Introdução à Ciência da Computação

Disciplina: 113913

Prof. Edison Ishikawa

python™

Python 3.0

Capítulo 11

Arquivos e Exceções

UnB | CIC

Sumário

- Arquivos e exceções
  - Arquivos texto
  - Gravando variáveis
  - Diretórios
  - Pickling
  - Exceções

Introdução

- 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.
- Arquivos usualmente são guardados em um disco rígido (HD), num disquete ou em um CD-ROM.

Introdução

- Quando existe um número muito grande de arquivos, eles muitas vezes são organizados dentro de diretórios (também chamados de “pastas” ou ainda “folders“).
- Cada arquivo é identificado por um nome único, ou uma combinação de um nome de arquivo com um nome de diretório.

Arquivos - Analogia com livros

- Trabalhar com arquivos é muito parecido com trabalhar com livros.
  - Para utilizar um livro, você tem que abrí-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 está no livro.
- Tudo isso se aplica do mesmo modo a arquivos. Para abrir um arquivo, você especifica o nome dele e indica o que você quer, seja ler ou escrever (gravar).

Abrindo Arquivos

- Abrir um arquivo cria um objeto arquivo.

a variável f se referencia ao novo objeto arquivo

Exemplo

```
>>> f = open("teste.dat", "w")
>>> print (f)
<open file "teste.dat", mode "w" at fe820>
```

nome do arquivo **o modo** **localização do objeto na memória**

- 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.dat, ele será criado. Se já existir um, ele será substituído pelo arquivo que estamos gravando (ou escrevendo).

Escrevendo e Fechando um Arquivo

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

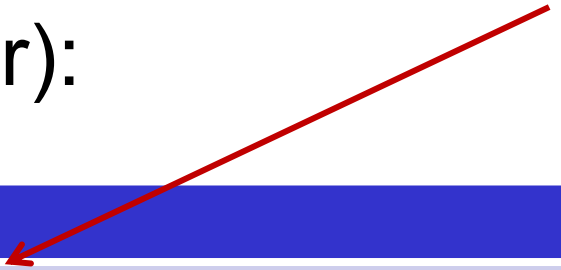
Exemplo

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

- Fechar o arquivo diz ao sistema que terminamos de escrever (gravar) e que o arquivo está livre para ser acessado por outro programa

Lendo Arquivos

- Agora podemos abrir o arquivo de novo, desta vez somente para leitura, e ler o seu conteúdo para uma string.
- Desta vez, o argumento modo é “r” para leitura (“reading“, escrever):



Exemplo

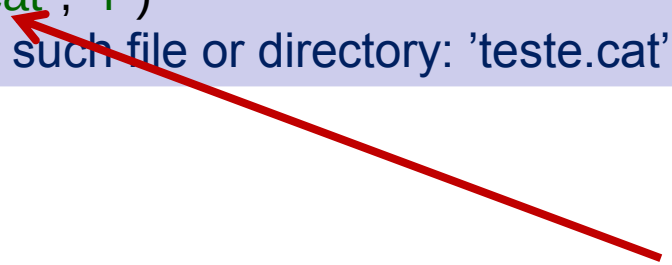
```
>>> f = open("teste.dat", "r")
```

Erro na abertura de arquivo

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

Exemplo

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



teste.**c**at não existe

é teste.**d**at

Lendo arquivos

- Agora sem erros!

Exemplo:

```
>>> f = open("teste.dat", "r")
>>>
>>> texto = f.read()
>>> print (texto)
Agora é horade fechar o arquivo
```

Não existe espaço entre “hora” e “de” porque não colocamos (gravamos) um espaço entre as strings

Lendo número fixo de caracteres

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

Exemplo:

```
>>> f = open("teste.dat", "r")
>>> print (f.read(9))
Agora é h
```

- Se não houver caracteres suficientes no arquivo, read retorna os caracteres restantes. Quando chegamos ao final do arquivo, read retorna a string vazia:

Exemplo:

```
>>> print (f.read(1000006))
orade fechar o arquivo
>>> print (f.read())

>>>
```

Funções que manipulam arquivos

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

Exemplo:

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

O loop while é infinito porque o valor 1 é sempre verdadeiro.
O único modo de sair do loop é executando o break, o que ocorre quando texto é a string vazia, o que ocorre quando alcançamos o fim do arquivo.

comando novo: break
Faz saltar a execução para fora do loop;
o fluxo de execução passa para o primeiro comando depois do loop.

Arquivos texto

- Um arquivo texto é um arquivo que contém caracteres imprimíveis e espaços, organizados dentro de linhas separadas por caracteres de nova linha.
- Pyhton é especialmente projetado para processar arquivos texto
  - possui métodos que tornam esta tarefa mais fácil.

Exemplo: criar um arquivo texto com três linhas de texto separadas por caracteres de nova linha:

```
>>> f = open("teste.dat", "w")
>>> f.write("linha um\nlinha dois\nlinha três\n")
>>> f.close()
```

Arquivos texto

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

Exemplo:

```
>>> f = open("teste.dat", "r")
>>> print (f.readline())
linha um

>>>
```

Repare que ele pulou uma linha!

Arquivos Texto

- readlines retorna todas as linhas restantes como uma lista de strings:

Exemplo:

```
>>> print (f.readlines())
['linha dois\012', 'linha três\012']
```

- Neste caso, a saída está em formado de lista, o que significa que as strings aparecem entre aspas e o caractere de nova linha aparece como a sequência de escape 012.

Arquivos Texto

- No fim do arquivo, readline retorna a string vazia e readlines retorna a lista vazia:

Exemplo:

```
>>> print (f.readline())

>>> print (f.readlines())
[ ]
>>>
```

Arquivos Texto

- filtraArquivo faz uma cópia de velhoArquivo, omitindo quaisquer linhas que comecem por #:

Exemplo de um programa de processamento de linhas.

```
def filtraArquivo(velhoArquivo, novoArquivo):
        f1 = open(velhoArquivo, "r")
        f2 = open(novoArquivo, "w")
            while 1:
                texto = f1.readline()
                if texto == "":
                    break
                if texto[0] == '#':
                    continue
                f2.write(texto)
        f1.close()
        f2.close()
        return
```

O comando continue termina a iteração corrente do loop, mas continua iterando o loop. O fluxo de execução passa para o topo do loop, checa a condição e prossegue conforme o caso.

Assim, se texto for a string vazia, o loop termina.

Se o primeiro caractere de texto for o jogo da velha (? # ?), o fluxo de execução passa para o topo do loop..

Somente se ambas as condições falharem é que texto será copiado para dentro do novo arquivo

Gravando variáveis

- O argumento de write tem que ser uma string, assim se quisermos colocar outros valores em um arquivo, temos de convertê-los para strings primeiro.
- A maneira mais fácil de fazer isso é com a função str:

Exemplo:

```
>>> x = 52
>>> f.write(str(x))
```

Gravando variáveis

- Uma alternativa é usar o operador de formatação %.
  - Quando aplicado a inteiros, % é o operador módulo. Mas quando o primeiro operador é uma string, % é o operador de formatação.
- O primeiro operando é a string de formatação, e o segundo operando é uma tupla de expressões.
  - O resultado é uma string que contém os valores das expressões, formatadas de acordo com a string de formatação.

a seqüência de formatação “%d” significa que a primeira expressão na tupla deve ser formatada como um inteiro. Aqui a letra **d** representa **decimal**.

Exemplo:

```
>>> carros = 52
>>> "%d" % carros
’52’
```

O resultado é a string “52”, que não deve ser confundida com o valor inteiro 52.

Gravando variáveis

- Uma seqüência de formatação pode aparecer em qualquer lugar na string de formatação, assim, podemos embutir um valor em uma seqüência:

Exemplo:

```
>>> carros = 52
>>> "Em julho vendemos %d carros." % carros
'Em julho vendemos 52 carros.'
```

Gravando variáveis

- A seqüência de formatação “%f” formata o próximo item da tupla como um número em ponto flutuante, e “%s” formata o próximo como uma string:

Exemplo:

```
>>> "Em %d dias fizemos %f milhões %s." % (34,6.1,’reais’)
’Em 34 dias fizemos 6.100000 milhões de reais.’
```

Gravando variáveis

- Por padrão, o formato de ponto flutuante exibe seis casas decimais.
- O número de expressões na tupla tem que ser igual ao número de sequências de formatação na string. Além disso, os tipos das expressões têm que ser iguais aos da sequência de formatação:

Exemplo:

```
>>> "%d %d %d" % (1,2)
TypeError: not enough arguments for format string
>>> "%d" % 'reais'
TypeError: illegal argument type for built-in operation
```

Note que não existem expressões suficientes

Veja que a expressão é do tipo errado

Gravando variáveis

- Para um controle maior na formatação de números, podemos especificar o número de dígitos como parte da sequência de formatação:

Exemplo:

```
>>> "%6d" % 62
'    62'
>>> "%12f" % 6.1
'   6,100000'
>>> "%-6d" % 62
'62    '
```

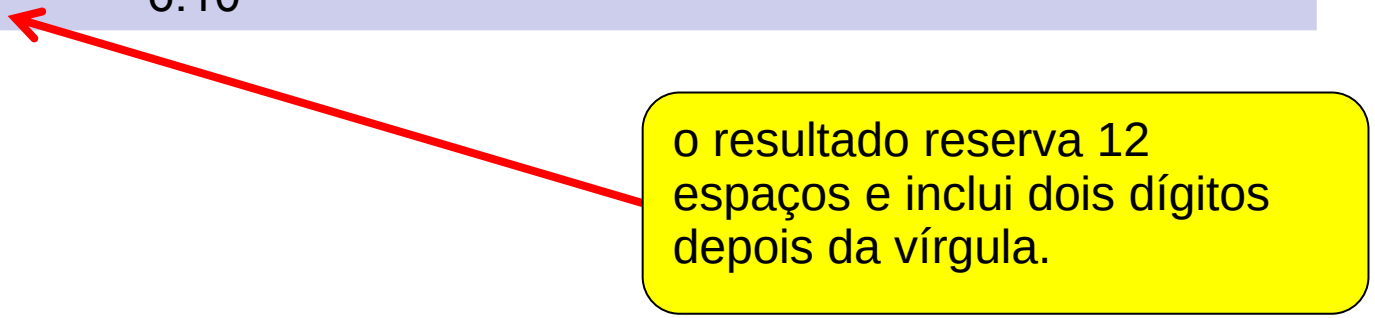
Se o número de espaços for negativo, os espaços serão adicionados depois

Gravando variáveis

- Para números em ponto-flutuante, também podemos especificar o número de dígitos depois da vírgula

Exemplo:

```
>>> "%12.2f" % 6.1
’          6.10’
```



- Esta formatação é útil para exibir valores monetários com os centavos alinhados.

Gravando variáveis - Exemplo

- Imagine um dicionário que contém nomes de estudantes como chaves e salários-hora como valores

Exemplo: função que imprime o conteúdo do dicionário como um relatório formatado

```
def relatorio(salarios):
        estudantes = salarios.keys()
        estudantes.sort()
        for estudante in estudantes:
                print ("%-20s %12.02f" % (estudante, salarios[estudante]))
```

Para testar esta função, criaremos um pequeno dicionário e imprimiremos o conteúdo:

```
>>> salarios = {'maria': 6.23, 'joão': 5.45, 'josué': 4.25}
>>> relatorio(salarios)
joão                 5.45
josué                4.25
maria                6.23
```

Controlando a largura de cada valor, podemos garantir que as colunas ficarão alinhadas, desde que os nomes contenham menos que vinte e um caracteres e os salários sejam menores do que um bilhão de reais por hora.

Diretórios

- 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.

Diretórios

- 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.

Exemplo:

```
>>> f = open("/usr/share/dict/words", "r")
>>> print (f.readline())
Aarhus
```

abre um arquivo chamado words que reside em um diretório de nome dict, o qual reside em share, o qual reside em usr, o qual reside no diretório de mais alto nível do sistema, chamado /.

Obs 1: Você não pode usar / como parte do nome de um arquivo; ela é um caractere reservado como um delimitador entre nomes de diretórios e nomes de arquivos.

Obs 2: O arquivo /usr/share/dict/words contém uma lista de palavras em ordem alfabética, na qual a primeira palavra é o nome de uma universidade Dinamarquesa.

Pickling

- Para colocar valores em um arquivo, você tem que convertê-los para strings

Exemplo: usando 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:

Exemplo:

```
>>> f.readline()
"12.3[1, 2, 3]"
```

Pickling

- A solução é o pickling, assim chamado porque “preserva” estruturas de dados.
- O módulo pickel contém os comandos necessários.
- Para usá-lo, importe pickle e então abra o arquivo da maneira usual:

Exemplo:

```
>>> import pickle
>>> f = open("test.pck", "w")
```

Pickling

- Para armazenar uma estrutura de dados, use o método dump e então feche o arquivo do modo usual:

Exemplo:

```
>>> pickle.dump(12.3, f)
>>> pickle.dump([1,2,3], f)
>>> f.close()
```

Pickling

- Então, podemos abrir o arquivo para leitura e carregar as estruturas de dados que foram descarregadas (dumped):

Exemplo:

```
>>> f = open("test.pck", "r")
>>> x = pickle.load(f)
>>> x
12,3
>>> type(x)
<class "float">
>>> y = pickle.load(f)
>>> y
[1, 2, 3]
>>> type(y)
<class "list">
```

- Cada vez que invocamos load, obtemos um único valor do arquivo, completo com seu tipo original.

Exceções

- Sempre que que um erro em tempo de execução acontece, ele gera uma exceção.
- Usualmente, o programa para e Python exibe uma mensagem de erro.

Exemplo: dividir por zero gera uma exceção

```
>>> print (55/0)
ZeroDivisionError: int division or modulo by zero
```

Exceções

- Do mesmo modo, acessar um item de lista inexistente:

Exemplo:

```
>>> a = []
>>> print (a[5])
IndexError: list index out of range
```

- Ou acessar uma chave que não está em um dicionário:

Exemplo:

```
>>> b = {}
>>> print (b["what"])
KeyError: 'what'
```

Exceções

- 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

Exemplo:

```
>>> def fatorial(n):
    if n > 1:
        return n*fatorial(n-1)
    else:
        return 1

>>> n = int(input())
5
>>> print(fatarila(n))
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    print(fatarila(n))
NameError: name 'fatarila' is not defined
>>>
```

Exceções

- 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.

Exemplo: pedir ao usuário um nome de arquivo e então tentar abrí-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)
```

- A instrução try executa os comandos do primeiro bloco. Se não ocorrerem exceções, ele ignora a instrução except. Se qualquer exceção acontece, ele executa os comandos do ramo except e continua.

Exceções

- Podemos encapsular esta habilidade numa função: existe toma um nome de arquivo e retorna verdadeiro se o arquivo existe e falso se não existe:

Exemplo:

```
def existe(nomedoarquivo)
    try:
        f = open(nomedoarquivo)
        f.close()
        return 1
    except:
        return 0
```

- Pode-se usar múltiplos blocos except para tratar diferentes tipos de exceções. Veja o manual do Python.

Exceções

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

Exemplo: toma uma entrada do usuário e testa se o valor é 17. Se 17 não é uma entrada válida, lança uma exceção

```
class ErroNumeroRuim(Exception):
      pass

def entraNumero():
      x = int(input ("Escolha um número: "))
      if x == 17:
            raise ErroNumeroRuim("17 é um número ruim")
      return x
```

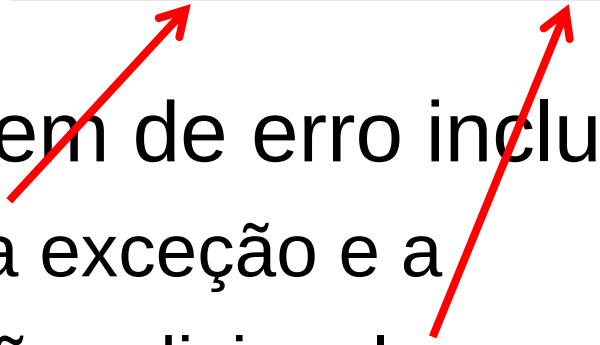
- O comando **raise** toma um argumento que é um objeto da classe Exception.
  - O objeto recebe, na hora de sua criação, informações específicas sobre o erro.
  - ErroNumeroRuim é uma nova classe de exceção criada para esta aplicação.

Exceções

- Se a função que chamou entraNumero trata o erro, então o programa pode continuar; de outro modo, Pyhton exibe uma mensagem de erro e sai:

Exemplo:

```
>>> entraNumero()
Escolha um número: 17
ErroNumeroRuim: 17 é um número ruim
```



- A mensagem de erro inclui a
  - classe da exceção e a
  - informação adicional que você forneceu.

Exceções

- Exercício
  - Como um exercício, escreva uma função que use entraNumero para pegar um número do teclado e que trate a exceção ErroNumeroRuim.

Referências

- Aprenda Computação com Python 3.0, Versão 1. Allen Downey, Jeff Elkner and Chris Meyers
  - ➔ Capítulo 5: Funções frutíferas