Algoritmos e Programação de Computadores

Arquivos

Ref.: material original (1o S., T. KLMN). por **Profa. Sandra Avila**, Instituto de Computação (IC/ Unicamp)

Agenda

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

Arquivos Textos

- Até o momento, foram tratados exemplos de programas que apenas obtiveram os dados de **IO** (entrada/saída) de usuários via teclado (função `input()`) ou mostrados na tela/display (função `print()`).

```
nomes = input() ;  print("valor final médio é :", x/2)
```

- Uma outra forma de ingressar ou gerar dados como entradas/saídas de programa e a través de **arquivos texto** (ou binários, ex. arquivos de audio e/ou imagens).
- Um arquivo texto utiliza caracteres que podem ser mostrados na tela ou modificados por um editor de textos simples. Exemplos (cod. ASCII): código Python, documento texto simples (plain text), páginas HTML.

Arquivos Textos

Texto

O texto constitui a forma mais utilizada de se divulgar informação em diversos meios e formatos. O texto em formato digital pode ser criado através de editores de texto, como por exemplo o Notepad do Windows, dando origens a conteúdos não formatados chamados *plain text*. Por outro lado, pode ser criado através de processadores de texto, como por exempo o Microsoft Word, dando origem a conteúdos formatados chamados *rich text*.

Principais tipos de texto:

- Plain Text
- Rich Text
- Hypertext

Formatos mais comums:

- .txt
- .doc
- .rtf
- .pdf
- .html

Arquivos Textos

- Quando comparado à entrada e saída de dados via interfaces de teclado, tela/display, as principais vantagens de manipular dados de entrada e/ou saída por meio de arquivos são:
  - Volume de dados : O conjunto de dados/informação pode ser muito maior.
  - Desempenho/Eficiência: Os dados podem ser inseridos e processados pelos programas muito mais rapidamente e com menos probabilidade de erro.
  - Repetibilidade/Compartilhamento: Os dados podem ser usados repetidamente com o mesmo programa ou com diferentes programas (local ou remotamente)

Arquivos Textos

- Um **nome** incluindo (ou não) o seu **caminho** (*path*) únicos são os parâmetros usados por usuários em programas ou *scripts* para obter acesso a um arquivo (texto ou binário), para fins de leitura e/ou modificação.
- As tarefas básicas envolvidas na manipulação de arquivos são a **leitura, escrita** ou a inclusão de dados no fim de arquivo.
- Python possue mecanismos que facilitam o gerenciamento da leitura e da escrita de dados em arquivos via diversos métodos.

Arquivos Textos

- Para trabalhar com arquivos em Python, deve-se abri-los e associá-los com uma variável.
- A variável será um objeto do tipo `file` associada a um conjunto de funções e métodos para efetivar a leitura e escrita dos arquivos.
- O primeiro passo é abrir o arquivo com a função (built-in) `open`:

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

Arquivos Textos

- Forma de abrir um arquivo texto (plain text) usando `open`:

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

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

➢ Existe também um modo "rb" para arq. binários, porém fora do escopo desta aula

Abrindo um Arquivo Texto para Leitura

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

- O primeiro parâmetro para `open` é uma string com o nome do arquivo
  - O nome pode incluir um caminho absoluto, por exemplo: `"/home/<user>/tarefas.txt"`
  - Ou um caminho relativo, como por exemplo: `"../tarefas.txt"`
- O segundo parâmetro é uma string informando o modo como o arquivo será aberto: **leitura** ou **gravação** de dados, ou **ambos**.
  - No exemplo, `"r"` significa abrir um arquivo texto para leitura.

Abrindo um Arquivo Texto para Leitura

- Ao tentar acesso (r) a um arquivo que não exista, 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:
'tarefas.txt'
```

Abrindo um Arquivo Texto para Leitura

- Caso o arquivo não exista, o erro pode ser tratado usando o recurso (*"feature"*) de tratamento de exceçoes (*exception handling)*, via os blocos dedicados `try - except` de Python.
- Todo **erro** em python gera o que é conhecido como **exceção**.
- Caso ocorra um **erro** na execução dos comandos dentro de um bloco `try`, o controle do programa é transferido de forma imediata para o bloco `except`, que é o tratamento da exceção.

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

Abrindo um Arquivo Texto para Leitura

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

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

Lendo Dados de um Arquivo Texto

- Para ler dados do arquivo aberto, utiliza-se 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, recebendo 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 (***eof****: end-of-file*).

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.
  - *Clean procedure*: considerada uma boa prática de programação

Lendo Dados de um Arquivo Texto

- O programa pode ser alterado para ler todo o arquivo de uma vez.
  - O processamento de um arquivo muito grande pode produzir uma sobrecarga da memória do computador causando impacto no desempenho (velocidade) ou mesmo uma falha/trava (*crash*).

```
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 a leitura de um byte por vez (e menos arriscada), é a leitura de uma única **linha por vez**.
- Para isso utiliza-se 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

- Ao realizar a leitura de um caractere, ou uma linha, automaticamente o indicador de posição do arquivo se movimenta 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, este pode ser fechado e aberto na sequência, ou pode ser usado o método `seek`.

Lendo Dados de um Arquivo Texto

- `seek(offset, from_what)`: o primeiro parâmetro indica quantos bytes o indicador 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, este 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

- **Sobreescreve** 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 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

- Pode-se ler todo o texto de um arquivo e fazer qualquer alteração que seja 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, altera-se o que for preciso, depois transforma-se a lista em string novamente para finalmente 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

- Alterando o arquivo texto e 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 que um programa em Python possa receber parâmetros diretamente da linha de comandos, ao executar um programa (ex. no Unix/Linux *prompt)*.
- Para tal deve-se importar o módulo **sys**, os parametros vão para uma 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 (*prompt*)
  - Ex. `Unix/Linux prompt > python3 python_program.py param1 param2`

Parâmetros do Programa

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

```
import sys

File_0 = sys.argv[1]
File_1 = sys.argv[2]
print("Você executou o programa com",len(sys.argv),"parâmetros!")
print("Parametros : ");
for i in sys.argv:
    print(i)
```

Execucao em Linux/Unix prompt >

```
python3 program_exemplo.py
arquivo_de_entrada.txt
arquivo_de_saida.txt
```

```
Você executou o programa com 3 parâmetros!
Parametros :
program_exemplo.py
arquivo_de_entrada.txt
arquivo_de_saida.txt
```

Parâmetros do Programa: Argc e Argv

- O seu uso é útil em programas onde dados de entrada e/ou saída são passados via linha de comando (ex. `Linux *prompt*`).
- Exemplo: dados a serem processados que estejam ou desejem ser gerados em um ou mais arquivos, cujos nomes são passados ao programa Python no momento de invocar o arquivo .py, diretamente na linha de comandos.
- `Linux/Unix prompt >`

```
python3 program_exemplo.py arquivo_de_entrada.txt arquivo_de_saida.txt
```

```
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 aluno de uma turma de estudantes. O nome de cada estudante está no início da cada linha e é seguido pelas suas notas.
- Escrever um programa que imprime o nome dos alunos 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.

Referências & Exercícios

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