

    dados = input("Digite o nome, idade, profissão e se está empregado (sim/não): ")
    fp = open(nome_arquivo, "a")
    fp.write(f"{dados}\n")
    fp.close()

def imprimir_dados(nome_arquivo):
    """Imprime o conteúdo do arquivo."""

    fp = open(nome_arquivo, "r")
    while True:
        linha = fp.readline()
        if linha=="":
            break
        print(linha)
    fp.close()
```

Solução v2

É adequado a função inserir_dados requisitar um valor por vez. Isso possibilita realizar uma verificação se os dados estão no formato correto e também salvá-los em um formato específico. Por exemplo, separados por vírgula:

```
def inserir_dados(nome_arquivo):
    """Requisita valores do usuário e os insere em um arquivo."""

    nome = input("Digite o nome: ")
    idade = input("Digite a idade: ")
    prof = input("Digite a profissão: ")
    empregado = input("Atualmente empregado (sim/não): ")

    dados = f"{nome} {idade} {prof} {empregado}\n"
    fp = open(nome_arquivo, "a")
    fp.write(dados)
    fp.close()
```