Computação II

MAB 225 - EPT/EP1

Manipulação de Arquivos

Brunno Goldstein

bfgoldstein@cos.ufrj.br

www.cos.ufrj.br/~bfgoldstein

Ementa

- Programação Orientada a Objetos
- Tratamento de Exceções
- Módulos
- Manipulação de Arquivos
- Interface Gráfica (Tkinter)
- Biblioteca Numérica (Numpy)

Ementa

- Programação Orientada a Objetos
- Tratamento de Exceções
- Módulos
- Manipulação de Arquivos
- Interface Gráfica (Tkinter)
- Biblioteca Numérica (Numpy)

Entrada e Saída

- Operações que realizam comunicação com "o mundo exterior" ao seu programa;

- Interfaces de comunicação:
  - Monitor;
  - Teclado;
  - HDs;
  - Interface de rede (wifi, cabo, bluetooth, etc)
  - ...

Arquivos

- São estruturas que armazenam dados;
- Arquivos são persistidos:
  - Seus dados são mantidos para uso futuro;
- Tal persistência é feita, normalmente, em disco rígido.
- Em Python, arquivos podem ser lidos/escritos através de objetos da classe **File**;

Arquivos

- Três comandos, já conhecidos, que usam Arquivos:
  - PRINT
    - Dado escrito no comando vai para o arquivo sys.stdout
  - INPUT / RAW_INPUT
    - Dado lido por estes comandos vem do arquivo sys.stdin
  - Mensagem de erro
    - Mensagens de erro ou de exceções vão para o arquivo sys.stderr

Arquivos - Exemplo

```
>>> import sys
>>> sys.stdout.write("Teste!")
>>> Teste!
>>> print "Teste!"
>>> Teste!
>>> sys.stdin.readline()
Oi!
'Oi!\n'
>>> raw_input()
Oi!
'Oi!'
```

Arquivos

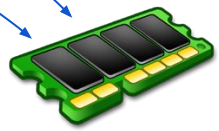
```
class Pessoa(object):

    def __init__(self, nome, tipo, endereco):
        self.nome = nome
        self.tipo = tipo
        self.endereco = endereco

    def resumo(self):
        print 'Nome: ' + self.nome
        print 'Tipo: ' + self.tipo
        print 'Endereco: ' + self.endereco
```

pessoa_class.py

```
>>> from pessoa import Pessoa
>>> obj_1 = Pessoa("Pessoa_1", "1", "Fundao")
>>> obj_1.nome
'Pessoa_1'
>>> obj_1.endereco
'Fundao'
>>> quit()
$
```



Memória RAM

Dados não foram persistidos e são "perdidos / liberados" da memória ram.

Manipulando Arquivos

- Alguns comandos *built-in* para arquivos:

- Abertura e fechamento:
  - open
  - close

- Manipulação dos dados:
  - write
  - read
  - readline
  - readlines
  - writelines
  - seek
  - tell

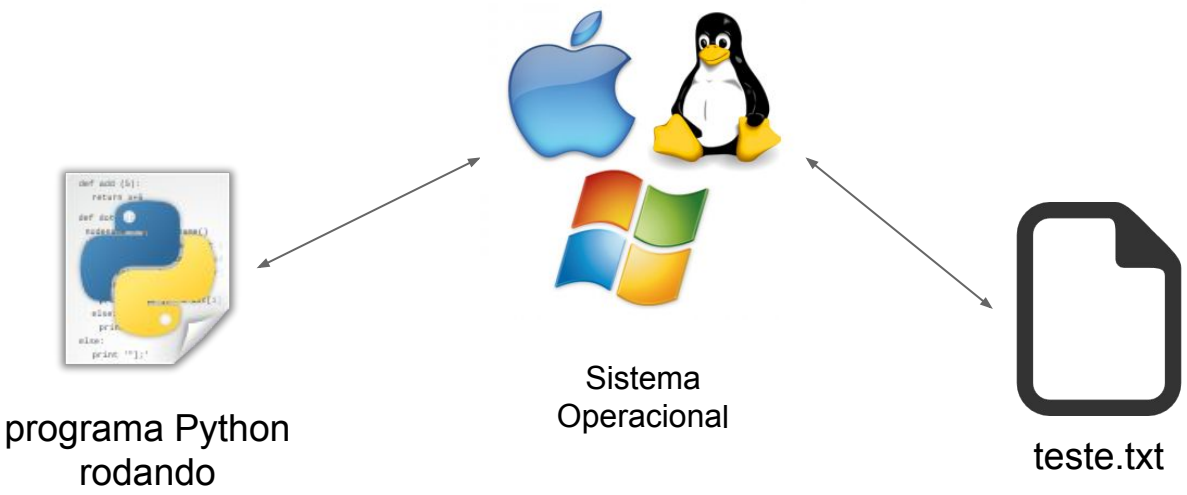
Manipulando Arquivos

➔ **open**(name[, *mode*[, *buffering*]])

- Retorna um objeto do tipo **File**:
  - Parâmetros
    - name: String com o nome do arquivo;
    - mode: String com o modo de abertura do arquivo
      - 'r': Apenas para leitura; (default)
      - 'w': Escrita;
      - 'a': Concatenação com arquivo existente;
      - 'b': Arquivo binário.
    - buffering: Tamanho (bytes) do buffer para armazenar em memória

Manipulando Arquivos - Exemplo

```
>>> arq = open ("teste.txt", "w")
```



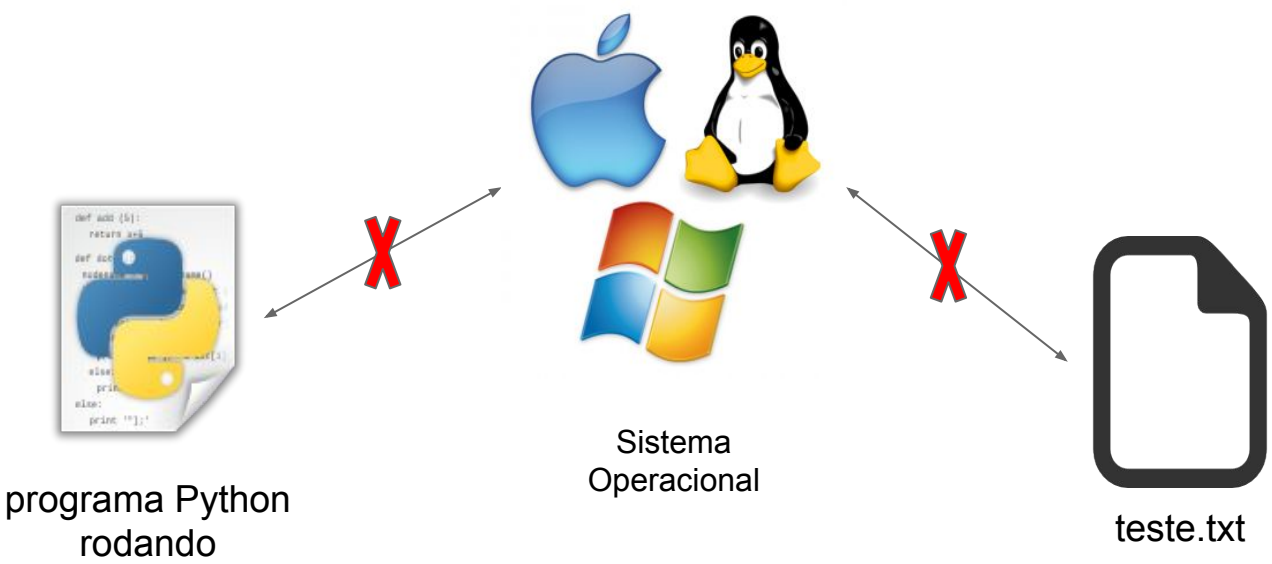
Manipulando Arquivos

➔ **obj.close()**

- Indica ao SO que a manipulação do objeto **File** terminou;
- Todos recursos que o SO usava para manipular o arquivo são liberados;
- Método não possui parâmetros.

Manipulando Arquivos - Exemplo

```
>>> arq = open ("teste", "w")
>>> arq.close ()
```



Manipulando Arquivos

- read(*num*)
  - Método que lê *num* bytes do arquivo e retorna a string correspondente;
  - *num* é opcional. Se omitido, todos os bytes são lidos desde o ponto atual até o fim do arquivo.

- write(*string*)
  - Método que escreve uma *string* no arquivo;
  - A escrita é bufferizada, ou seja, não é feita automaticamente no arquivo;
  - Primeiro se escreve no buffer da memória principal e, em algum momento, o SO escreve no arquivo;
  - Para garantir que essa escrita seja feita no momento determinado, o método *flush()* pode ser usado;
  - O método *close()* também irá garantir que tudo foi escrito até o "fechamento" do arquivo.

Manipulando Arquivos - Exemplo

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

Manipulando Arquivos

- readline(*n*)
  - Método que lê uma linha do arquivo e armazena em uma string;
  - *Se n* não for omitido, lê uma linha de até no máximo *n* caracteres do arquivo;
  - Se *n* for omitido, retorna exatamente uma linha lida.

- readlines(*n*)
  - Método que lê todas as linhas do arquivo e armazena em uma **lista** de strings;
  - Se *n* for especificado, a leitura é limitada a *n* bytes no máximo.

Manipulando Arquivos

- writelines(*n*)
  - Método que escreve uma lista (ou qualquer sequência) de strings, no arquivo;
  - Caracteres terminadores de linha **não são** acrescentados.

Manipulando Arquivos - Fim de linha

- Existe uma convenção para informar o fim de uma linha em arquivos;

- Linhas são separadas por caracteres especiais da seguinte forma:
  - Linux/Unix: **\n**
  - Windows: **\r\n**
  - Mac: **\r**

- Python usa sempre **\n** para separar as linhas!!
  - Arquivos em modo texto (sem usar o 'b' de binário) devem possuir o caractere \n no final das linhas;

Interação com o Sistema Operacional

- Toda operação de entrada e saída é uma "chamada" ao sistema operacional;
  - O programa em Python pede ao SO para realizar uma determinada função de I/O

- Algumas funções que facilitam a interação com o SO são fornecidas pelo módulo **OS**:

  - os.getcwd()
    - Retorna o diretório corrente, onde o programa Python está executando;

  - os.chdir(*dir*)
    - Muda o diretório corrente de execução do programa Python;

  - os.sep
    - retorna uma string com o separador usado pelo SO
    - ' / ' para Unix (Linux & Mac)
    - ' \\ ' para Windows

  - os.path.exists(*path*)
    - retorna **True** caso o *path* exista e **False** caso contrário

Interação com o Sistema Operacional

```
>>> import os
>>> os.getcwd()
'/Users/bfgoldstein'
>>> os.chdir('/Users/bfgoldstein/Documents')
>>> os.getcwd()
'/Users/bfgoldstein/Documents'
>>> os.sep
'/'
>>> os.path.exists('/Users/bfgoldstein/Documents')
True
```

Persistindo Objetos em Arquivos

- Objetos também podem ser persistidos (salvos) em arquivos;
- Para tal, primeiro é necessário "serializar" esse objeto;
- Serialização é transformar um objeto em uma cadeia de caracteres;
- Deserialização é o inverso de serialização;
- Python possui um módulo chamado Pickle que faz todo o trabalho de serialização e deserialização do objeto;

Persistindo Objetos em Arquivos

➢ O módulo pickle nos fornece duas funções principais para serializar e salvar o objeto, bem como a deserialização e leitura do objeto:

- pickle.dump(obj, fileObj)
  - Método dump serializa e salva o objeto (obj) no arquivo;
  - Parâmetros
    - obj: Objeto a ser salvo no arquivo;
    - fileObj: Objeto do tipo File retornado pelo método open().

- pickle.load(fileObj)
  - Método load que retorna um objeto salvo no arquivo fileObj;
  - Parâmetros
    - fileObj: Objeto do tipo File retornado pelo método open().

Persistindo Objetos em Arquivos

```
>>> import pickle
>>> a = ['test value','test value 2','test value 3']
>>> a
['test value','test value 2','test value 3']
>>> fileObject = open("testfile",'wb')
>>> pickle.dump(a,fileObject)
>>> fileObject.close()
>>> fileObject = open("testfile",'rb')
>>> b = pickle.load(fileObject)
>>> b
['test value','test value 2','test value 3']
```

Exercícios

1. Crie um programa que irá abrir um arquivo, utilizando modo escrita, e salvar a string "Olá!" no mesmo.

2. Crie um programa que irá abrir o arquivo anterior (em modo leitura) e irá ler todo seu conteúdo. Salve este conteúdo em uma variável e exiba na tela.

3. Crie um programa que irá criar uma lista de frases digitadas pelo usuário. Tal programa deverá salvar esta lista em arquivo. Obs.: Utilize os caracteres de fim de linha.

4. Crie a classe Pessoa com os métodos e atributos que desejar. No python shell, crie um objeto dessa classe e salve-o em arquivo utilizando o módulo pickle.

   Verifique se salvou corretamente carregando tal objeto no arquivo em uma variável.

Referências

- Python Documentation: https://docs.python.org/
- Python: Entrada e Saída - Claudio Esperança