Algoritmos e Programação de Computadores

Arquivos

Prof. Edson Borin
Instituto de Computação (IC/Unicamp)

Agenda

- Arquivos textos
  - Abrindo um arquivo texto
  - Lendo um arquivo texto
  - Escrevendo um arquivo texto
- Extra:
  - Parâmetros do programa: sys.argv

Arquivos Textos

- Até agora, nós vimos no curso exemplos de programas que obtiveram os dados de entrada de usuários via teclado.

```
nomes = input()
```

- A maioria desses programas pode receber seus dados de entrada de **arquivos texto** também.
- Um arquivo texto armazena caracteres que podem ser mostrados diretamente na tela ou modificados por um editor de textos simples. Exemplos: código python, documento texto simples, páginas HTML.

Arquivos Textos

- Quando comparado à entrada de dados via teclado, as principais vantagens de se obter dados de entrada de um arquivo são:
  - O conjunto de dados pode ser muito maior.
  - Os dados podem ser inseridos muito mais rapidamente e com menos chance de erro.
  - Os dados podem ser usados repetidamente com o mesmo programa ou com diferentes programas.

Arquivos Textos

- Um **nome** e **caminho** únicos são usados por usuários ou em programas ou *scripts* para acessar um arquivo texto para fins de leitura e modificação.
- As tarefas básicas envolvidas na manipulação de arquivos são **ler** dados de arquivos e **escrever** ou anexar dados em arquivos.
- Leitura e escrita em arquivos em Python são muito fáceis de gerenciar.

Arquivos Textos

- Para se trabalhar com arquivos devemos abri-lo e associá-lo com uma variável.
- A variável será um objeto do tipo `file` que contém métodos para ler e escrever no arquivo.
- O primeiro passo então é abrir o arquivo com o comando `open`:

```
variavel_arquivo = open("nome do arquivo", "modo")
```

Arquivos Textos

- O primeiro passo então é abrir o arquivo com o comando `open`:

```
variavel_arquivo = open("nome do arquivo", "modo")
```

- O `"nome do arquivo"` pode ser relativo ou absoluto.
- O `"modo"` pode ser `"r"`(leitura), `"r+"`(leitura e escrita), `"w"`(escrita), `"a"`(append).

Abrindo um Arquivo Texto para Leitura

```
arquivo = open("tarefas.txt", "r")
```

- O primeiro parâmetro para `open` é uma string com o nome do arquivo
  - Pode ser absoluto, por exemplo: `"/home/edson/tarefas.txt"`
  - Pode ser relativo como no exemplo acima: `"tarefas.txt"`
- O segundo parâmetro é uma string informando como o arquivo será aberto. Se para leitura ou gravação de dados, ou ambos.
  - No nosso exemplo o `"r"` significa que abrimos um arquivo texto para leitura.

Abrindo um Arquivo Texto para Leitura

- Se abrirmos um arquivo para leitura e ele não existir, ocorrerá um erro:

```
arquivo = open("notas-exame-final.txt", "r")
```

```
---------------------------------------------------------------
FileNotFoundError
Traceback (most recent call last)
<ipython-input-2-9a68a6e13907> in <module>()
----> 1 arquivo = open("notas-exame-final.txt", "r")

FileNotFoundError: [Errno 2] No such file or directory:
'notas-exame-final.txt'
```

Abrindo um Arquivo Texto para Leitura

- Se o arquivo não existir, podemos tratar o erro usando os comandos `try except` de Python.
- Todo **erro** em python gera o que chamamos de **exceção**.
- Quando comandos são executados dentro de um bloco `try`, se ocorrer uma exceção automaticamente passa a ser executado os comandos do bloco `except`.

```
try:
    comandos que podem gerar exceção
except:
    comandos executados se houver alguma exceção
```

Abrindo um Arquivo Texto para Leitura

- Ao se trabalhar com arquivos é bom colocar a abertura do arquivo no bloco `try`, e o tratamento da exceção no bloco `except`.

```
try:
    arquivo = open("tarefas.txt", "r")
    print("Abri o arquivo com sucesso.")
except:
    print("Não foi possível abrir o arquivo.")
```

Lendo Dados de um Arquivo Texto

- Para ler dados do arquivo aberto, usamos o método `read`.
  - `read(num bytes)`: Retorna uma string contendo os próximos `num bytes` do arquivo.
  - `read()`: Sem parâmetro é retornado uma string contendo **todo** o arquivo!

```
try:
    arquivo = open("tarefas.txt", "r")
    conteudo = arquivo.read()
except:
    print("Não foi possível abrir o arquivo.")
```

Lendo Dados de um Arquivo Texto

- Quando um arquivo é aberto, um **indicador de posição** no arquivo é criado, e este recebe a posição do início do arquivo.
- Para cada dado lido do arquivo, este indicador de posição é automaticamente incrementado para o próximo dado não lido.
- Eventualmente o indicador de posição chega ao fim do arquivo:
  - O método `read` devolve uma string vazia caso o indicador de posição esteja no fim do arquivo.

Lendo Dados de um Arquivo Texto

- O exemplo mostra o conteúdo do arquivo `"tarefas.txt"` na tela.

```
try:
    arquivo = open("tarefas.txt", "r")
    while True:
        s = arquivo.read(1)
        print(s, end="")
        if (s == ""):
            break
    arquivo.close()
except:
    print("Não foi possível abrir o arquivo.")
```

Lendo Dados de um Arquivo Texto

- O método `close` deve sempre ser usado para fechar um arquivo que foi aberto.
  - Quando escrevemos dados em um arquivo, este comando garante que os dados serão efetivamente escritos no arquivo.
  - Ele também libera recursos que são alocados para manter a associação da variável com o arquivo.

Lendo Dados de um Arquivo Texto

- O programa pode ser alterado para ler todo o arquivo de uma vez.
  - Lembre-se que se o arquivo for muito grande isto pode acarretar em uma sobrecarga da memória do seu computador fazendo com que este fique lento ou mesmo trave.

```
try:
    arquivo = open("tarefas.txt", "r")
    s = arquivo.read()
    print(s, end="")
    arquivo.close()
except:
    print("Não foi possível abrir o arquivo.")
```

```
try:
    arquivo = open("tarefas.txt", "r")
    while True:
        s = arquivo.read(1)
        print(s, end="")
        if (s == ""):
            break
    arquivo.close()
except:
    print("Não foi possível abrir o arquivo." )
```

```
try:
    arquivo = open("tarefas.txt", "r")
    s = arquivo.read()
    print(s, end="")
    arquivo.close()
except:
    print("Não foi possível abrir o arquivo." )
```

Lendo Dados de um Arquivo Texto

- Uma maneira mais eficiente do que se ler um *byte* por vez e menos arriscada do que se ler todo o arquivo de uma única vez, é ler uma linha por vez.
- Para isso usamos o método `readline()` que devolve uma linha do arquivo em formato string.

Lendo Dados de um Arquivo Texto

```
try:
    arquivo = open("tarefas.txt", "r")
    while True:
        s = arquivo.readline()
        print(s, end="")
        if (s == ""):
            break
    arquivo.close()
except:
    print("Não foi possível abrir o arquivo.")
```

Lendo Dados de um Arquivo Texto

- Notem que ao realizar a leitura de um caractere, ou uma linha, automaticamente o indicador de posição do arquivo se move para o próximo caractere (ou linha).
- Ao chegar no fim do arquivo o método `read(readline())` retorna a string vazia.
- Para voltar ao início do arquivo novamente você pode fechá-lo e abri-lo mais uma vez, ou usar o método `seek`.

Lendo Dados de um Arquivo Texto

- `seek(offset, from_what)`: o primeiro parâmetro indica quantos bytes se move a partir do valor inicial `from_what`.
- Os valores de `from_what` podem ser:
  - 0: indica início do arquivo.
  - 1: indica a posição atual no arquivo.
  - 2: indica a posição final do arquivo.
- O programa a seguir imprime duas vezes o conteúdo do arquivo `"tarefas.txt"`.

```
try:
    arquivo = open("tarefas.txt", "r")
    while True:
        s = arquivo.readline()
        print(s, end="")
        if (s == ""):
            break
    arquivo.seek(0,0) #mover indicador de posição
                      #0 bytes a partir do início
    while True:
        s = arquivo.readline()
        print(s, end="")
        if (s == ""):
            break
    arquivo.close()
except:
    print("Não foi possível abrir o arquivo.")
```

Escrevendo Dados de um Arquivo Texto

- Para escrever em um arquivo, ele deve ser aberto de forma apropriada usando o modo `"w"`, `"a"` ou `"r+"`.
- `arquivo = open("nome do arquivo", "modo")`
  - `"w"`: se o arquivo existir ele será sobreescrito, ou seja todo o conteúdo anterior será apagado.
  - `"a"`: o indicador de posição ficará no fim do arquivo, e dados escritos serão adicionados no fim do arquivo.
  - `"r+"`: o indicador de posição ficará no início do arquivo, e dados serão escritos sobre dados anteriores.

Escrevendo Dados de um Arquivo Texto

- **Sobrescreve** o início do arquivo `"tarefas.txt"`:

```
try:
    arquivo = open("tarefas.txt", "r+")
    arquivo.write("Alterei o começo do arquivo\n")
    arquivo.close()
except:
    print("Erro no arquivo.")
```

Escrevendo Dados de um Arquivo Texto

- **Adiciona** mais um texto no fim do arquivo `"tarefas.txt"`.

```
try:
    arquivo = open("tarefas.txt", "a")
    arquivo.write("Adicionei no fim do arquivo\n")
    arquivo.close()
except:
    print("Erro no arquivo.")
```

Escrevendo Dados de um Arquivo Texto

- **Apaga** todo conteúdo anterior e escreve um novo texto.

```
try:
    arquivo = open("tarefas.txt", "w")
    arquivo.write("Arquivo novo do zero\n")
    arquivo.close()
except:
    print("Erro no arquivo.")
```

Resumindo .... `open`

```
arquivo = open("nome do arquivo", "modo")
```

modo	operador	indicador de posição
`r`	leitura	início do arquivo
`r+`	leitura e escrita	início do arquivo
`w`	escrita	início do arquivo
`a`	(append) escrita	final do arquivo

Resumindo .... `open`

- Se um arquivo for aberto para leitura (`r`) e ele não existir, `open` gera um erro.
- Se um arquivo for para escrita (`w`) e existir, ele é sobrescrito. Se o arquivo não existir, um novo arquivo é criado.
- Se um arquivo for aberto para leitura/escrita (`r+`) e existir, ele não é apagado. Se o arquivo não existir, `open` gera um erro.

Alterando um Texto

- Podemos ler todo o texto de um arquivo e fazer qualquer alteração que julgarmos necessária.
- O texto alterado pode então ser sobrescrito sobre o texto anterior.
- Como exemplo vamos fazer um programa para alterar um texto substituindo toda ocorrência da letra ’a’ por ’A’.
- Como uma string é imutável primeiro transformaremos esta em lista, alteramos o que precisar, depois transformamos a lista em string novamente para então escrever em arquivo.

Alterando um Texto

- Transformando strings em listas e vice-versa.

```
string = "abc"
string = list(string)
string
```

```
['a', 'b', 'c']
```

```
string = "".join(string)
string
```

```
'abc'
```

Alterando um Texto

- Programa que altera arquivo texto trocando ocorrências de ’a’ por ’A’.

```
try:
    arquivo = open("tarefas.txt", "r+")
    t = arquivo.read()
    t = list(t) #transformamos em lista
    for i in range(len(t)):
        if(t[i] == 'a'):
            t[i] = 'A'
    arquivo.seek(0,0)
    t = "".join(t)
    arquivo.write(t)
    arquivo.close()
except:
    print("Erro no arquivo.")
```

Parâmetros do Programa

- É possível um programa em Python receber parâmetros diretamente da linha de comando quando o programa é executado.
- Para isso devemos importar o módulo **sys** e ler os dados armazenados na lista **sys.argv**.
  - O primeiro parâmetro na lista **sys.argv** é o nome do arquivo que contém o programa.
  - Os demais parâmetros aparecem na mesma ordem em que foram digitados na linha de comando.

Parâmetros do Programa

- O programa abaixo imprime os parâmetros da linha de comando, um por linha.

```
import sys
print("Você executou o programa com",len(sys.argv),"parâmetros!")
print("Os parâmetros foram")
for p in sys.argv:
    print(p)
```

```
Você executou o programa com 3 parâmetros!
Os parâmetros foram
/home/sandra/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py
-f
/run/user/1000/jupyter/kernel-76cde6dd-a7c0-4e01-ae94-d2d420f76643.json
```

Parâmetros do Programa: Argc e Argv

- O seu uso é útil em programas onde dados de entrada são passados via linha de comando.
- Exemplo: dados a serem processados estão em um arquivo, cujo nome é passado na linha de comando.

```
import sys

# Apenas para simular o comando
sys.argv = ['programa.py', 'tarefas.txt']

if (len(sys.argv) != 2):
    print("Execute\npython programa.py nome_do_arquivo" )
else:
    try:
        arquivo = open(sys.argv[1], "r")
        while True:
            t = arquivo.readline()
            print(t, end="")
            if (t == ""):
                break
        arquivo.close()
    except:
        print("Arquivo não existe.")
```

Exemplo

- O arquivo `notas.txt` contém uma linha para cada alun* de uma turma de estudantes. O nome de cada estudante está no início da cada linha e é seguido pelas suas notas.
- Escreva um programa que imprime o nome d*s alun*s que têm mais de seis notas.

```
jose 9 4 6 8 5
pedro 5 8 3 9
suzana 8 8 7 4 3 7 4 10 9
gisela 10 8 10 5 6 10
joao 8 7 5 6 9
```

```
try:
    arquivo = open("notas.txt", "r")
    nomes = []
    linha = arquivo.readline()
    while linha:
        estudante = linha.split()
        if (len(estudante) > 7): # mais de 6 notas
            nomes.append(estudante[0]) # estudante[0] é o nome
        linha = arquivo.readline()
    arquivo.close()
    print("Estudantes: ", nomes)
except:
    print("Não foi possível abrir o arquivo.")
```

```
try:
    arquivo = open("notas.txt", "r")
    nomes = []
    # readlines lê todas as linhas do arquivo
    linhas = arquivo.readlines()
    for estudante in linhas:
        estudante = estudante.split()
        if (len(estudante) > 7): # mais de 6 notas
            nomes.append(estudante[0]) # estudante[0] é o nome
    arquivo.close()
    print("Estudantes: ", nomes)
except:
    print("Não foi possível abrir o arquivo.")
```

Exercícios

- Para o arquivo `notas.txt`, escreva um programa que calcula a média das notas de cada estudante e imprime o nome e a média de cada estudante.

- Para o arquivo `notas.txt`, escreva um programa escreva um programa que calcula a nota mínima e máxima de cada estudante e imprima o nome de cada estudante junto com a sua nota máxima e mínima.

Exercício

Faça um programa para ler os dados de um arquivo CSV (*Comma-Separated Values*) e realizar algum processamento.

Exercício

Exemplo: Faça um programa para ler os dados de ocorrência de COVID-19 no Brasil e reportar alguma estatística sobre os dados.

Arquivos CSV disponíveis em: https://covid.saude.gov.br/

Referências & Exercícios

- Os slides dessa aula foram baseados no material de MC102 do Prof. Eduardo Xavier (IC/Unicamp) e da Profa. Sandra Ávila
- https://wiki.python.org.br/ExerciciosArquivos
- https://panda.ime.usp.br/pensepy/static/pensepy/10-Arquivos/files.html