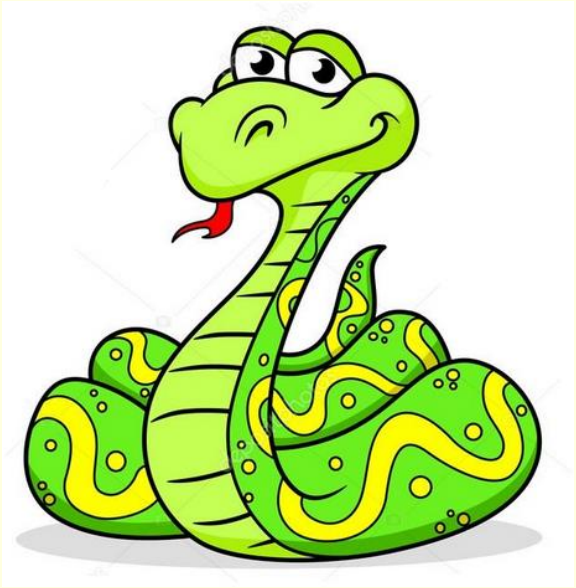
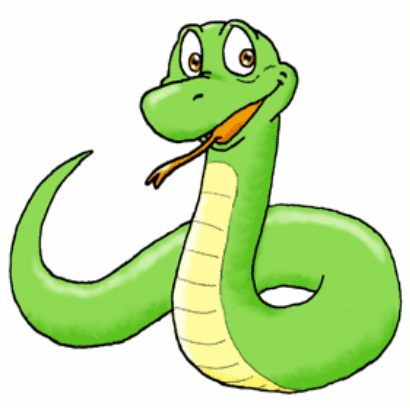
Python: Arquivos



Prof. Dr. Dilermando Piva Jr.

Arquivos

- Entrada e saída são operações de comunicação de um programa com o mundo externo
- Essa comunicação se dá usualmente através de *arquivos*
- Arquivos estão associados a dispositivos
  - Por exemplo, disco, impressora, teclado
- Em Python, um arquivo pode ser lido/escrito através de um objeto da classe file



Arquivos default

- Já usamos, sem saber, três arquivos *default*
  - Sempre que um comando print é executado, o resultado vai para um arquivo chamado **sys.stdout**
  - Sempre que lemos um dado através do comando input, na verdade estamos lendo de um arquivo chamado **sys.stdin**
  - Mensagens de erro ou de rastreamento de exceções são enviadas para um arquivo chamado **sys.stderr**

Exemplo

```
>>> import sys
>>> sys.stdout.write("alo")
alo
>>> print ("alo“)
alo
>>> sys.stdin.readline()
sfadfas
'sfadfas\n'
>>> input()
fasdfadsf
'fasdfadsf'
```


Redirecionamento

- Os arquivos sys.stdin, sys.stdout e sys.stderr normalmente estão associados ao teclado e ao display do terminal sendo usado, mas podem ser reassociados a outros dispositivos
  - Em *Unix/Linux* e *Windows*:
    - programa > arquivo
      - Executa programa redirecionando stdout para arquivo
    - programa < arquivo
      - Executa programa redirecionando stdin de arquivo
    - programa1 | programa2
      - Executa programa1 e programa2 sendo que a saída de programa1 é redirecionanda para a entrada de programa2

Abrindo arquivos

- **open (*name*, *mode*, *buffering*)**
  - *name* : nome do arquivo a abrir
  - *mode* : (opcional) modo de abertura – string contendo
    - r : leitura (default)
    - w : escrita
    - x : abre para escrita, somente se o arquivo não existir. Se existir → Erro
    - b : binário
    - a : escrita, adicionando o conteudo ao final do arquivo. Caso o arquivo não exista, será criado um novo.
    - + : (usado com r) indica leitura e escrita

Abrindo arquivos

- *buffering* : (opcional) indica se memória (*buffers*) é usada para acelerar operações de entrada e saída
  - 0 : buffers não são usados
  - 1 (ou qq número negativo): um buffer de tamanho padrão (default)
  - 2 ou maior: tamanho do buffer em bytes

Métodos *Read*, *Write* e *Close*

- read(*num*)
  - Lê *num* bytes do arquivo e os retorna numa string
  - Se *num* não é especificado, todos os bytes desde o ponto atual até o fim do arquivo são rretornados
- write(*string*)
  - Escreve *string* no arquivo
  - Devido ao uso de buffers, a escrita pode não ser feita imediatamente
    - Use o método flush() ou close() para assegurar a escrita física
- close()
  - Termina o uso do arquivo para operações de leitura e escrita

O objeto *file*

- O comando open retorna um objeto do tipo *file* (arquivo)
  - Na verdade, em Python 2.4 em diante, open é o mesmo que file, e portanto o comando é um construtor
- O objeto retornado é usado subseqüentemente para realizar operações de entrada e saída:

```
>>> arq = open ("teste", "w")
>>> arq.write ("Oi")
>>> arq.close ()
>>> arq = open ("teste")
>>> x = arq.read()
>>> x
'Oi'
```

O objeto *file*

- Leitura de Arquivos...

```
arquivo = open('texto.txt')
# Para ler o conteúdo de um arquivo, após sua abertura, devemos utilizar a função read()
ret = arquivo.read()
print(type(ret))
print(ret)
# OBS: O Python, utiliza um recurso para trabalhar com arquivos chamado cursor. Esse cursor, funciona como o cursor quando estamos escrevendo.
# OBS: A função read() lê todo o conteúdo do arquivo.
```

O objeto *file*

- Escrita em Arquivos
  - Ao abrir um arquivo para leitura, não podemos realizar a escrita nele. Apenas ler. Da mesma forma, se abrirmor um arquivo para escrita, não podemos lê-lo, somente escrever nele.
  - Ao abrir um arquivo para escrita, o arquivo é criado no sistema operacional.
  - Para escrevermos dados em um arquivo, após abrir o arquivo utilizamos a função write().
  - Esta função recebe uma string como parâmetro, caso contrário teremos um TypeError
  - Abrindo um arquivo para escrita com o modo 'w', se o arquivo não existir será criado, caso ele já exista, o anterior será apagado e um novo será criado. Dessa forma, todo o conteúdo no arquivo anterior é perdido.

O objeto *file*

Escrita em Arquivos

```
# Exemplo de escrita - modo 'w' - write (escrita)
# Forma tradicional de escrita em arquivo (Não Pythônica)
arquivo = open('mais.txt', 'w')
arquivo.write('Um texto qualquer.\n')
arquivo.write('Mais um texto.')
arquivo.close()

# Forma Pythônica
with open('novo.txt', 'w') as arquivo:
    arquivo.write('Novos dados.\n')
    arquivo.write('Outros podemos colocar quantas linhas quisermos.\n')
    arquivo.write('Mais Esta é a última linha.')

with open(‘teste1.txt', 'w') as arquivo:
    arquivo.write(‘Piva ' * 1000)
```


O objeto *file*

- Escrita em Arquivos

```
# Forma Pythônica


with open('frutas.txt', 'w') as arquivo:
    while True:
        fruta = input('Informe uma fruta ou digite sair: ')
        if fruta != 'sair':
            arquivo.write(fruta)
            arquivo.write('\n')
        else:
            break
```

Outros Exemplos...

Exemplo x

```
try:
    with open(‘piva.txt', 'x') as arquivo:
        arquivo.write('Teste de conteúdo.\n')
except FileExistsError:
    print('Arquivo já existe')
```


Outros Exemplos...

Exemplo a

```
with open('frutas.txt', 'a') as arquivo:
    while True:
        fruta = input('Informe uma fruta ou sair: ')
        if fruta != 'sair':
            arquivo.write(fruta)
            arquivo.write('\n')
        else:
            break
```

Outros Exemplos...

Exemplo r+

```
with open('outro.txt', 'r+') as arquivo:
    arquivo.write('Adicionada\n')
    arquivo.seek(11)
    arquivo.write('Nova linha.\n')
    arquivo.seek(32)
    arquivo.write('Mais uma linha.\n')
```


Convenção de fim de linha

- Arquivos de texto são divididos em linhas usando caracteres especiais
  - Linux/Unix: \n
  - Windows: \r\n
  - Mac: \r
- Python usa sempre \n para separar linhas
  - Ao se ler/escrever um arquivo aberto em modo texto (não binário) faz traduções de \n para se adequar ao sistema operacional
  - Em modo binário, entretanto, a conversão não é feita

Interação com o Sistema Operacional

- Operações de entrada e saída são na verdade realizadas pelo sistema operacional
- O módulo os possui diversas variáveis e funções que ajudam um programa Python a se adequar ao sistema operacional, por exemplo:
  - os.getcwd() retorna o diretório corrente
  - os.chdir(*dir*) muda o diretório corrente para *dir*
  - os.sep é uma string com o caractere que separa componentes de um caminho ('/' para *Unix*, '\\' para *Windows*)
  - os.path.exists(*path*) diz se *path* se refere ao nome de um arquivo existente

Lendo e escrevendo linhas

- readline(*n*)
  - Se *n* não é especificado, retorna exatamente uma linha lida do arquivo
  - Caso contrário, lê uma linha, mas busca no máximo *n* caracteres pelo final de linha
- readlines(*n*)
  - Se *n* não é especificado, retorna o restante do conteúdo do arquivo em uma lista de strings
  - Caso *n* seja especificado, a leitura é limitada a *n* caracteres no máximo

Lendo e escrevendo linhas

- writelines(*seqüência*)
  - Escreve a lista (ou qualquer seqüência) de strings, uma por uma no arquivo
  - Caracteres terminadores de linha ***não são*** acrescentados

Acesso direto

- É possível ler e escrever não seqüencialmente em alguns tipos de arquivo
  - Devem estar associados a dispositivos que permitem acesso direto, como discos, por exemplo

- seek(*offset,whence*)
  - *offset* indica o número do byte a ser lido e escrito pela próxima operação de entrada e saída
  - *whence* indica a partir de onde *offset* será contado
    - 0 (default) : do início
    - 1 : do ponto corrente
    - 2 : do final