Laboratório de Programação 1

Aula 12

Mário Hozano
professor@hozano.com

Ciência da Computação
UFAL - Arapiraca

Relembrando a aula anterior...

- O que são dicionários?
- Como acessar os items de um dicionário?
- Os dicionários são mutáveis ou imutáveis?
- O que são chaves em dicionários?
- As chaves dos dicionários devem ser de que tipo?
- E os valores ?

Roteiro da aula

- Arquivos
- Abrindo Arquivos
- Escrevendo em Arquivos
- Leitura em Arquivos
- O Módulo *pickle*
- Exceções

Arquivos

- Durante a execução de um programa, seus dados ficam na memória RAM
- Quando o computador é desligado (ou o programa termina), seus dados são perdidos
- Para armazenar dados permanentemente, deve-se utilizar **arquivos**
- Arquivos são organizados em diretórios (pastas) e armazenados no HD, CD-ROM, Flash Cards etc.

Arquivos

- Trabalhar com arquivos em programação é como trabalhar com livros no dia-a-dia
- Para ler ou escrever em um livro, é necessário abri-lo
- Enquanto o livro estiver aberto você pode ler ou escrever nele
- Em qualquer caso você sabe onde você está situado no livro (página, parágrafo, linha)
- Após fazer as leituras e escritas, o livro deve ser fechado

Arquivos

- Trabalhar com arquivos em programação é como trabalhar com livros no dia-a-dia
- Para ler ou escrever em um livro, é necessário **abri-lo**
- Enquanto o livro estiver aberto você pode **ler** ou **escrever** nele
- Em qualquer caso você sabe onde você está **situado** no livro (página, parágrafo, linha)
- Após fazer as leituras e escritas, o livro deve ser **fechado**

Abrindo Arquivos

- Para abrir um arquivo deve-se utilizar a função *open()*
- Esta função recebe como parâmetros o nome e o modo da “abertura” do arquivo

```
>>> arquivo = open('teste.dat', 'w')
>>> type(arquivo)
<type 'file'>
```

- Podemos indicar o caminho completo do arquivo (Ex.: '/home/alunos/teste.dat')
- Caso o caminho não seja indicado, será considerado o diretório corrente (ou do script em execução)

Abrindo Arquivos

- As operações sobre os arquivos devem obedecer o modo de abertura do mesmo.
- Leitura e escrita só são permitidas dependendo do modo
- Os modos de abertura de arquivos são:

Modo	Descrição
w	Modo de escrita (*write*). Um novo arquivo é criado. Se o arquivo já existir, ele é sobrescrito.
r	Modo somente leitura (*read*).
a	Modo de escrita em anexo (*append*). As operações de escrita são realizadas no final do arquivo.
r+	Modo de leitura e escrita. As operações de escrita são feitas a partir da última linha lida ou do início.

Escrevendo e Fechando Arquivos

- Depois de aberto em um modo de escrita, podemos inserir dados no arquivo com o método *write()*

```
>>> arquivo = open('teste.dat', 'w')
>>> arquivo.write('primeira escrita')
>>> arquivo.write('segunda escrita')
>>> arquivo.close()
```

- Depois de inserir dados é necessário fechar o arquivo com o método *close()*

- A chamada do método close() deve ser realizada mesmo após operações de leitura em arquivos

Lendo Arquivos

- É possível ver o conteúdo de arquivos após a abertura do mesmo com um modo compatível
- O método *read()* permite a leitura completa do arquivo aberto, como no arquivo que criamos anteriormente

```
>>> arquivo = open('teste.dat', 'r')
>>> print arquivo.read()
'primeira escritasegunda escrita'
>>> arquivo.close()
```

Lendo Arquivos

- É possível ver o conteúdo de arquivos após a abertura do mesmo com um modo compatível
- O método *read()* permite a leitura completa do arquivo aberto, como no arquivo que criamos anteriormente

```
>>> arquivo = open('teste.dat', 'r')
>>> print arquivo.read()
'primeira escritasegunda escrita'
>>> arquivo.close()
```

Por que a leitura retornou a palavra 'escritasegunda' ?

Escrevendo e Lendo linhas em Arquivos

- Uma escrita em arquivo inicia do ponto em que a última escrita terminou ou do início do arquivo

- Para indicar uma quebra de linha em um arquivo, deve-se utilizar o marcador **'\n'**

```
>>> arquivo = open('teste.dat', 'w')
>>> arquivo.write('primeira escrita \n')
>>> arquivo.write('segunda escrita \n')
>>> arquivo.close()
>>> arquivo = open('teste.dat', 'r')
>>> print(arquivo.read())
primeira escrita
segunda escrita
```

Leitura por linha

- Como vimos a escrita em arquivo inicia após o último caractere inserido na última escrita (apontador)
- A leitura de arquivos também é sequencial, e utiliza-se de um apontador
- Caso você leia um arquivo inteiro e chame o método *read()* novamente, será retornada uma *string* vazia

```
>>> arquivo = open('teste.dat', 'r')
>>> print(arquivo.read())
primeira escrita
segunda escrita
>>> print(arquivo.read())
```

Leitura por linha

- O método *readline()* permite a leitura por linha, evitando a leitura completa do arquivo

```
>>> arquivo = open('teste.dat', 'r')
>>> print(arquivo.readline())
primeira escrita
>>> print(arquivo.readline())
segunda escrita
```

- É possível também ler um número específico de bytes no arquivo indicando o número de bytes no método *read()*

```
>>> arquivo = open('teste.dat', 'r')
>>> print(arquivo.read(5))
prime
>>> print(arquivo.read(11))
ira escrita
```

Relocando o apontador

- Uma vez lido o arquivo inteiro (ou parte dele) não se pode ler novamente uma linha que esteja antes do apontador
- Nestes casos deve-se utilizar o método *seek()* para colocar o apontador em um novo ponto de leitura
- O método *seek()* recebe como argumento um número de bytes que ele tem que percorrer do início do arquivo para o ponto em que ele será relocado

```
>>> arquivo = open('teste.dat', 'r')
>>> print(arquivo.readline())
primeira escrita
>>> arquivo.seek(0)
>>> print(arquivo.readline())
primeira escrita
```

O módulo *pickle*

- A escrita e leitura de arquvios com os métodos *read()* e *write()* utiliza apenas *strings* na troca de informações
- Apesar de se poder converter *strings* em outros tipos, esta limitação dificulta a utilização de tipos mais complexos (como listas, dicionários, tuplas etc.)
- Neste contexto, o módulo *pickle* permite que tipos complexos de dados sejam armazenados em arquivos de forma simples
- Uma vez armazenados em arquivos os dados complexos podem ser recuperados posteriormente

O módulo *pickle*

- A escrita com o método *dump(dado, arquivo)*

```
>>> import pickle
>>> arquivo = open('teste.dat', 'w')
>>> pickle.dump( [1,2,3],  arquivo)
>>> pickle.dump(13.5, arquivo)
```

- A leitura com o método *load(arquivo)* é realizada dado por dado

```
>>> import pickle
>>> arquivo = open('teste.dat', 'r')
>>> a = pickle.load(arquivo) # a = [1,2,3]
>>> b = pickle.load(arquivo) # b = 13.5
```

Exceções

- Sempre que um erro em tempo de execução acontece, é gerada uma **exceção** e o programa pára

```
>>> a = 3/0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: integer division or modulo by zero
```

```
>>> a = []
>>> a[3]
(...)
IndexError: list index out of range
```

Exceções

- Às vezes precisamos executar uma ação que pode gerar uma exceção, mas não queremos que o programa pare
- Nestes casos podemos **tratar uma exceção** utilizando as instruções *try* e *except*
- Exemplo: podemos pedir pro usuário o nome de um arquivo a ser aberto, mas o arquivo pode não existir

```
# ...
nome_arquivo = raw_input('Nome do arquivo: ')
arquivo = open(nome_arquivo, 'r') # pode gerar exceção
# ...
```

Exceções

- No exemplo anterior a exceção que pode ser gerada na abertura do arquivo pode ser tratada da seguinte forma:

```
nome_arquivo = input('Nome do arquivo: ')
try:
    f = open (nome_arquivo, 'r')
except:
    print("Não existe o arquivo", nome_arquivo)
```

- O interpretador executa os comandos do bloco *try*. Se não ocorrem exceções ele ignora o bloco *except.*

- Caso alguma exceção aconteça, o interpretador executa os comandos do bloco except e continua

Exceções

- O tratamento de exceções pode usar vários blocos *except* para poder identificar as exceções com mais precisão
- A documentação de python mantém uma lista das exceções mais comuns na linguagem
- Futuramente, aprenderemos como criar as nossas próprias exceções de acordo com as características dos nossos projetos
- O bom uso e tratamento de exceções é indicativo de qualidade no desenvolvimento de sistemas

Exercícios

1. Crie um programa que copie todo o texto de um arquivo indicado pelo usuário e armazene em um novo arquivo contendo o texto todo em letras maiúsculas

2. Utilize o módulo *pickle* para armazenar tipos variados de dados e depois armazene-os todos em modo texto

3. Crie um programa de perguntas e respostas utilizando arquivos de texto. Trate as possíveis exceções.

4. Crie um programa para armazenar dados de alunos, disciplinas e notas em arquivos.