Paradigmas de Linguagem de Programação em Python

Arquivos

Prof. Henrique Mota

arquivos.ipynb

Introdução

- Até então, vinhemos utilizamos o interpretador para executar programas em **`Python`**.
- Olhando para os dados, percebemos que todos eles desaparecem quando o programa é encerrado, pois são armazenados na **memória volátil** (**RAM**).
- Desse modo, em várias aplicações, precisamos de uma forma de **armazenar dados permanentemente**.
- Trabalharemos então com **`Arquivos`**: uma excelente forma de entrada e saída de dados para programas.

Introdução

- Com **`Arquivos`**, poderemos ler e salvar dados não apenas de nossos programas, mas também de outros (até mesmo da internet).
- Mas o que são esses dados?
  - Variáveis, mensagens, vetores, listas, texto, etc
- E formalmente, o que é um **`Arquivos`**?
  - É **uma área da memória <u>não-volátil</u>** (e.g., disco rígido ou SSD), na qual podemos realizar operações de **`leitura & escrita`**. Essa área é gerenciada pelo Sistema Operacional (SO).

Introdução

- O fluxo de trabalho num **`arquivo`** consiste em
  - **Abertura**;
  - **Operações** (**`leitura`** **`&`** **`escrita`**);
  - **Fechamento**.

Arquivos: Abertura

- Para acessar um arquivo, precisamos abri-lo.
- Nessa etapa, informamos:
  1. O nome do arquivo que queremos abrir (incluindo o diretório em que ele se encontra);
  2. O tipo de operação queremos realizar (modo): **`leitura`** e/ou **`escrita`**.
- Em **`Python`**, abrimos arquivos com a função **`open`**, que recebe os parâmetros nome e modo.

Arquivos: Abertura

Modo	Operações
`r`	Leitura `[default]`
`w`	escrita (apagando o conteúdo anterior, se já existir)
`a`	escrita (preservando o conteúdo anterior, se já existir)
`t`	modo texto `[default]`
`b`	modo binário
`+`	Atualização (leitura e escrita)

- Os modos podem ainda ser combinados (`"r+"`, `"w+"`, `"a+"`, `"r+b"`, `"w+b"`, `"a+b"`)

Arquivos: Abertura

- O SO, então, prepara esse arquivo na memória não-volátil e abre uma região na memória volátil (RAM) que serve de **interface** entre o Python e o arquivo.
- Ao término da abertura, temos uma **variável** na RAM que nos permitir´a operar no arquivo.

Arquivos: Abertura

- Assuma que você tem o seguinte **arquivo**, na mesma pasta do script **Python**

arq_teste.txt

```
Olá! Bem vindo ao nosso arquivo teste.
Ele será utilizando puramente para isso: testar.
Boa sorte!
```

Arquivos: Abertura

- Para abri-lo, utilize a função **open()**

```
arq = open("arq_teste.txt", "r")
```

- Ou para um arquivo em outro diretório:

```
arq = open("/home/pedro/documents/arq_teste.txt", "r")
```

Arquivos: Operações

- Vamos agora realizar operações com os arquivos.
- Como primeiro exemplo, vamos criar um arquivo e nele escrever alguns dados
- **Escrita**:

```
arq = open("numeros.txt", "w")
valores = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

for v in valores:
    arq.write("%d\n" % v)
```

Arquivos: Operações

- Lembre-se que o modo **`w`** **cria** um novo **arquivo**, **caso** ele **não exista**

- **`Escrita`**:

```
arq = open("numeros.txt", "w")
valores = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

for v in valores:
    arq.write("%d\n" % v)
```

Arquivos: Operações

- Perceba que o **\n** sai criando os elementos dispostos em linhas diferentes
- A função **write** faz a escrita acontecer oficialmente.
- **Escrita**:

```
arq = open("numeros.txt", "w")
valores = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

for v in valores:
    arq.write("%d\n" % v)
```

Arquivos: Operações

- Agora vamos ler o arquivo recém criado? Para isso, vamos abrir o arquivo em modo leitura (**r**).
- Lembre-se que se o arquivo não existir, ocorrerá um erro.
- **Leitura**:

```
arq = open("numeros.txt", "r")
linhas = arq.readlines()

for l in linhas:
 print(l)
```

Arquivos: Operações

- Utilizamos a função **`readlines()`**, que retorna uma linha por vez - cada linha terminada em **`\n`**
- A operação de leitura **sempre** retorna **`string`**
- **`Leitura`**:

```
arq = open("numeros.txt", "r")
linhas = arq.readlines()

for l in linhas:
 print(l)
```

Arquivos: Operações

- Então, se estivermos trabalhamos com tipos diferentes, como números, devemos fazer as devidas conversões:
  - **`int()`**, **`float()`**, etc
- **Leitura**:

```
arq = open("numeros.txt", "r")
linhas = arq.readlines()

for l in linhas:
 print(l)
```

Arquivos: Operações

- Se quisermos abrir no modo leitura e escrita, podemos utilizar **`w+`** ou **`r+`**
  - Recomendo a segunda opção, pois nao apaga conteúdo pré-existente

Arquivos: Fechamento

- Ao término das operações, é importante informar ao SO.
- Mas porque?
  1. Para efetivar operações que por ventura alteraram conteúdo do arquivo;
  2. Avisar ao SO que aquela região da RAM não será mais necessária para fazer a interface com o arquivo
- Para isso, utilizamos a função `close()`
- Atualizando os códigos anteriores, teríamos...

Arquivos: Fechamento

```
arq = open("numeros.txt", "w")
valores = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

for v in valores:
    arq.write("%d\n" % v)


arq.close()
```

Arquivos: Fechamento

```
arq = open("numeros.txt", "r")
linhas = arq.readlines()

for l in linhas:
    print(l)

arq.close()
```

Arquivos: Fechamento

- Ainda, **`Python`** possui um recurso que garante o fechamento de arquivos, mesmo se não utilizarmos o **`close`**.
  - Isso é importante porque nosso programa pode ser interrompido ou simplesmente parar antes de fecharmos um arquivo
- A estrutura **`with`** permite criarmos um contexto, ou seja, um bloco em que um objeto é válido

Arquivos: Fechamento

```
with open("numeros.txt", "r") as arq:
    for linha in arq.readlines():
            print(l)
```

- O **with** funciona atribuindo o resultado do **open** à variável **arq**.
- No caso de contexto de arquivos, é como se tivéssemos chamado o close manualmente
- Dessa forma, garantimos o fechamento e evitamos esquecimentos comuns
- O **with** ainda protege arquivos de exceções

Arquivos: Cursor

- Um conceito importante em arquivos é o de cursor.
- O cursor é como se fosse um apontador para a região “atual” do arquivo.
- Para esse propósito, temos a função `seek()`, que recebe o parâmetro indicando a posição de apontamento
  - `0`: 1º caractere
  - `1`: 2º caractere
  - etc...

Arquivos: Cursor

```
# modo leitura: cursor no inicio do arquivo
arq = open("numeros.txt", "r")
linhas = arq.readlines()

# a cada readlines , cursor anda uma linha
for l in linhas:
 print(l)
# no final do for, cursor no final do arquivo
# nao tem mais nada para ler

# podemos voltar o cursor ao inicio
arq.seek(0)

# podemos usar readlines novamente
print(arq.readlines())
arq.close()
```

Arquivos
Praticando...

- Assuma que você tem o seguinte **arquivo**, na mesma pasta do script **Python**

estoque.txt

```
Abacate 30 6.5
Kiwi 20 11.0
Morango 5 15.0
```

- Podemos acessar os elementos de cada linha via **split()**

Arquivos
Praticando...

```
arq = open("estoque.txt", "r")
for linha in arq.readlines():
    fruta, qtde, preco = linha.split()

arq.close()
```

Arquivos
Utilizando Numpy

```
import numpy as np

# armazenando apenas as frutas
frutas = np.loadtxt("estoque.txt", usecols=[0],
dtype=np.str)

# armazenando apenas os valores, sem as strings
valores = np.loadtxt("estoque.txt", usecols=[1, 2])

# salvando os vetores
np.savetxt("frutas.txt", frutas, fmt="%s")
np.savetxt("valores.txt", valores, fmt="%d %.2f")
```

Arquivos
Utilizando Numpy

- Utilizando arquivos com Numpy, não precisamos fechá-los!
  - Seu conteúdo já é armazenado imediatamente num vetor numpy - a própria função já fecha o arquivo.
- O formato `%.2f` significa: **float** com apenas **2 casas decimais**.
- Dúvidas?
  - Verifique a documentação oficial das funções de arquivo em Numpy:

    https://numpy.org/doc/stable/reference/routines.io.html

Obrigado!

Alguma dúvida?

Prof. Henrique Mota

profhenriquemota@gmail.com