centro universitário FEI

CC1612
Fundamentos de Algoritmos

Prof. Danilo H. Perico

Arquivos

Arquivos

- **Utilizar arquivos** é uma **forma de garantir o armazenamento permanente dos dados** que são importantes no seu programa, pois nenhuma variável, nem mesmo a lista, continua existindo depois que o programa termina.
- Então, utilizar um arquivo é uma maneira excelente de trabalhar com a entrada e a saída de dados para os programas.
- **Arquivos são linhas de texto**, normalmente salvos com a extensão ***.txt*** ou ***.dat***

Arquivos

- Na programação, assim como na nossa interação com o computador, o primeiro passo para acessar um arquivo é abri-lo.
- Para abrir o arquivo, utilizamos a função ***open***
- Sintaxe:

```
arquivo = open("teste.dat", "w")
```

  - A variável ***arquivo*** salva o arquivo em si. É por meio desta variável que executaremos as funções de escrita e leitura.
  - ***open( )*** tem dois parâmetros: **nome do arquivo** e **modo de acesso**

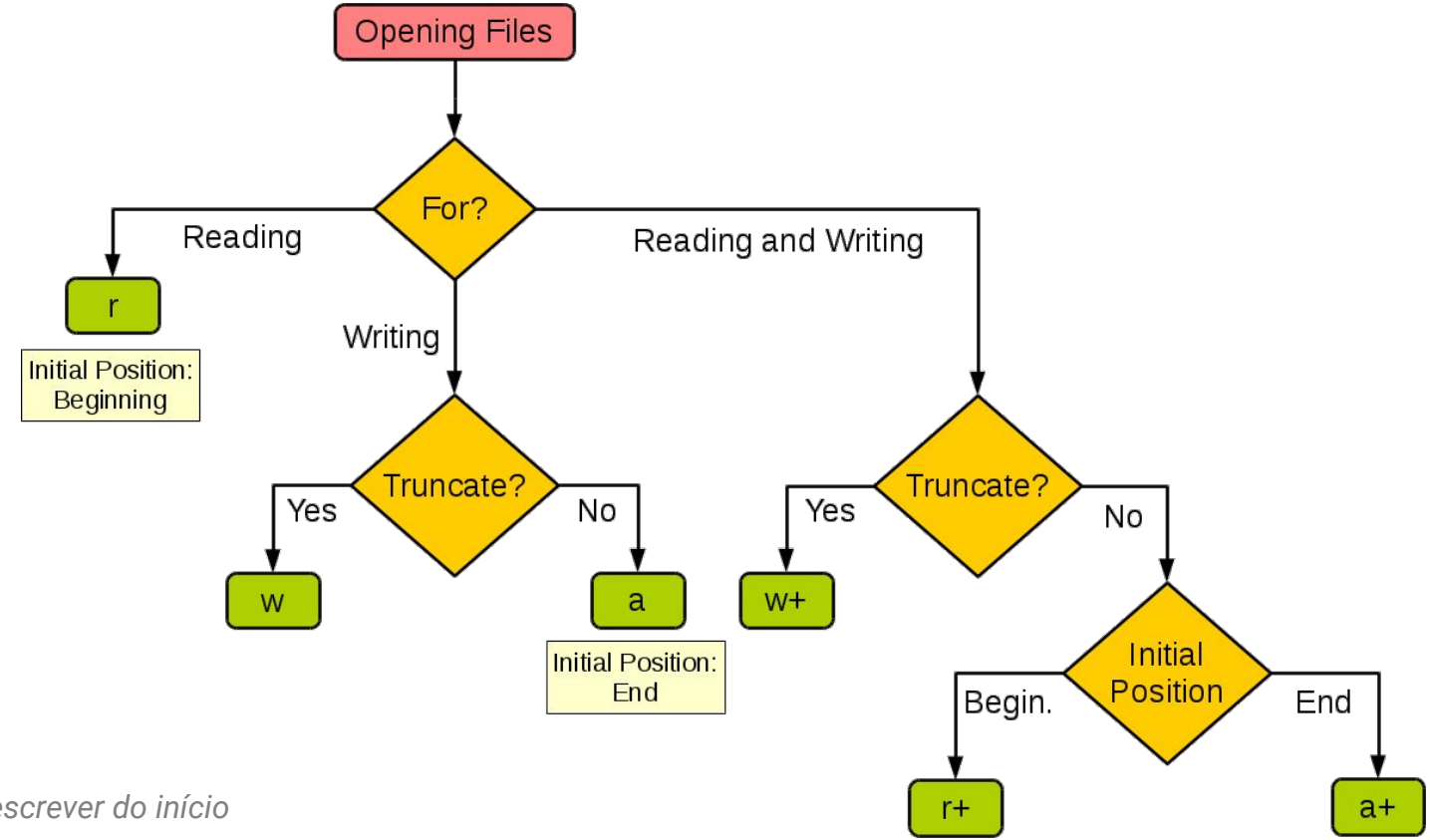
Arquivos

- Os modos de acesso mais comuns são:

modo	operação
r	leitura (*read*)
w	escrita (*write*)
a	escrita, preservando o conteúdo existente (*append*)
+	atualização (combinado com r ou w)

```
arquivo = open("teste.dat", "w")
```

Modos de Abertura de Arquivo



**truncar: sobrescrever do início*

Arquivos - Escrita

- Para escrever no arquivo, utilizamos o método ***write***, que vai ser chamado pela variável ***arquivo****:*

```
arquivo.write("texto a ser escrito no arquivo")
```

- O método ***write*** funciona de maneira similar que o ***print*** com marcadores ***(%d, %f, %s)***, porém precisamos sempre incluir o ***“\n”*** quando queremos ir para a próxima linha.

Arquivos - Escrita

- Depois que escrevemos no ***arquivo***, precisamos fechá-lo, utilizando o método ***close***

```
arquivo.close()
```

- É sempre importante fechar o arquivo para informar ao Sistema Operacional que não vamos mais utilizá-lo.
- Muitas vezes, o Sistema Operacional salva as informações que queremos escrever em uma memória auxiliar e deixa a operação de escrever realmente no arquivo só quando informamos que vamos fechá-lo.
- Então, se não fechamos, corremos o risco de perder o que gostaríamos de escrever.

Arquivos - Escrita

- Exemplo completo de escrita em arquivo texto:

```
arquivo = open("teste.dat", "w")

for linha in range(1,101):
    arquivo.write("Linha %d\n" % linha)
arquivo.close()
```

Arquivo: teste.dat:

```
Linha 1
Linha 2
Linha 3
Linha 4
Linha 5
Linha 6
Linha 7
Linha 8
Linha 9
Linha 10
Linha 11
Linha 12
Linha 13
Linha 14
Linha 15
```

Arquivos - Leitura

- Para ler do arquivo, precisamos seguir o mesmo procedimento:
  - Abrir o arquivo em modo leitura ***“r”***

```
arquivo = open("teste.dat", "r")
```

  - Utilizar um método para ler o arquivo
  - Fechar o arquivo com o método ***close***

Arquivos - Leitura

- Para ler do arquivo, podemos utilizar o método ***readlines()***
- Exemplo:

```
arquivo = open("teste.dat", "r")

for linha in arquivo.readlines():
    print(linha)
arquivo.close()
```

```
Linha 1

Linha 2

Linha 3

Linha 4

Linha 5

Linha 6
```

Arquivos

- Exemplo: gerar e gravar números pares e ímpares em arquivos separados. Números de 0 a 999.

```
impares = open("impares.txt", "w")

pares = open("pares.txt", "w")

for n in range(1000):
    if n % 2 == 0:
        pares.write("%d\n" % n)
    else:
        impares.write("%d\n" % n)
impares.close()
pares.close()
```

Arquivos

- Podemos realizar diversas operações com os arquivos
- Por exemplo:
  - Ler
  - Processar
  - Gerar novos arquivos

Arquivos

- Exemplo: Utilizando o arquivo ***“pares.txt”,*** gerado no último exemplo, vamos criar outro arquivo que deve conter somente os números múltiplos de 4.

```
multiplos4 = open("multiplos_4.txt", "w")

pares = open("pares.txt", "r")

for linha in pares.readlines():
    if int(linha) % 4 == 0:
            multiplos4.write(linha)
pares.close()
multiplos4.close()
```

Arquivos

- Até agora, estamos utilizando somente um dado por linha
- Porém, podemos salvar informações correlatas na mesma linha
- Exemplo:
    - Criar um arquivo com o nome e o telefone de pessoas, conforme são digitados pelo usuário. O programa deve funcionar em loop até que o nome digitado seja vazio.

Arquivos - Escrita de dois dados na mesma linha

```
1  contatos = open("contatos.dat", "w")
2  nome = input("Nome: ")
3  telefone = input("Telefone: ")
4
5  while nome != "":
6      contatos = open("contatos.dat", "a")
7      contatos.write("%s %s\n" % (nome, telefone))
8      contatos.close()
9      nome = input("Nome: ")
10     telefone = input("Telefone: ")
11
```

```
Nome: fulano
Telefone: 123456
Nome: sicrano
Telefone: 9876543
Nome: beltrano
Telefone: 5555555
```

Arquivos - Leitura de dois dados na mesma linha

- Entendendo melhor o ***readlines()***
  - O ***readlines()*** retorna uma lista onde cada uma das linhas ocupa uma posição/ índice:

```
1 contatos = open("contatos.dat", "r")
2
3 conteudo_do_arquivo = contatos.readlines()
4
5 print(conteudo_do_arquivo)
```

```
['fulano 123456\n', 'sicrano 9876543\n', 'beltrano 5555555\n']
```

Método **split(x)**

- Divide as informações no caractere informado como parâmetro
- Exemplo:

```
nome, telefone = input("Entre com o nome e o telefone: ").split(" ")

print(nome)
print(telefone)
```

```
Entre com o nome e o telefone: fulano 1234567
fulano
1234567
```

Arquivos

- Como ler uma linha com duas informações?

```
1  contatos = open("contatos.dat", "r")
2
3  contato = []
4
5  for linha in contatos.readlines():
6      linha_separada = linha.split(" ")
7      contato.append(linha_separada)
8
9  print(contato)
10 print(contato[0])
11 print(contato[0][0])
12 print(contato[0][1])
```

Lista de listas

```
[['fulano', '123456\n'], ['sicrano', '9876543\n'], ['beltrano', '5555555\n']]
['fulano', '123456\n']
fulano
123456
```

Exercícios

1. Crie um programa que inverta a ordem das linhas do arquivo *pares.txt*. A primeira linha deve conter o maior número e a última linha o menor. Salve o resultado em outro arquivo, chamado *pares_invertido.txt*.

pares.txt

```
0
2
4
6
8
10
.
.
.
998
```

pares_invertido.txt

```
998
996
994
992
990
988
.
.
.
0
```

Exercícios

2. Escreva uma função em Python para retornar a somatória de todos os números que estão armazenados no arquivo *“numeros2.txt”*. Todos os números do arquivo estão na mesma e única linha, separados por espaço.

3. Escreva uma função que leia uma sequência numérica do arquivo *“numeros3.txt”* e salva os números na lista *num*. Esta função deve retornar *num*. Escreva outra função que recebe a lista *num* como parâmetro e retorna uma nova lista *num_unicos,* sem os elementos repetidos. Escreva uma terceira função que recebe a lista *num_unicos* e grava os números no arquivo “*numeros3unicos.txt*”

CRUD

- **CRUD**: acrônimo de ***Create***, ***Read***, ***Update*** e ***Delete***
- **Criar, Ler, Atualizar e Deletar**:
  - As quatro operações básicas utilizadas em bases de dados para realizar a interface de criação, consulta, atualização e destruição de dados.
  - **Podemos utilizar a ideia do CRUD para trabalhar com arquivos!**

Exercício

5. Crie uma agenda de telefones que salva os dados de maneira permanente. A agenda deve funcionar em loop infinito, até que o usuário decida sair. Os dados armazenados são: ***nome sobrenome***, ***telefone*** e ***e-mail***. A agenda deve apresentar o seguinte menu para o usuário:

Opções:

1 - Novo contato *(Create)*

2 - Procura (pelo nome) *(Read)*

3 - Atualiza contato *(Update)*

4 - Apaga contato *(Delete)*

0 - Sai

Passo a passo - Ex.5

1. Crie uma função para cada operação da agenda
2. A sua agenda deve gerenciar vários arquivos. Um para cada contato. O nome dos arquivos deve ser dado seguindo a regra: ***nome_sobrenome.txt***. Dentro de cada arquivo você salva o ***telefone*** e o ***e-mail***.
3. Para cada operação, abra, processe e feche o arquivo.
4. Para apagar, você pode deletar o arquivo pelo seu nome (que é também o nome do contato): ***nome_sobrenome.txt***. Para deletar um arquivo do sistema operacional pelo Python, você faz:

```
import os

os.remove(“nome_do_arquivo.txt”)
```