python™

Rafael Vieira Coelho

rafaelvc2@gmail.com

PARTE 1

9 - Arquivos e Exceções

9.1 Arquivos e exceções

Durante a execução de um programa, seus dados ficam na memória. Quando o programa termina, ou o computador é desligado, os dados na memória desaparecem. Para armazenar os dados permanentemente, você tem que colocá-los em um **arquivo**.

Trabalhar com arquivos é muito parecido com trabalhar com livros. Para utilizar um livro, você tem que abri-lo. Quando você termina, você tem que fechá-lo. Enquanto o livro estiver aberto, você pode tanto lê-lo quanto escrever nele. Em qualquer caso, você sabe onde você está situado no livro. Na maioria das vezes, você lê o livro inteiro em sua ordem natural, mas você também pode saltar através de alguns trechos.

Abrindo um Arquivo: open

Abrir um arquivo cria um objeto arquivo. Neste exemplo, a variável f se referencia ao novo objeto arquivo.

```
f = open("teste.txt", "w")
```

A função open recebe dois argumentos. O primeiro é o nome do arquivo, e o segundo é o modo. Modo "w" significa que estamos abrindo o arquivo para gravação ("write", escrever).

Se não existir nenhum arquivo de nome teste.txt, ele será criado. Se já existir um, ele será substituído pelo arquivo que estamos gravando (ou escrevendo).

Abrindo um Arquivo: open

Abrir um arquivo de uma forma que não apaga o que tinha antes (a = append).

f = open(“teste.txt", "a")

Escrevendo no Arquivo: write

Para colocar dados dentro do arquivo, invocamos o método write do objeto arquivo:

```
f.write("Agora é hora")
f.write("de fechar o arquivo")
```

Fechar o arquivo diz ao sistema que terminamos de escrever (gravar) e que o arquivo está livre para ser lido:

```
f.close()
```

Abrindo o Arquivo para Leitura

Agora podemos abrir o arquivo de novo, desta vez para leitura, e ler o seu conteúdo para uma string.

Desta vez, o argumento modo é “r” para leitura (“reading”, escrever):

```
f = open(“teste.txt", "r")
```

Se tentarmos abrir um arquivo que não existe, temos um erro:

```
f = open("teste.cat", "r")
IOError: [Errno 2] No such file or directory: ’teste.cat’
```

Lendo Dados do Arquivo: read

o método read lê dados do arquivo. Sem argumentos, ele lê todo o conteúdo do arquivo:

```
texto = f.read()
print (texto)
```

Agora é hora de fechar o arquivo.

read também pode receber um argumento que indica quantos caracteres ler:

```
print (f.read(9))
```

Agora é h

Exemplo

A função seguinte, copia um arquivo, lendo e gravando até cinqüenta caracteres de uma vez. O primeiro argumento é o nome do arquivo original; o segundo é o nome do novo arquivo:

```
def copiaArquivo(velhoArquivo, novoArquivo):
        f1 = open(velhoArquivo, "r")
        f2 = open(novoArquivo, "w")
        while 1:
                texto = f1.read(50)
                if texto == "":
                        break
                f2.write(texto + '\n')
        f1.close()
        f2.close()
        return
```

A comando break para a execução e salta para fora do loop;

o fluxo de execução passa para o primeiro comando depois do loop.

Neste exemplo, o loop while é infinito porque o valor 1 é sempre verdadeiro.

O único modo de sair do loop é executando o break.

python™

Outras formas de leitura

O método readline lê todos os caracteres até, e incluindo, o próximo caractere de nova linha:

```
f = open("teste.txt", "r")
for linha in f:
      print (linha)
```

9.2 Diretórios (Pastas)

Quando você cria um novo arquivo abrindo-o e escrevendo nele, o novo arquivo fica no diretório corrente (seja lá onde for que você esteja quando rodar o programa).

Do mesmo modo, quando você abre um arquivo para leitura, Python procura por ele no diretório corrente.

Se você quiser abrir um arquivo que esteja em algum outro lugar, você tem que especificar o caminho (path) para o arquivo, o qual é o nome do diretório (ou folder) onde o arquivo está localizado:

```
f = open("/usr/share/dict/words", "r")
```

Write grava somente strings

Para colocar valores em um arquivo, você tem que convertê-los para strings. Você já viu como fazer isto com str:

```
f.write (str(12.3))
f.write (str([1,2,3]))
```

O problema é que quando você lê de volta o valor, você tem uma string. O Tipo original da informação foi perdido.

De fato, você não pode sequer dizer onde começa um valor e termina outro:

```
f.readline()
```

“12.3[1, 2, 3]"

9.4 Exceções

Sempre que um erro em tempo de execução acontece, ele gera uma exceção.
Neste caso, o programa pára e Python exibe uma mensagem de erro.

Por exemplo, dividir por zero gera uma exceção:

```
print (55/0)
```

```
ZeroDivisionError: int division or modulo by zero
```

Do mesmo modo, acessar um item de lista inexistente:

```
a = []
print (a[5])
```

```
IndexError: list index out of range
```

Ou acessar uma chave que não está em um dicionário:

```
b = {}
print (b["what"])
```

```
KeyError: 'what'
```

Mensagem de Erro

Em cada caso, a mensagem de erro tem duas partes: o tipo do erro antes dos dois pontos, e especificidades do erro depois dos dois pontos.

Normalmente, Python também exibe um “traceback” de onde estava a execução do programa, mas nós temos omitido esta parte nos exemplos.

Às vezes queremos executar uma operação que pode causar uma exceção, mas não queremos que o programa pare.

Nós podemos tratar a exceção usando as instruções try e except.

Usando try except

Por exemplo, podemos pedir ao usuário um nome de arquivo e então tentar abri-lo.

Se o arquivo não existe, não queremos que o programa trave; queremos tratar a exceção:

```
nomedoarquivo = input("Entre com o nome do arquivo: ")
try:
        f = open (nomedoarquivo, "r")
except:
        print ("Não existe arquivo chamado", nomedoarquivo)
```

Lançando uma exceção

Se o seu programa detecta uma condição de erro, você pode fazê-lo lançar uma exceção.

Aqui está um exemplo que toma uma entrada do usuário e testa se o valor é 345.

Supondo que 345 não seja uma entrada válida por uma razão qualquer, nós lançamos uma exceção.

```
def entraNumero():
        x = int(input ("Escolha um número: "))
        if x == 345:
                raise Exception(“345 é um número ruim")
        return x
```

O comando raise toma um argumento que é um objeto da classe Exception. O objeto tomado pelo comando raise recebe, na hora de sua criação, informações específicas sobre o erro.

```
BaseException
 +-- SystemExit
 +-- KeyboardInterrupt
 +-- GeneratorExit
 +-- Exception
      +-- StopIteration
      +-- StandardError
      |    +-- BufferError
      |    +-- ArithmeticError
      |    |    +-- FloatingPointError
      |    |    +-- OverflowError
      |    |    +-- ZeroDivisionError
      |    +-- AssertionError
      |    +-- AttributeError
      |    +-- EnvironmentError
      |    |    +-- IOError
      |    |    +-- OSError
      |    |         +-- WindowsError (Windows)
      |    |         +-- VMSError (VMS)
      |    +-- EOFError
      |    +-- ImportError
      |    +-- LookupError
      |    |    +-- IndexError
      |    |    +-- KeyError
      |    +-- MemoryError
      |    +-- NameError
      |    |    +-- UnboundLocalError
      |    +-- ReferenceError
```

```
|    +-- RuntimeError
      |    |    +-- NotImplementedError
      |    +-- SyntaxError
      |    |    +-- IndentationError
      |    |         +-- TabError
      |    +-- SystemError
      |    +-- TypeError
      |    +-- ValueError
      |         +-- UnicodeError
      |              +-- UnicodeDecodeError
      |              +-- UnicodeEncodeError
      |              +-- UnicodeTranslateError
      +-- Warning
           +-- DeprecationWarning
           +-- PendingDeprecationWarning
           +-- RuntimeWarning
           +-- SyntaxWarning
           +-- UserWarning
           +-- FutureWarning
             +-- ImportWarning
             +-- UnicodeWarning
             +-- BytesWarning
```

http://docs.python.org/library/exceptions.html#bltin-exceptions

Analisando Textos

- Podemos analisar arquivos-texto que contenham blocos inteiros. Muitas obras clássicas de literatura estão disponíveis como arquivos-texto simples, pois estão em domínio público. Os textos usados nesta seção foram extraídos do Projeto Gutenberg (http://gutenberg.org/).

- Vamos obter o texto de Alice in Wonderland e tentar contar o número de palavras do texto. Usaremos o método de string split(), que cria uma lista de palavras a partir de uma string.

- Exemplo de uso do split():

```
>>> title = "Alice in Wonderland"
>>> title.split()
['Alice', 'in', 'Wonderland']
```

Analisando Textos

```
filename = 'Alice.txt'
try:
   with open(filename) as f_obj:
      contents = f_obj.read()
      words = contents.split()
      num_words = len(words)
      print(“O arquivo " + filename + " tem " + str(num_words) + " palavras.")
except FileNotFoundError:
     msg = “Desculpe, o arquivo " + filename + " não existe."
     print(msg)
```

Exercício

1) Faça um programa para gerenciar uma agenda de contatos. Para cada contato armazene o nome, o telefone e o aniversário. O programa deve permitir (1) inserir contato, (2) remover contato, (3) pesquisar um contato pelo nome, (4) listar todos os contatos, (5) listar os contatos cujo nome inicia com uma dada letra, (6) salvar dados em arquivo. Sempre que o programa for encerrado, os contatos devem ser armazenados em um arquivo. Quando o programa iniciar, os contatos devem ser lidos do mesmo arquivo.

Exercício Extra

2) Crie um programa que receba como entrada do usuário o número de alunos de uma disciplina. Devem ser lidos os nomes e as notas dos três trimestres e do exame caso haja necessidade. Salve em um arquivo texto chamado media.txt o nome e a média final de todos os alunos lidos (use o \t para separar o nome da média ao escrever no arquivo). Caso o usuário não informe alguma nota, esta deve ser colocada como zero.