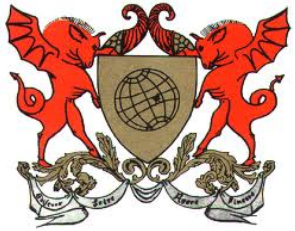
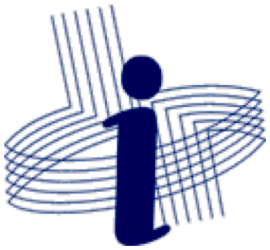


Universidade Federal de Viçosa
Departamento de Informática
Centro de Ciências Exatas e Tecnológicas



INF101 – Introdução à Programação II

Arquivos

Definição

- Arquivo é uma estrutura de dados que reside em memória secundária (discos, *pendrives*, fitas etc.)
- Somente uma parte de um arquivo é que fica na memória primária; esta parte é denominada *buffer*
- As operações de leitura e escrita são normalmente realizadas nesse *buffer*
- Existem outras operações com arquivos além das de entrada e saída


Operações com Arquivos

- Abertura
  - Para acessar qualquer arquivo, é necessário abri-lo de antemão
- Entrada (leitura)
  - Depois de aberto, um arquivo que já exista, pode ser lido usando operações de entrada
- Saída (escrita)
  - Um novo arquivo é criado com operações de escrita
- Fechamento
  - Depois de usado, um arquivo deve ser fechado para liberar os recursos que o sistema operacional alocou para o arquivo; além disso, esta operação garante a integridade dos dados armazenados no arquivo


Abertura de Arquivo

- Em Python, abrimos um arquivo com a função open
- Esta função necessita de dois parâmetros: o nome externo do arquivo e o modo de abertura
- O nome externo é o nome por meio de que o sistema operacional conhece o arquivo
- Por outro lado, o programa em Python conhece o arquivo por meio de uma variável
- É por meio desta variável que todas as operações são realizadas

Abertura de Arquivo

- Modos de abertura de arquivo

Modo	Operações
`r`	leitura (é o *default*)
`w`	escrita (apaga o que já existir)
`a`	escrita (preserva o que já existir)
`b`	modo binário
`t`	modo texto (é o *default*)
`+`	atualização (leitura e escrita)


Abertura de arquivos

- Os modos podem ser combinados:
  - `"r+"`, `"w+"`, `"a+"`, `"r+b"`, `"wb"`, `"ab"`
  - Observe que o modo `+` tem de ser usado conjuntamente com `r`, `w` ou `a`
- A função `open` retorna um objeto do tipo arquivo (`file`, em Python)
- É esse objeto que utilizamos para ler ou escrever dados no arquivo
- Utilizamos o método `write` para escrever ou gravar dados no arquivo
- E o método `read` para ler dados do arquivo


Operações com Arquivos

- A abertura realiza a ligação entre o programa e o espaço em disco gerenciado pelo sistema operacional
- As operações de leitura e/ou escrita são o que realmente desejamos realizar no programa
- O fechamento informa ao sistema operacional que não vamos mais trabalhar com o arquivo
- Assim o fechamento é importante para liberar os recursos que foram alocados para o nosso arquivo


Exemplo

```
arq = open("resultados.txt", "w")
for linha in range(1, 101):
    arq.write("%d\n" % linha)
arq.close()
```


Outro Exemplo

```
arq = open("dados.txt", "r")
for linha in arq.readlines():
    print(linha, end="")
arq.close()
```


Operações de Entrada

- O Python fornece três operadores para arquivos de texto:
  - `read()` que lê todo o arquivo de entrada para o buffer de uma só vez, portanto só funciona para arquivos não muito grandes; o resultado da leitura é um imenso *string* sem nenhuma estruturação
  - `readlines()` que também lê todo o arquivo de uma só vez para o *buffer* e o estrutura na forma de uma lista de linhas; cada linha mantém o caractere `'\n'` no final
  - `readline()` que lê apenas uma linha de cada vez para o *buffer* mantendo o caractere `'\n'` no final da linha; funciona para qualquer tamanho de arquivo

Exemplo de Leitura com readline

- Lê um arquivo linha por linha; cada linha contém dois dados separados por um espaço em branco: um número inteiro e outro *float*; depois insere os dados em uma lista de 2-tuplas

```
arq = open('dados.txt', 'r')
lista = []
linha = arq.readline().rstrip('\n')
while linha != '':
    dados = linha.split(' ')
    v0 = int(dados[0])
    v1 = float(dados[1])
    lista.append((v0, v1))
    linha = arq.readline().rstrip('\n')
arq.close()
```


Parâmetros da Linha de Comando

- Podemos acessar os parâmetros passados ao programa na linha de comando
- Para tanto, podemos usar o módulo `sys` e a lista `argv` definida nesse módulo
- Exemplo

```
import sys
print("Nº de parâmetros: %d" % len(sys.argv))
for n, p in enumerate(sys.argv):
    print("Parâmetro %d = %s" % (n, p))
```


Parâmetros da Linha de Comando

- Experimente chamar o programa dado na transparência anterior das seguintes formas:

```
python3 comline.py prim seg ter
python3 comline.py 1 2 3
python3 comline.py readme.txt 5
```

- Observe que `comline.py` é o nome que demos ao código fonte e o que segue são os parâmetros passados pela linha de comando


Parâmetros da Linha de Comando

- Observe que cada parâmetro foi passado como um elemento da lista `sys.argv`
- Observe também que os parâmetros são separados por espaços em branco na linha de comando
- E se precisarmos passar algum parâmetro que tenha espaços em branco em si?
- Basta escrever o parâmetro entre aspas; por ex.:

```
python3 comline.py "Ana Lobo" INF101 FIS201
```


Exercício

- Escreva um programa que processe um arquivo de entrada com as seguintes regras:
  - O arquivo consiste em linhas
  - Cada linha no seu início contém ou não um caractere de controle de processamento:
    - ; significa que a linha é para ser ignorada
    - > significa que a linha deve ser impressa alinhada à direita
    - < significa que a linha deve ser impressa alinhada à esquerda
    - * significa que a linha deve ser centralizada
    - Nada significa que a linha não sofrerá nenhum processamento; deixe-a como está


Uma Solução

```
largura = 79
entrada = open("entrada.txt")
for linha in entrada.readlines():
    if linha[0] == ';':
        continue
    elif linha[0] == '>':
        print(linha[1:].rjust(largura))
    elif linha[0] == '<':
        print(linha[1:].lstrip().ljust(largura))
    elif linha[0] == '*':
        print(linha[1:].center(largura))
    else:
        print(linha)
entrada.close()
```


Arquivos e Diretórios

- Quem usa computador pessoal sabe que, de vez em quando, é necessário navegar pelo sistema de arquivos da máquina
- Em geral, os sistemas operacionais modernos fornecem algum utilitário de navegação
- Como podemos fazer isso de dentro de um programa em Python?
- Alguns sistemas operacionais chamam os *diretórios* de *pastas*


Obtenção do Diretório Atual

```
>>> import os
>>> os.getcwd()
/home/alunos
```

Troca de Diretório

- Antes, vamos criar alguns diretórios na linha de comando de algum emulador de terminal:

```
$ mkdir a
$ mkdir b
$ mkdir c
```

- Se preferir, você pode criar os diretórios `a`, `b` e `c` usando o navegador de arquivo de seu sistema operacional


Troca de Diretório

```
import os
os.chdir("a")
print(os.getcwd())
os.chdir("..")
print(os.getcwd())
os.chdir("b")
print(os.getcwd())
os.chdir("../c")
print(os.getcwd())
```


Abreviaturas de Diretórios

- Podemos referenciar os diretórios por abreviaturas:
  - Diretório atual

    `.`
  - Diretório pai

    `..`
  - Diretório *home*

    `os.path.expanduser('~')`
  - Diretório raiz

    /

    (em Windows, é \)


Caminhos

- Os diretórios ou arquivos podem ser denotados pelo:
  - Caminho relativo (começa do diretório atual)

    `a/dados.txt`
  - Caminho absoluto (começa do diretório raiz)

    `/home/alunos/a/dados.txt`


Criação de Diretório e Navegação

- As funções `mkdir` e `getcwd` são fundamentais:

```
import os
os.mkdir("d")
os.mkdir("e")
os.mkdir("f")
print(os.getcwd())
os.chdir("d")
print(os.getcwd())
os.chdir("../e")
print(os.getcwd())
os.chdir("..")
print(os.getcwd())
os.chdir("f")
print(os.getcwd())
```


Criação de Subdiretórios com um Comando Apenas

- Para criar uma estrutura de subdiretórios de uma vez só, use a função `makedirs`:

```
import os
os.makedirs("g/h/i")
os.makedirs("g/j/k")
```


Alteração de Nomes de Arquivos e Diretórios

- Para renomear arquivos ou diretórios, use a função `rename`:

```
import os
os.mkdir("velho")
os.rename("velho", "novo")
```


Alteração de Nomes de Arquivos e Diretórios

- A função `rename` também pode ser usada para mover arquivos ou diretórios:

```
import os
os.makedirs("a/b/c")
os.makedirs("d/e/f")
os.rename("a/b/c", "d/e/c")
```


Remoção de Arquivos e Diretórios

- Para remover um diretório, use a função `rmdir`
- Para remover um arquivo, use a função `remove`:

```
import os
# Cria um arquivo e fecha-o imediatamente
open("teste.txt", "w").close()
os.mkdir("efemero")
os.rmdir("efemero")
os.remove("teste.txt")
```


Listagem de Diretórios

- Podemos listar os nomes do que um diretório contém com a função `listdir`:

```
import os
print(os.listdir("."))
print(os.listdir("a"))
print(os.listdir("a/b"))
print(os.listdir("d/e"))
```


Verificação Se É Diretório ou Arquivo

- O módulo `os.path` traz várias outras funções para obter informações sobre os diretórios e arquivos em disco
- As primeiras que vamos considerar são os predicados `isdir` e `isfile`:

```
import os
import os.path
for a in os.listdir("."):
    if os.path.isdir(a):
        print("%s/" % a)
    elif os.path.isfile(a):
        print("%s" % a)
```


Verificação Se um Diretório ou Arquivo Já Existe

- Podemos verificar a existência de arquivos ou diretórios com a função `exists`:

```
import os.path
if os.path.exists("d"):
   print("O diretório d existe.")
else:
   print("O diretório d não existe.")
```

Exercício

- Modifique o programa da transparência 28 de forma a receber o nome do arquivo ou diretório a ser verificado pela linha de comando. Além disso, caso exista, diga se é arquivo ou se é diretório


Obtenção de Mais Informações sobre um Arquivo

- Existem outras funções que retornam mais informações sobre arquivos ou diretórios como, por exemplo, o tamanho, datas de modificação, criação e acesso
- `getsize` retorna o tamanho do arquivo em bytes
- `getctime` retorna a data e a hora de criação do arquivo
- `getmtime` retorna a data e a hora da última modificação do arquivo
- `getatime` retorna a data e a hora do último acesso ao arquivo


Exemplo

```
import os
import time
import sys
nome = sys.argv[1]
print("Nome: %s" % nome)
print("Tamanho: %d bytes" % os.path.getsize(nome))
print("Criado em: %s" % time.ctime(os.path.getctime(nome)))
print("Modificado em: %s" % time.ctime(os.path.getmtime(nome)))
print("Acessado em: %s" % time.ctime(os.path.getatime(nome)))
```


Uso de Caminhos

- Quando se trabalha com arquivos, é comum ter que manipular caminhos dentro do sistema de arquivos para se localizar o arquivo desejado
- Já vimos na transparência 20 que existem o caminho relativo e o absoluto
- Nos sistemas Linux e macOS, o separador de diretórios é a barra /
- No sistema Windows, é a contrabarra \


Uso de Caminhos (Exemplo)

```
>>> import os.path
>>> caminho = "d/e/f"
>>> os.path.abspath(caminho)
'/Users/lcaalbuquerque/d/e/f'
>>> os.path.basename(caminho)
'f'
>>> os.path.dirname(caminho)
'd/e'
>>> os.path.split(caminho)
('d/e', 'f')
>>> os.path.splitext("info.py")
('info', '.py')
>>> os.path.splitdrive("c:/Windows")
('c:', '/Windows')
```


Combinação dos Componentes de um Caminho

- A função `join` junta os componentes de um caminho, separando-os com barras, se necessário:

```
>>> import os.path
>>> os.path.join("c:", "inf101", "programas")
'c:inf101\\programas'
>>> os.path.abspath(os.path.join("c:", "inf101",
                         "programas"))
'C:\\Python35\\inf101\\programas'
```


Visita a Todos os Subdiretórios Recursivamente

- A função `os.walk` facilita a navegação em uma hierarquia (árvore) de diretórios
- Imagine que queiramos percorrer todos os diretórios a partir de um diretório inicial, retornando o nome do diretório (`raiz`), os subdiretórios encontrados dentro do diretório visitado (`diretorios`) e todos os arquivos (`arquivos`)


Visita a Todos os Subdiretórios Recursivamente (Exemplo)

```
import os
import sys
for raiz, diretorios, arquivos in os.walk(sys.argv[1]):
    print("\nCaminho:", raiz)
    for d in diretorios:
        print("  %s/" % d)
    for f in arquivos:
        print("  %s" % f)
    print("%d diretório(s), %d arquivo(s)" %
          (len(diretorios), len(arquivos)))
```