Arquivos e exceções

Arquivos

- São estruturas de dados manipuladas fora do ambiente do programa.
- São armazenados em dispositivos de memória secundária, normalmente para serem utilizados posteriormente por outros programas.
- Arquivos texto são formados por conjuntos de caracteres, enquanto outros tipos de arquivo são geralmente formados por conjuntos de registros.
- Arquivos podem ser usados como alternativa à entrada e/ou saída de dados pelo terminal, mas
  também como complemento.
  - O uso de arquivos de entrada não impede que se faça também algum tipo de leitura de dados do usuário.
    - Por exemplo pedir o nome de arquivos...
  - O uso de arquivos de saída não impede que se faça também algum tipo de saída de dados para o usuário.
    - Por exemplo, mostrar mensagens de erro...
- Um programa pode também manipular quantos arquivos de entrada e/ou saída for preciso.

Em Python

Passos genéricos para processar um arquivo:

'r' = somente para ler
'w' = somente escrever
'r+' = ler e escrever
'a' = acrescentar

- **Abrir o arquivo:**
  - Criar objeto que representará/acessará o arquivo e fará a leitura e/ou escrita, etc.;
  - Receberá o nome do arquivo físico e o tipo de acesso que se deseja como parâmetros – ambos String.
- **Manipular o arquivo:**
  - Usar comandos/métodos de leitura e/ou gravação bem como alguns métodos auxiliares;
  - Os dados são lidos e/ou gravados em formato String.
- **Fechar o arquivo no final:**
  - Libera o controle do arquivo de volta para o sistema operacional.

Passos específicos para ler um arquivo:

```
arqEnt = open (nomeArqEnt, ‘r’)
```

- Onde arqEnt representará o arquivo e nomeArqEnt é o nome do arquivo físico - constante ou variável String.
  - Ex: “nomeArq.txt” ou “c://Temp/nomeArq.txt”.
- Se não for fornecido um caminho completo, o Python assumirá que o arquivo está dentro do diretório padrão da instalação do Python.
- O segundo parâmetro (‘r’) é opcional.
- Obs: O arquivo físico deve existir.
  - Se não existir, a exceção IOError será lançada...
- Para ler o conteúdo de um arquivo texto, que é normalmente retornado em formato String:

```
arqEnt.read(n) #lê e retorna os próximos n bytes.

arqEnt.read() #lê e retorna o arquivo inteiro de uma vez.
```

```
arqEnt.readlines() #lê o arquivo inteiro de uma vez e retorna o
list(arqEnt) #(cont.)como uma lista de linhas.

arqEnt.readline() #lê e retorna uma linha do arquivo;
#Obs: o \n estará armazenado no final da linha...

arqEnt.Close () #Fechamento do arquivo é opcional mas recomenda
```

- Em Python, o comando for pode ser usado para acessar as linhas de um arquivo “uma a uma”
- Dentro do for o problema passa a ser de manipulação de strings, pois é preciso extrair cada dado da linha e mudar o tipo dos dados que não forem do tipo string.

```
for linha in arquivo : # 'linha' receberá uma linha por vez
    comandoUsandoLinha
```

- Comandos auxiliares que podem ser úteis:

```
arqEnt.tell() #retorna a posição atual do cursor que é usado
#para controlar a manipulação do arquivo.

arqEnt.seek(0, 0) #posiciona cursor no início do arquivo.

arqEnt.seek(n) #desloca o cursor n posições. Se n for negativo,

arqEnt.seek(n,p) #desloca o cursor n posições tomando p como ref
#p=0 significa que o referencial é o início do arquivo.
#p=1 significa que a posição atual do cursor é o referencial.
#p=2 significa que o referencial é o final do arquivo.
```

Passos específicos para gravar um arquivo:

```
arqSai = open (nomeArqSai, ‘w’)
```

- Onde arqSai representará o arquivo e nomeArqSai é o nome do arquivo físico - constante ou variável String.
  - Ex: “nomeArquivo.txt” ou “c://Python27/nomeArquivo.txt”.
  - **Se o arquivo físico já existir será sobrescrito!**
  - A opção ‘a’ (ao invés de ‘w’) faz append no arquivo.
- Se não for fornecido um caminho completo, o Python grava o arquivo no diretório padrão da instalação.
  - Obs: Se o diretório não existir ou se o Python não tiver permissão para escrever no diretório, a exceção IOError também será lançada...

```
arqSai.write(String) #gravação
#pode-se usar concatenação (‘+’) ou interpolação (‘%’) no parâme
```

- Para mudança de linha no arquivo deve-se incluir um ou mais \n no conteúdo do string a ser gravado.

```
arqSai.close ()
#Fechamento do arquivo de saída ao final é essencial,
#senão os dados gravados serão perdidos!
```

Exceções

- Recomenda-se usar tratamento de exceções nas operações envolvendo arquivos:
- Um bloco try para encapsular todas as operações em arquivos e também um bloco except para tratar os casos de IOError.

```
try :
    #operações que lidam com arquivos...
```

```
except IOERROR as mens :
    print(“Erro ao lidar com arquivo –”, mens)
```

- O mecanismo de exceções é uma alternativa, utilizada por Python para o tratamento
  dos erros e/ou situações indesejadas.

- **Erros de programação:**
  - Divisão por zero.
  - Acesso a uma posição inválida de string, lista, etc.
  - Acesso a objeto usando uma referência nula.
- **Situações indesejadas:**
  - Ausência de um arquivo procurado localmente.
  - Conexão indisponível em uma comunicação remota.

- Ao invés de códigos de erro, pode-se usar as exceções!

Tratamento de exceções

- Exceções são tratadas usando blocos try-except:

```
try:
    #codigos que podem causar exceções
except excecao1, var1:
    #mensagem para tratar a exceção
except excecao2:
    #mensagem para tratar a exceção
except:
    #mensagem para exceção qualquer
finally:
    #codigo executado no final
```

- A execução do bloco try termina ao final do bloco ou assim que uma exceção acontecer.

- O primeiro except compatível com a exceção será executado e, depois, o fluxo de controle passará para o código seguinte ao último except.
    - Exceções mais específicas devem ser capturadas primeiro.
- Um trecho de código no finally é sempre executado, havendo ou não exceções.
- Quando uma exceção acontece e não há nenhum except compatível no método, a exceção e o fluxo de execução passam para o código invocador e sobem pelas chamadas de subrotinas e/ou métodos até que a exceção seja tratada.
    - Se a exceção não for tratada em lugar nenhum, acontece um erro fatal – o programa pára.

Exemplo

- Fazer um programa para:
  – Ler um arquivo do Detran onde cada registro contém as seguintes informações: placa, marca, modelo e ano de cada veículo, além do CPF do proprietário. O nome físico do arquivo será fornecido pelo usuário.
  – Gravar em um outro arquivo com nome externo “veicVelhos.txt” a placa e ano do veículo e o CPF dos proprietários de todos os veículos que sejam do ano 2000 ou mais velhos.
  – Informar no final ao usuário o número de veículos velhos gravados no arquivo.

```
nomeArqEnt = raw_input ("Digite o nome do arquivo: ")
try :
    arqEnt = open (nomeArqEnt)
    arqSai = open ('C://Python27/veicVelhos.txt', 'w')
    numVeic = 0
    for line in arqEnt :
        placa = line[0:7]
        marca = line[8:13]
        modelo = line[14:22]
        ano = int(line[23:27])
        cpf = line[28:39]
```

```
        if ano <= 2000 :
            arqSai.write('%s %d %s\n' % (placa, ano, cpf) )
            numVeic = numVeic + 1
    print "Número de veículos gravados = %d" % (numVeic)
    arqEnt.close()
    arqSai.close()
except IOError as mens:
    print 'ERRO em arquivo -', mens
except :
    print 'ERRO - Outros'
```

- Se quiser pedir novamente o nome do arquivo em caso de erro no open:

```
nomeArqEnt = raw_input ("Digite o nome do arquivo: ")
erroArq = True
while erroArq :
    try :
        arqEnt = open (nomeArqEnt)
        erroArq = False
    except IOERROR :
        print "Erro na abertura do arquivo -“
        nomeArqEnt = raw_input ("Digite nov. o nome do arquivo:'
```

- Aparte: Como efetivamente garantir tipos de dados na leitura?

```
valorInv = -999999999

n = valorInv
while n == valorInv :
    try :
        n = int(raw_input(‘Digite um inteiro: ‘))
    except :
        print(‘Não foi digitado um número inteiro’)
```

Exercícios

1. Fazer um programa para:
   – Ler um arquivo com profissões, onde cada profissão é formada por um código (positivo) e um nome (String), uma profissão por linha.
   – Receber do usuário uma lista de códigos para que o programa informe o nome de cada profissão.
   – Se o código da profissão não existir no arquivo, mostrar a mensagem “Profissão Inexistente”, gravar em outro arquivo estes códigos de profissões, e continuar.
   – O programa pára com a digitação de um código inválido (negativo ou zero).

2. Considere um arquivo texto com nome externo ´discipold.txt´, contendo informações de disciplinas (Código com 5 posições, nome com 35 posições e créditos com 2 posições), uma disciplina por linha. Faça um programa Python para criar um segundo arquivo com nome externo ´discipnew.txt´ contendo informações das mesmas disciplinas, mas com as seguintes modificações:
   – As disciplinas cujos códigos são IF125 e IF380 devem ser excluídas, i.e., não devem ser gravadas no novo arquivo.
   – Os números de créditos das disciplinas cujos códigos começam por MA devem ser alterados para 5.
   – O novo arquivo terá um campo a mais (carga horária, com 3 posições) cujo valor deve ser o número de créditos da disciplina multiplicado por 15.
   No final o seu programa deve imprimir o número de disciplinas de cada arquivo e também o número de disciplinas que tiveram seus números de créditos alterados.

3. Considere um arquivo texto com nome externo 'discip.txt', contendo informações de disciplinas: código com 5 posições, nome com 35 posições, créditos com 2 posições, e código do centro a que pertence com 2 posições; uma disciplina por linha.
   Considere também um outro arquivo texto nome externo 'centros.txt', contendo informações de centros: código com 2 posições e nome com 25 posições; também um centro por linha.
   Faça um programa Python para:
   – Usando um procedimento, ler o arquivo de centros e colocar seu conteúdo em uma tabela.

– Gravar um arquivo texto com nome externo 'discipCompleto.txt', contendo todas as informações do arquivo 'discip.txt' e mais o nome do centro a que pertence.
– Escrever uma função para achar o nome do centro a partir do seu código.
– No final imprimir a quantidade de disciplinas e de centros.