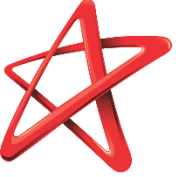


CEUNSP
Centro Universitário
N. Sra. do Patrocínio

CIÊNCIA DA COMPUTAÇÃO
2º SEMESTRE

Laboratório de Desenvolviemnto de Algoritimos

Prof. Me. Paulo Djalma Martins
paulo.martins@ceunsp.edu.br

Tratando Exceções

Exemplo:

```
numero = 35
resultado = numero/0
```

resultado da execução → *exceptions.ZeroDivisionError: integer division or modulo by zero*

Obs: Objetivo não é resolver potenciais erros, mas sim evitar que essas exceções encerrem a execução do script.

try e **except**

```
try:
        arquivo = "notas.txt"
        arq = open (arquivo,'r')
        texto = arq.read()
except:
        print ("Arquivo" + arquivo + " não foi encontrado")
```

Definindo FUNÇÕES

Função → é um conjunto de instruções que podem ser acessadas em qualquer parte de um programa

Podem receber argumentos de entrada e/ou retornar algum valor para o código que chamou a função.

Exemplo1:

```
def imprimirMensagem ():
        print ("2 sem de Ciência da Computação")

#inicio do script
imprimirMensagem()
```

Definindo FUNÇÕES

Exemplo2:

```
def imprimirMensagem (mensagem):
        print (mensagem)

#inicio do script
imprimirMensagem("Essa é uma ótima turma")
```

Definindo FUNÇÕES – Utilizando a cláusula **return**

Exemplo2:

```
    def somar (num1, num2):
→           return num1 + num2
```

```
n1 = int(input("Digite o primeiro numero"))
n2 = int(input("Digite o segundo numero"))

print("A soma de " + str(n1) + "com " + str(n2) + " é: " + str(somar(n1,n2)))
```

Definindo FUNÇÕES – Utilizando a cláusula **return**

Especificando um valor padrão, se não foi informado o mesmo.
Se informado, o valor padrão é desconsiderado.

Exemplo3:

```
def potencia (num1, num2=2):
        return num1 ** num2

print ("5 elevado a 2 é : " + str(potencia(5)))
print ("5 elevado a 3 é : " + str(potencia(5,3)))
```

MÓDULOS

As funções são encapsulados em módulos e importados conforme a necessidade dos programas.

Tanto podemos definir nossos próprios módulos como podemos importar módulos fornecidos pelo Python.

Muitas funcionalidades complexas são disponibilizadas por módulos predefinidos.

MÓDULOS

Módulo	Finalidade
gelopt	Manipulação de parâmetros de linha de comando
math	Funções para processamento matemático
os	Funções ligadas ao sistema operacinal
random	Funções para geração de número randômico
socket	Criação de socket para comunicação em redes
string	Manipulação de strings
sys	Funções ligadas ao interpretador Python
time	Manipular valores de tempo
tkinter	Funções para criação de interfaces gráficas elaboradas
win32gui	Interfaces gráficas para ambientes Windows

Módulos Módulo Math

Para chamar uma função, temos que especificar

o **nome do módulo**, seguido por ponto (.) e pelo **nome da função**.

Exemplo:

```
import math

# logaritmo na base 10
valor1 = math.log10(20)

# seno de um valor
valor2 = math.sin(valor1)

resultado = valor1 * math.pi + math.sqrt(valor2)

print (resultado)
```

Obs:

Na linha 1:cláussula **import** indica que o módulo math será utilizado.

As funções log10(), sin(), pi() e sqrt() são funções desse módulo.

Módulos

Módulo OS

Com este módulo, podemos acessar variáveis e programas dos sistema.

Com **módulo Os**, podemos desenvolver, por exemplo, scripts alternativos ao Programa principal.

Exemplo:

```
import  os
os.system (´dir´)
```

Obs:

O exemplo ao lado utiliza a função system() do módulo os para acessar diretamente o comando dir.

Módulos

Módulo OS

Todo script Python que tenha extensão .py pode ser importado como um módulo.

Podemos assim definir funções, variáveis em um script e depois importá-los em outro programa.

Exemplo 1 :

```
def fatorial (numero):
        if numero == 1:
                return 1
        if numero == 0:
                return 1
        return numero * fatorial (numero – 1)
```

Obs:

Salve este programa com o nome → shazan.py

Exemplo 2 :

```
import shazan

print (shazan.fatorial (10))
```

Orientação a Objetos – POO

Classificação, Abstração e Instanciação

No início da infância, o ser humano aprende e pensa de maneira semelhante à filosofia da orientação a objetos, representando seu conhecimento por meio abstrações e classificações.

As crianças aprendem conceitos simples, como pessoa, carro e casa, e, ao fazerem isso definem classes, ou seja, grupos de objetos, sendo cada um deles um exemplo de um determinado grupo, tendo as mesmas características e comportamentos de qualquer objeto do grupo em questão.

- Qualquer coisa que tiver cabeça, tronco e membros torna-se uma pessoa
- Qualquer construção na qual as pessoas possam entrar passa a ser uma casa
- Qualquer peça de metal com quatro rodas que se locomova de um lugar para outro transportando pessoas recebe denominação de carro.

Esse processo envolve um grande esforço de abstração, em razão, de os carros apresentarem diferentes formatos, cores, e estilos.

Uma criança deve se sentir um pouco confusa no começo ao descobrir que o objeto amarelo e o objeto vermelho têm, ambos, a mesma classificação: **carro**.

A partir desse momento, precisa abstrair o conceito de carro para chegar à conclusão de que "**carro**" é um termo geral que se refere a muitos objetos.

Cada um dos objetos-carro tem características semelhantes entre si:

- Todos têm quatro rodas e, no mínimo, duas portas, além de luzes de farol e freio, bem como vidros frontais e laterais.

Além disso, os objetos-carro podem realizar determinadas tarefas sendo a principal delas transportar pessoas de um lugar para outro.

Orientação a Objetos – POO

Classificação, Abstração e Instanciação

No momento em que a criança compreende esse conceito, percebe que "**carro**" é a denominação de um grupo, ou seja, ela abstraiu uma classe: a classe **carro**.

Assim, sempre que perceber a presença de um objeto com as características já determinadas, concluirá que aquele objeto é um exemplo do grupo carro, ou seja, uma **instância** da classe carro.

Instanciação constitui-se em criar um exemplo de um grupo, uma classe.

Quando instanciamos um objeto de uma classe, estamos criando um novo item do conjunto representado por essa classe, com as mesmas características e comportamentos de todos os outros objetos já instanciados.

Orientação a Objetos – POO

Classes de Objetos

- Uma classe representa uma categoria e os objetos são os membros ou exemplos dessa categoria.

- Em geral a classe contém atributos e métodos.

Orientação a Objetos – POO

Atributos ou Propriedades

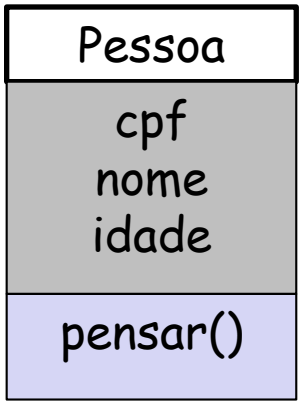
- Os atributos representam as características de uma classe, ou seja, as peculiaridades que costumam variar de um objeto para outro, como o nome, o CPF ou a idade em um objeto da classe Pessoa.
- Os atributos são apresentados na segunda divisão da classe e contém duas informações: o **nome** que identifica o atributo e o **tipo de dado** que o atributo armazena como integer, float ou string.
- Podemos dizer que não é a classe que contém os atributos, mas, sim, as instâncias, os objetos da classe.
- Nunca poderemos trabalhar com a classe **Pessoa**, apenas com suas instâncias, como Carlos, Pedro ou José, que são nomes que identificam três objetos da classe **Pessoa**.
- Todas as instâncias de uma mesma classe têm os mesmos atributos, no entanto eles podem assumir valores diversos.
  - EXEMPLO:
    - atributo nome do objeto **pessoa1** = Pedro
    - atributo nome do objeto **pessoa2** = José

Orientação a Objetos – POO

Operações, Métodos ou Comportamentos

- Classes costumam ter métodos, também conhecidos como operações ou comportamentos .
- Um método representa a atividade que um objeto de uma classe pode executar.
- Um método representa um conjunto de instruções executadas quando o método é chamado.

  EXEMPLO:
  - Um objeto da classe **Pessoa** pode executar a atividade de pensar



- Acrescentamos uma terceira divisão para armazenar o método pensar().

Orientação a Obejtos - OO

Programas orientados a objetos utilizam objetos como elementos principais na Construção de programas de computador

Exemplo 1 :

```
mensagem = "Boa noite a todos!"

#mensagem impressa sem alterações
print (mensagem)

#todas as letras maiúsculas
print (mensagem.upper())

#todas as letras minúsculas
 print (mensagem.lower())
```

Obs:

No exemplo ao lado, a variável mensagem armazena um valor textual e possui um conjunto de **métodos** (funções ou rotinas) que podem ser utilizados para processar o valor armazenado na variável.

Assim a variável mensagem representa, na verdade, um **objeto** do tipo string.

A variável mensagem pode apontar para objetos de diferente tipos durante a execução do programa.

A estrutura fundamental para definir objetos é a **classe**.

Toda classe possui:
nome
atributos(variáveis)
métodos(funções)

porém apenas o nome é obrigatório.

CLASSES

Exemplo de classe :

```
class Pessoa:
      nome = None
      idade = 20

      def imprimirNome (self):
            print (self.nome)

      def imprimiridade (self):
            print (self.idade)

      def colocaridade (self, num):
            self.idade = num

      def colocarNome (self, nom):
            self.nome = nom

emerson = Pessoa()
emerson.colocaridade(30)
emerson.colocarNome ("Emerson Ferrovia Costa")

emerson.imprimirNome ()
emerson.imprimiridade ()
```

Obs:

Na classe Pessoa foram definidos duas variáveis e quatro módulos.

Foi utilizado a palavra reservada **None** para indicar que a variável nome possui inicialmente nenhum valor.

A palavra reservada **self** representa a instância do próprio objeto e deve ser sempre o primeiro argumento dos métodos em uma classe.

É com o self que referimos ao próprio objeto, para acessar, por exemplo, suas variáveis.

CONSTRUTOR - __init__
(2 sublinhados + init + 2 sublinhados)

Exemplo de classe :

```
class Pessoa:
    nome = None
    idade = None

    def __init__ (self,n,i):
        self.nome = n
        self.idade = i

    def imprimirNome (self):
        print (self.nome)

    def imprimiridade (self):
        print (self.idade)

    def colocaridade (self, num):
        self.idade = num

    def colocarNome (self, nom):
        self.nome = nom

emerson = Pessoa("Emerson Ferrovia Costa", 30)

emerson.imprimirNome ()
emerson.imprimiridade ()
```

Obs:

__init__

Este método é utilizado para passar atributos iniciais para a definição dos objetos.

Trabalhando na plataforma Windows

Criando uma Lista pelo IDLE

```
movies = ["Exterminador do futuro", "Rambo", "Star Wars", "Tropa de Elite"]

print (movies[1])
```

Para calcular numero de ítens da Lista:

```
print (len(movies))
```

Usando outros métodos: append(), pop() e extend()

```
movies.append("Os caçadores da Arca Perdida")
print (movies)
movies.pop()
print (movies)
movies.extend(["O poderoso Chefão", "O mágico de Oz", "Tubarão"])
print (movies)
```

Trabalhando na plataforma Windows

Removendo um item específico da Lista com método remove()

```
movies.remove("Rambo")
print (movies)
```

Adicione um item de dados antes de um local específico usando método insert ()

```
movies.insert(0,"Rambo")
print (movies)
```

Trabalhando na plataforma Windows

Usando um loop **For**:

```
for cada_filme in movies:
        print (cada_filme)
```

Exemplo 2:

```
for i in range (len(movies)):
        print (movies[i])
```

Obs:

A palavra-chave " in " separa o identificador de destino da Lista.

Os dois pontos " : " seguem o seu nome da Lista e indicam o inicio de seu código de processamento da lista.

Trabalhando na plataforma Windows

Usando um loop **while**:

```
cont = 0
while cont < len(movies):
        print (movies[cont])
         cont = cont+1
```

Trabalhando na plataforma Windows

Armazenando Lista dentro da Lista

```
movies = ["Exterminador do futuro", 1985, "Arnold Schwarzenegger",
    ["Rambo", 1982, "Sylvester Stallone", ["Star Wars", 1977, "Luke
    Skywalker/ Mark Hamill", ["Tropa de Elite", 2007, "Wagner Moura"]]]]

for cada_item in movies:
        print (cada_item)
```

Trabalhando na plataforma Windows

Os dados são uma lista que contém um lista aninhada que contém em si uma lista aninhada. O problema é que o código só sabe processar uma lista aninhada em uma lista de inclusão.

A solução é adicionar mais código para lidar com a lista aninhada adicionalmente.

Verificando se o item atual é uma Lista para poder imprimir

```
for cada_item in movies:
        if isinstance(cada_item, list):
            for aninhado_item in cada_item:
                print (aninhado_item)
        else:
            print (cada_item)
```

Trabalhando na plataforma Windows

Os dados são uma lista que contém um lista aninhada que contém em si uma lista aninhada. O problema é que o código só sabe processar uma lista aninhada em uma lista de inclusão.

A solução é adicionar mais código para lidar com a lista aninhada adicionalmente.

Verificando se o item atual é uma Lista para poder imprimir

```
for cada_item in movies:
        if isinstance (cada_item, list):
            for aninhado_item in cada_item:
                    if isinstance (aninhado_item, list):
                            for proximo_item in aninhado_item:
                                    print (proximo_item)
                    else:
                            print (aninhado_item)
        else:
            print (cada_item)
```

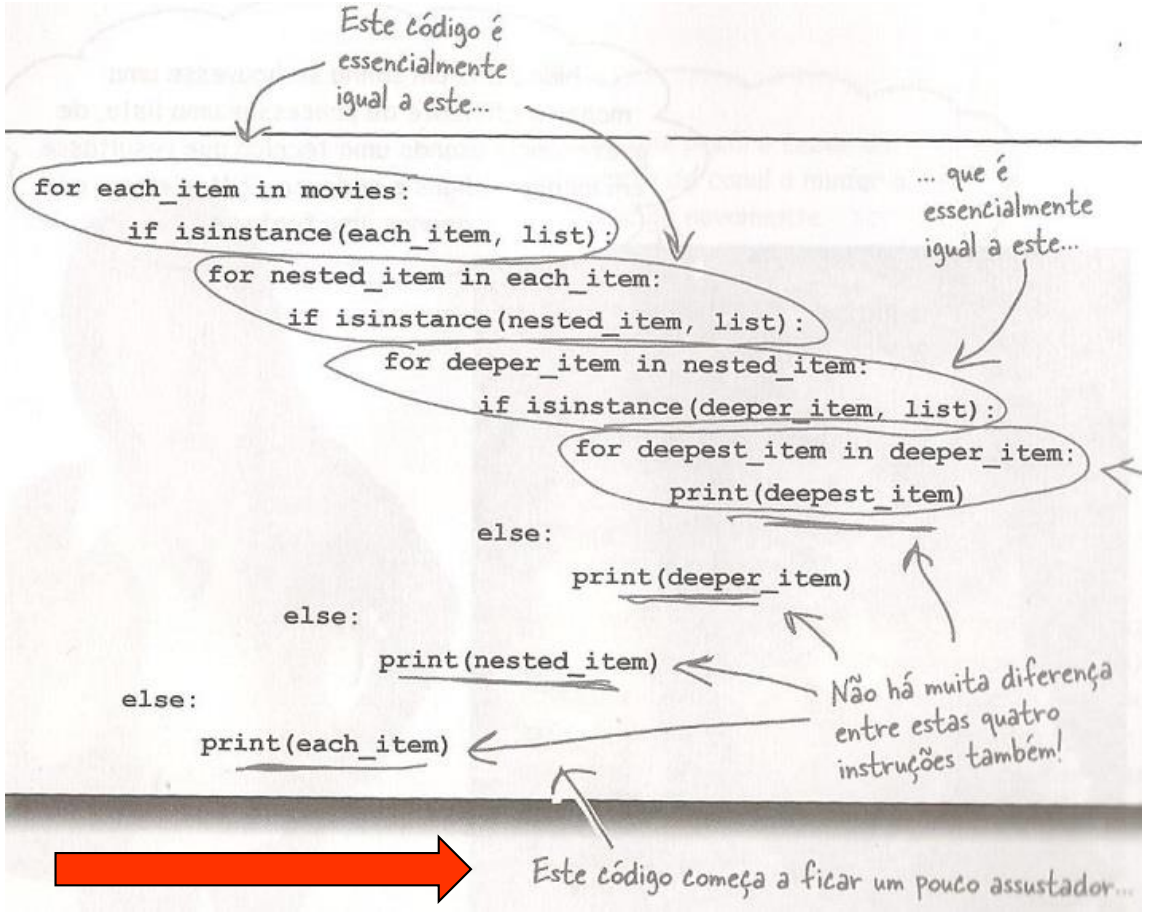
Trabalhando na plataforma Windows

Pode-se incluir outros filmes, cada um com o ator coadjuvante que atuou neles.

Se for adicionado os dados, poderá alterar o código para imprimi-los também?

Os dados têm de ser embutidos como outra lista aninhada dentro da lista já profundamente aninhada de atores coadjuvantes.

Alterar o seu código é apenas uma questão de acrescentar outro loop **for** e uma instrução **if**.



Trabalhando na plataforma Windows

Pode-se processar uma lista, de preferência usando uma técnica que resultasse em **menos códigos** e nada mais?

Quando o código se repete desta forma, a maioria dos programadores procura uma maneira de obter o padrão geral do código e transformá-lo em uma **função reutilizável**.

NÃO REPITA CÓDIGO; CRIE UMA FUNÇÃO

Módulos de funções

Um módulo é simplesmente um arquivo de texto que contém o código Python. O principal requisito é que o nome do arquivo deve terminar em **.py**: a extensão Python.

Para transformar sua função em um módulo, salve seu código em um arquivo nomeado.

No IDLE, clique em ***File*** / ***New File*** / digite o código abaixo e salve como **filmes.py**

```
""" Este é o módulo ' filmes.py ' e fornece uma função chamada print_filmes ()
que imprime listas que podem ou não incluir listas aninhadas """

def print_filmes(the_list):
        for cada_item in the_list:
            if isinstance (cada_item, list):
                    print_filmes (cada_item)
            else:
                    print (cada_item)
```

Módulos de funções

Clique com botão direito no arquivo ***filmes.py*** e edite ele com o IDLE Python.

Aperte a tecla **F5** para executar o código do módulo

Parece que nada acontece, exceto o shell do Python "reiniciar" e um prompt vazio aparecer

O que aconteceu é que o interpretador Python foi redefinido e o código em seu módulo foi executado. O código define a função, mas, além disso, faz pouca coisa. O interpretador está esperando que você faça algo com sua função recém-definida, portanto, **deve-se criar um lista de listas e chamar a função**.

Módulos de funções – PREPARE A SUA DISTRIBUIÇÃO

A fim de compartilhar o módulo recém-criado, é necessário preparar uma **distribuição**. Este é nome Python dado à coleção de arquivos que, juntos, permitem criar, reunir distribuir seu módulo.

- **1º. PASSO:**

Comece criando uma pasta para o seu módulo

- Com a pasta criada, copie o arquivo do módulo **filmes.py** para a pasta.

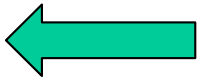
- **2º. PASSO:**

Crie um arquivo chamado "setup.py" em sua nova pasta

- Este arquivo contém metadados sobre sua distribuição. Edite esse arquivo, adicionando o seguinte código:

```
from distutils.core import setup

setup (
        name            = 'filmes',
        version         = '1.0.0',
        py_modules      = ['filmes'],
        author          = 'hfpython',
        author_email    = 'hfpython@firstlab.com',
        url             = 'https://www.headfirstlabs.com',
        description     = 'Uma simples impressão da lista  filmes',
      )
```



Importe a função de configuração dos utilitários de distribuição do Python

Módulos de funções – PREPARE A SUA DISTRIBUIÇÃO

- **3°. PASSO:**

Abra uma janela do terminal dentro de sua pasta que foi criada com os arquivos filme.py e setup.py

- Digite o comando:

python setup.py sdist

- **4°. PASSO:**

Instale sua distribuição em sua **cópia local do Python**

- No terminal, digite o seguinte comando:

python setup.py install

Módulos de funções

Revisão da distribuição

Graças aos utilitários de distribuição do Python, o módulo foi *transformado* em uma distribuição e *instalado* na cópia local do Python.

Você começou com uma única função, que forneceu em um arquivo chamado filmes.py, criando um módulo.

Em seguida criou uma pasta para hospedar seu módulo.

A adição de um arquivo chamado setup.py à sua pasta permitiu construir e instalar sua distribuição, o que resultou em uma série de outros arquivos e duas pastas novas que apareceram dentro de sua pasta criada.

Esses arquivos e pastas são criados para você pelos utilitários de distribuição.

Módulos de funções

Crie listas aninhadas e use o módulo filmes

NO IDLE, digite:

```
import filmes

movies = ["Exterminador do futuro", 1985, "Arnold Schwarzenegger",
    ["Rambo", 1982, "Sylvester Stallone", ["Star Wars", 1977, "Luke
    Skywalker/ Mark Hamill", ["Tropa de Elite", 2007, "Wagner
    Moura"]]]]

filmes.print_filmes(movies)
```

Módulos de funções

Como sei onde estão os módulos Python em meu computador?

NO IDLE, digite:

```
import sys;sys.path
```

Módulos de funções

Melhorando o módulo filmes.

Alterando a função para usar **range()** para recuar quaisquer listas aninhadas em um numero específico de tabulações.

```
""" Este é o módulo "filmes.py" e fornece uma função chamada print_filmes ()
que imprime listas que podem ou não incluir listas aninhadas """

def print_filmes(the_list, level):
        "" Esta função recebe um argumento posicional chamado "the_list", que é
              qualquer lista do Python (de - possivelmente listas de filmes). Cada item
    de dados na lista fornecida é impresso na tela em sua própria linha. Um
    segundo argumento chamado "level" é usado para inserir tabulações
    quando uma lista aninhada é encontrada. ""
        for cada_item in the_list:
                if isinstance (cada_item, list):
                        print_filmes (cada_item, level+1)
                else:
                        for tab_stop in range(level):
                                print("\t", end=' ')
                        print (cada_item)
```

Operação com Arquivos

open() → faz a referência a um objeto do tipo arquivo
read() → faz a leitura do arquivo
write() → escreve valores no arquivo

Exemplo 1

```
arquivo = open ("relatorio.txt")

texto = arquivo.read()

print (texto)
```

Operação com Arquivos

open() → faz a referência a um objeto do tipo arquivo
read() → faz a leitura do arquivo
write() → escreve valores no arquivo

Exemplo 2

```
arquivo = open ("relatorio.txt",'w')

for num in range (1,10):
    arquivo.write("*" * num + "\n")

arquivo.close()
```

Função	Utilização
read()	Retorna uma string com todo o conteúdo do arquivo
readline()	Retorna a próxima linha do arquivo, incrementando o ponteiro que marca a posição atual
readlines()	Retorna o conteúdo do arquivo em uma lista, com cada linha do arquivo sendo um elemento da lista
write (string)	Escreve a string especificada no arquivo, conforme a opção de utilização indicada no método open()
seek (pos)	Muda a posição atual do arquivo para o valor específico por pos.
close()	Fecha o arquivo

Exemplo 1

```
arquivo = open ("relatorio.txt",'r')
linha = arquivo.readline()

while (linha != ""):
        print (linha)
        linha = arquivo.readline()
```

Exemplo 2

```
arquivo = open ("relatorio.txt",'r')
linhas = arquivo.readlines()

cont=0
while (cont < len (linhas)):
        print (linhas[cont])
        cont+=1
```

Lendo dados do arquivo

A maioria de seus programas segue o modelo de ***entrada-processamento-saída***: os dados entram, são manipulados e, em seguida, são armazenados, exibidos, impressos ou transferidos.

O mecanismo básico de entrada no Python é baseado em linhas, quando lidos em seu programa a partir de um arquivo de texto, os dados chegam em uma linha de cada vez.

O **open ()** do Python existe para interagir com os arquivos. Quando combinados com uma instrução **for**, a leitura de arquivos é simples.

Crie uma pasta com um arquivo texto de nome *sketch.txt*

Inicie uma nova seção no IDLE, e importe o módulo **os** para mudar o diretório de trabalho atual para a pasta que contém seu arquivo de dados recém carregado.

Exemplo para área de trabalho:
os.chdir ('C:\\Users\Administrador\Desktop')

```
>>> import os
>>> os.getcwd ()
>>> os.chdir ('..\digite aqui o caminho da pasta que esta  seu arquivo')
>>> os.getcwd ()

>>> data = open('sketch.txt')
>>> print (data.readline(), end=' ')

>>> print (data.readline(), end=' ')
```

Lendo dados do arquivo

Vamos retroceder o arquivo para o inicio, em seguida, use uma instrução for para processar cada linha no arquivo:

```
>>> data.seek (0)        ← Use o método seek () para retornar o inicio do arquivo
>>> for cada_linha in data:
        print (cada_linha, end=' ')
```

Será impresso na tela o arquivo todo, após essa impressão, use o método close para fechar o arquivo.

```
>>> data.close ()        ← fechar o arquivo
```

Lendo dados do arquivo

Olhe atentamente para os dados. Eles parecem seguir um formato específico:

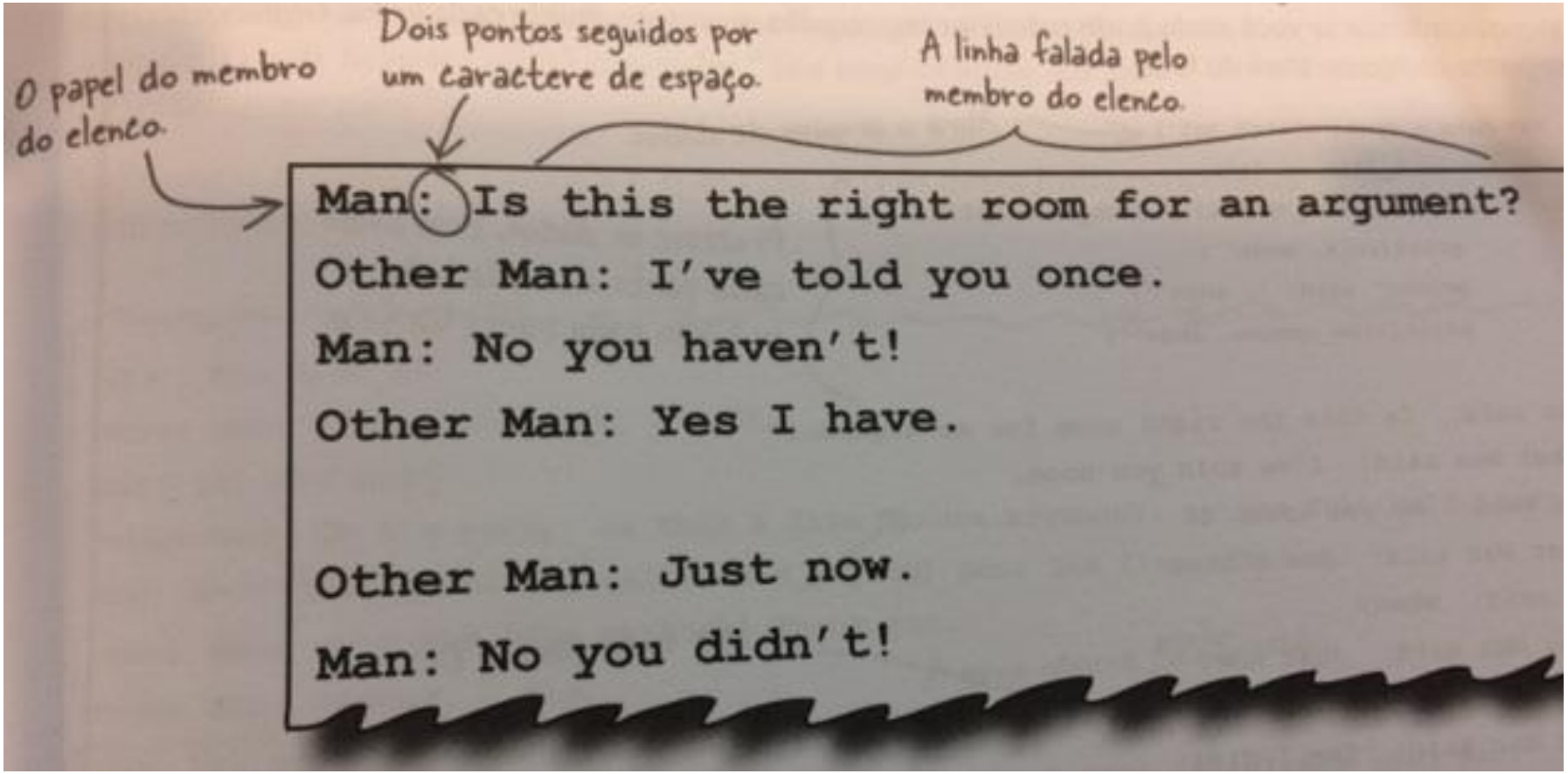


Figura 1 (2)

Olhe atentamente para os dados. Eles parecem seguir um formato específico:

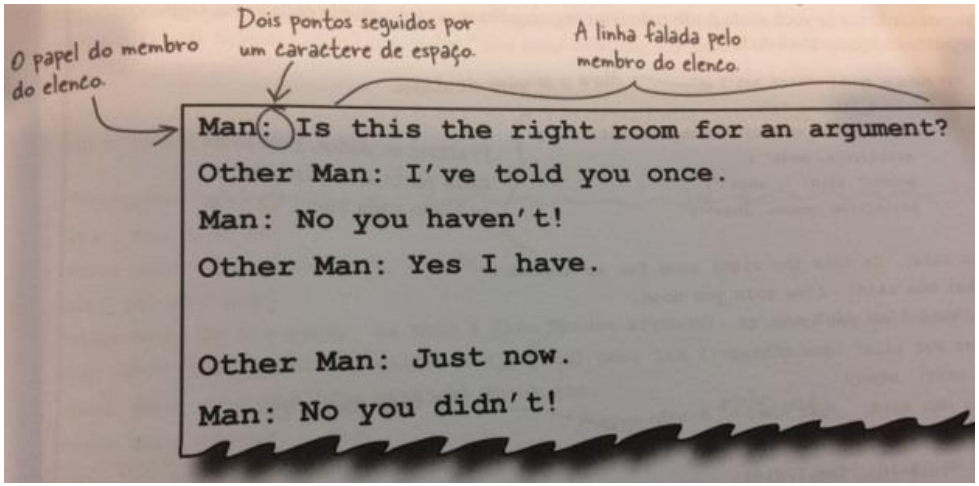
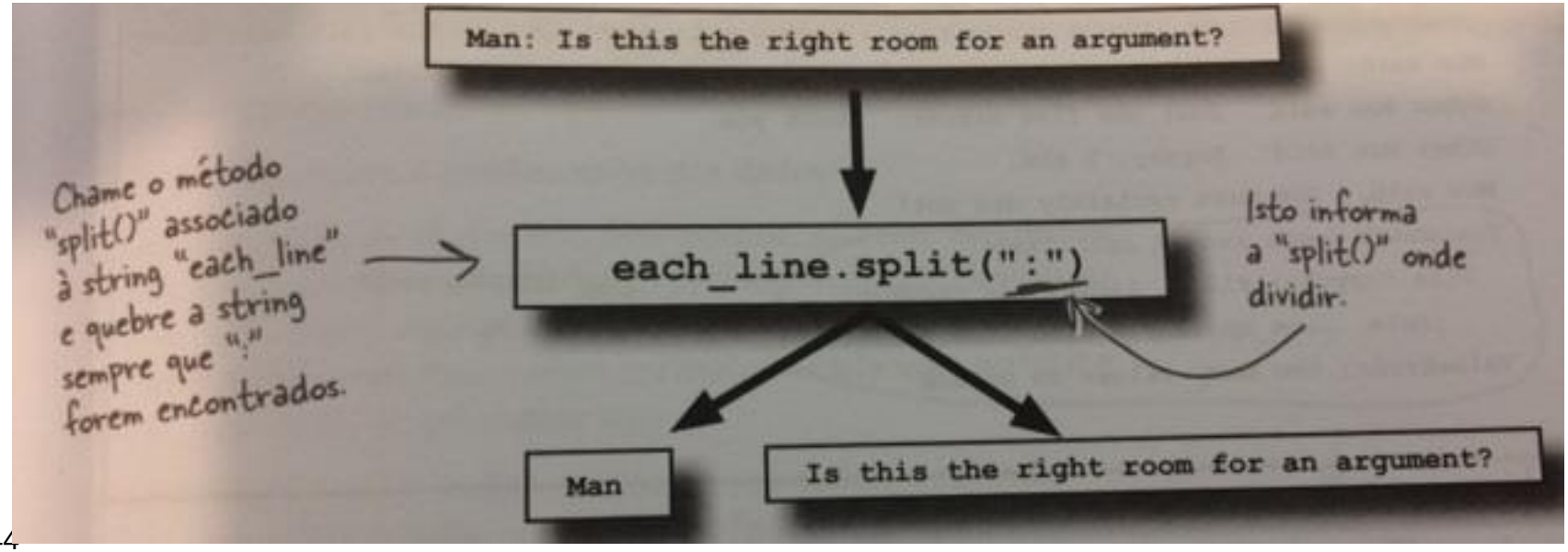


Figura 2 (2)

Com esse formato em mente, você pode processar cada linha para extrair partes dela, quando necessário. O método **Split ()** pode ajudar aqui:

Figura 3 (2)



Com esse formato em mente, você pode processar cada linha para extrair partes dela, quando necessário. O método **Split ()** pode ajudar aqui:

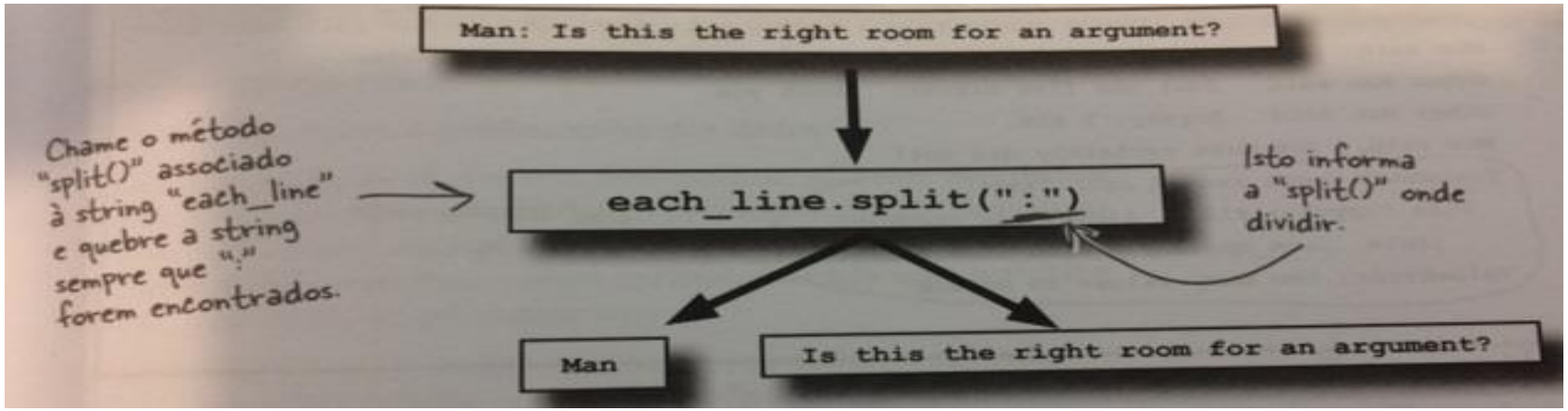


Figura 4 (2)

O método **Split ()** retorna uma lista de strings, que são atribuídas a uma lista de identificadores de destino. Isso é conhecido como atribuição múltipla.

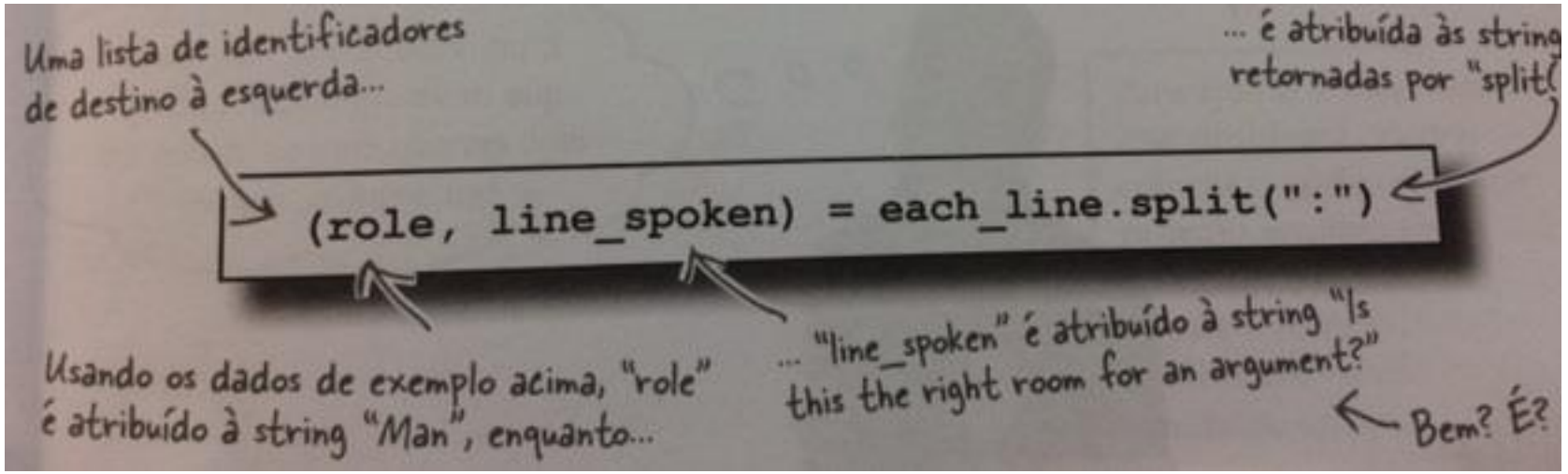
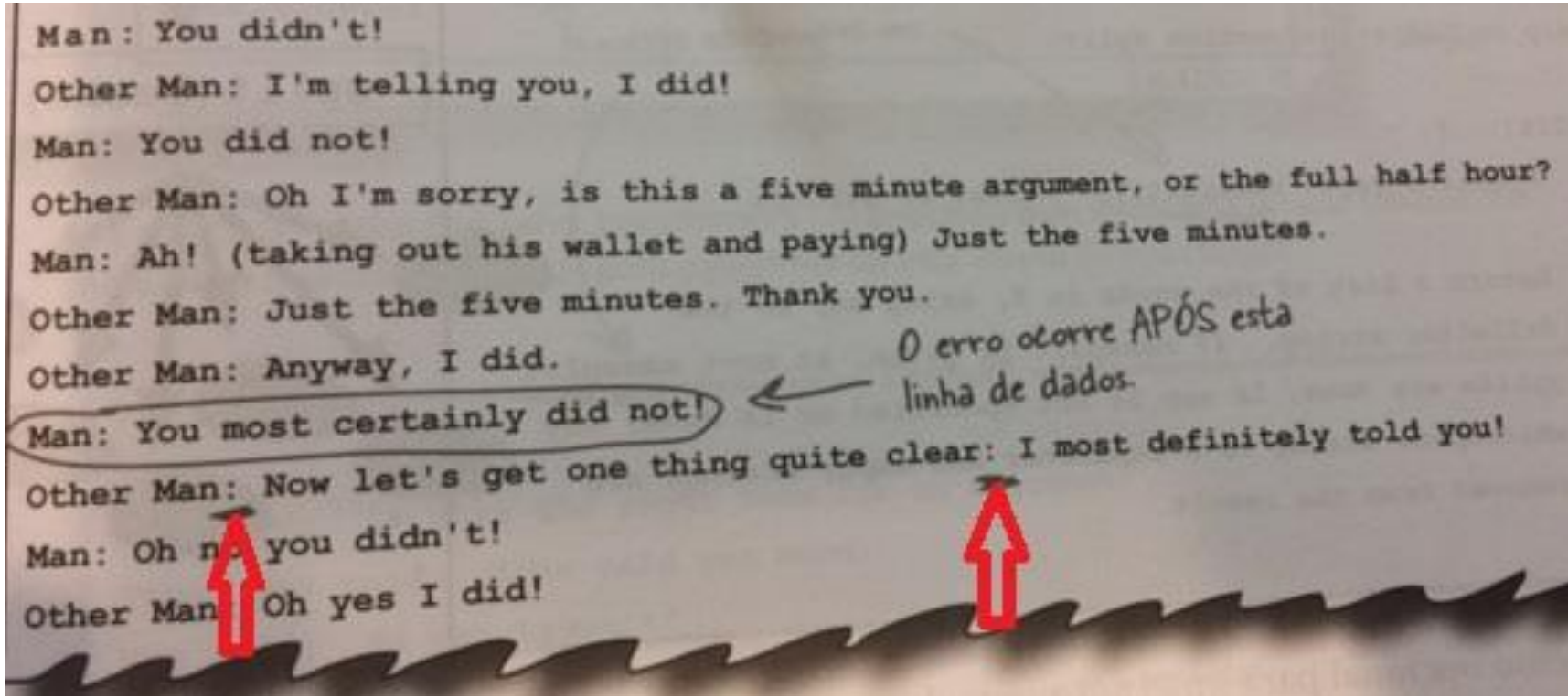


Figura 5 (2)

Usando Split ()

```
>>> data = open('sketch.txt')
>>> for cada_linha in data:
        (role, line_spoken) = cada_linha.split(':')
        print (role, end=' ')
        print (' disse: ', end=' ' )
        print (line_spoken, end=' ' )
```

Figura 6 (2)



Usando Split ()

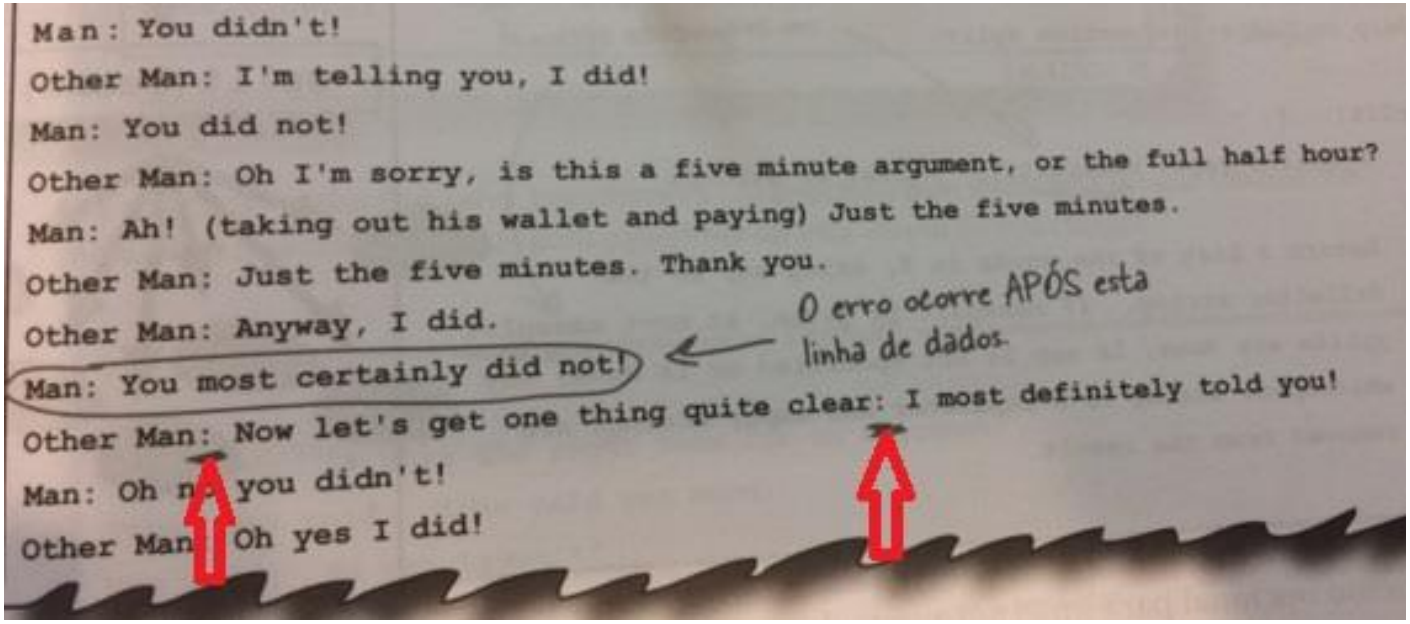


Figura 7 (2)

Percebe algo sobre a próxima linha de dados?

A próxima linha de dados tem dois pontos, não um. São dados extras suficientes para perturbar o método split () devido ao fato de que, como está seu código atualmente, split () espera quebrar a linha em duas partes, atribuindo cada uma a ***role*** e ***line_spoken***, respectivamente.

Quando dois pontos extras aparecem nos dados, o método split() quebra a linha em três partes. O código não disse a split() o que fazer com a terceira parte, então o interpretador Python gera um ValueError.

Lendo dados do arquivo

O argumento opcional para split() controla o número de quebras que ocorrem dentro de sua linha de dados.

Por padrão, os dados são divididos em tantas partes quanto possível. Mas, você só precisa de duas partes: o nome do personagem e a linha que ele falou.

Se for definido esse argumento opcional para **1**, a sua linha de dados só será quebrada em duas partes, efetivamente negando o efeito de qualquer dos dois pontos extras em qualquer linha.

Salve o código abaixo como sketch.py

```
import os
os.chdir ('..\digite aqui o caminho da pasta que esta  seu arquivo')

data = open('sketch.txt')
for  each_line in data:
            (role, line_spoken) = each_line.split(':', 1)
            print (role, end=' ')
            print (' disse: ', end=' '
            print (line_spoken, end=' ')

data.close()
```

Usando Split ()

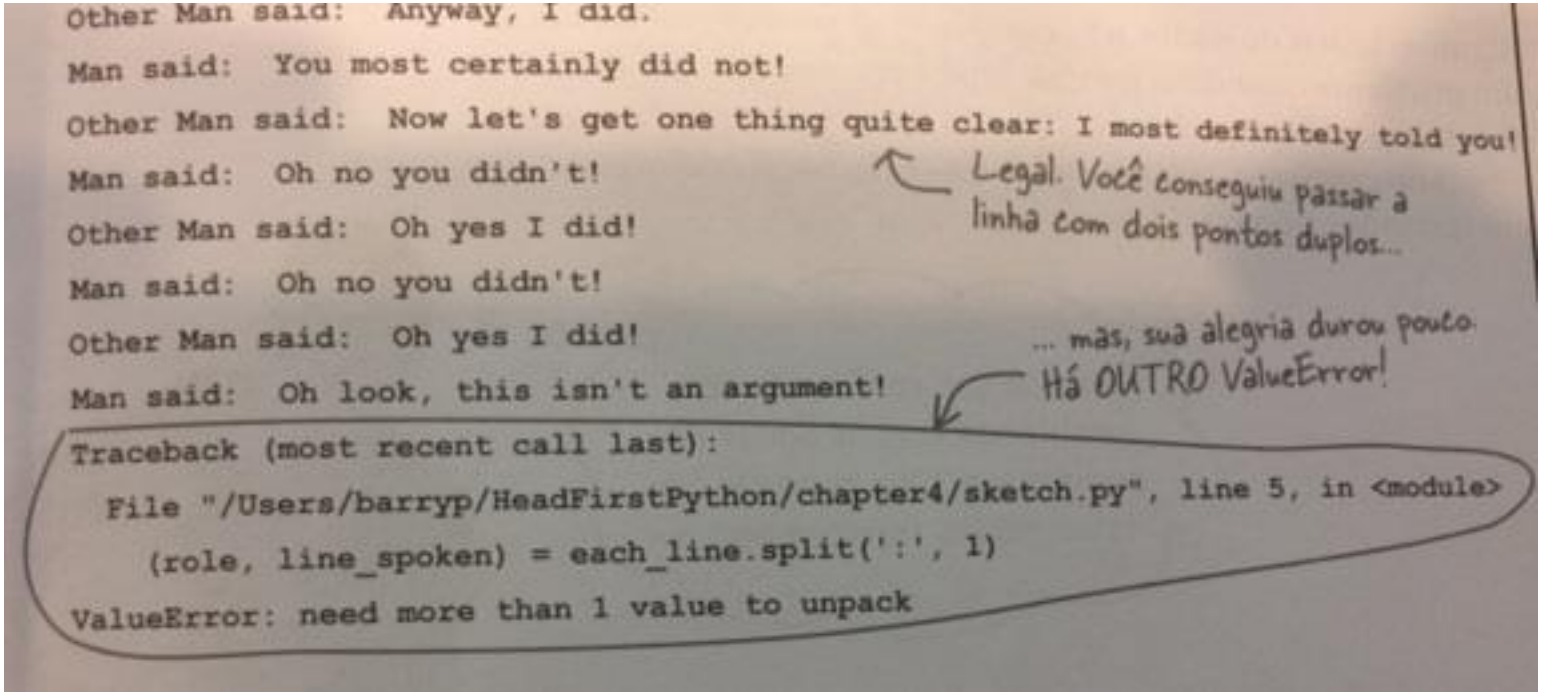


Figura 8 (2)

Seu código gerou outro ***ValueError***, mas desta vez, ao invés de reclamar que há muitos valores, o interpretador Python está dizendo que não tem dados suficientes para trabalhar: preciso de mais de 1 valor para descompactar.

Usando Split ()

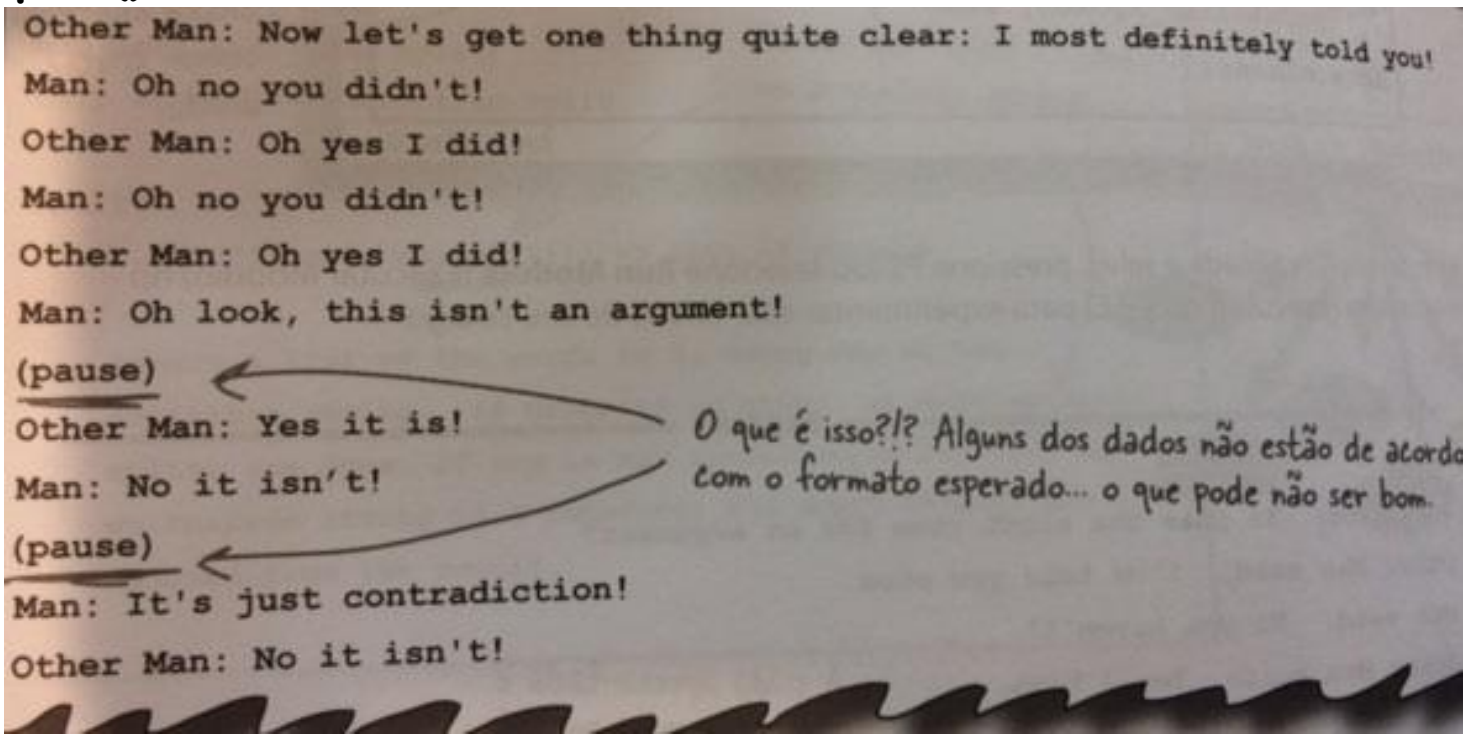


Figura 9 (2)

Algumas linhas de dados não contém dois pontos, o que causa um problema quando o método split() procura-os.

A falta dos dois pontos impede split () de fazer seu trabalho, causa o erro de execução.

Lendo dados do arquivo

Adicione uma lógica extra

Além do método split(), cada string Python tem também o método find () para tentar localizar uma substring em outra string e se não puder ser encontrada, o método find() retornará o valor -1.

Se o método localizar a substring, find() retornará a posição de índice da substing na string.

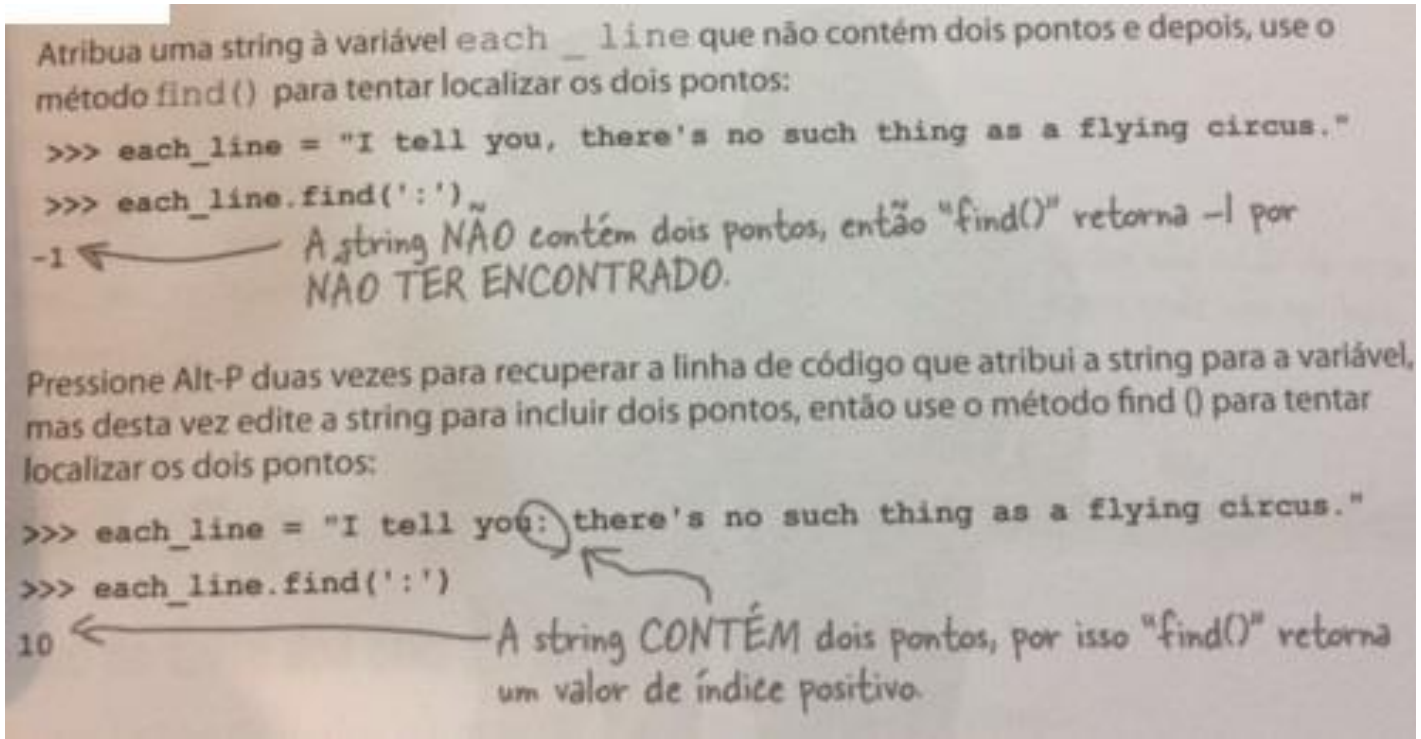


Figura 10 (2)

Lendo dados do arquivo

Adicione uma lógica extra

Ajuste o seu código para usar a técnica de lógica extra com o método find(), para lidar com as linhas que não contém o caractere de dois pontos.

```
import os
os.chdir ('..\digite aqui o caminho da pasta que esta  seu arquivo')

data = open('sketch.txt')
for  each_line in data:
            if not each_line.find (':') == -1:
                        (role, line_spoken) = each_line.split(':', 1)
                        print (role, end=' ')
                        print (' disse: ', end=' ')
                        print (line_spoken, end=' ')

data.close()
```

Lendo dados do arquivo

Agora seu programa está funcionando.

Se o formato do arquivo mudar, seu código terá que mudar também e mais códigos geralmente significa mais complexidade.

Ao invés de adicionar um código extra e lógica para impedir que coisas ruim aconteçam, o mecanismo de **tratamento de exceção** do Python permite que o erro ocorra, aponta o que aconteceu e depois, dá uma oportunidade para você se recuperar.

Ajuste seu código:

```
import os
os.chdir ('..\digite aqui o caminho da pasta que esta  seu arquivo')

data = open('sketch.txt')
for  each_line in data:
        try:
                (role, line_spoken) = each_line.split(':', 1)
                print (role, end=' ')
                print (' disse: ', end=' ')
                print (line_spoken, end=' ')
        except:
                pass

data.close()
```

Lendo dados do arquivo

Adicione mais código de verificação de erro...

Vamos implementar uma nova estratégia. O módulo **os** do Python tem alguns recursos que podem ajudar a determinar se um arquivo de dados existe, por isso precisamos importa-lo da biblioteca padrão, em seguida, adicionar a verificação requerida para o código:

```
import os

if os.path.exists ('sketch.txt'):
        data = open('sketch.txt')
        for  each_line in data:
                (role, line_spoken) = each_line.split(':', 1)
                print (role, end=' ')
                print (' disse: ', end=' ')
                print (line_spoken, end=' ')

        data.close()
else:
        print (' O arquivo de dados não esta na pasta')
```

Lendo dados do arquivo

Ou adicione outro nível de tratamento de exceção

```
try:
        data = open('sketch.txt')
        for  each_line in data:
                try:
                        (role, line_spoken) = each_line.split(':', 1)
                        print (role, end=' ')
                        print (' disse: ', end=' ')
                        print (line_spoken, end=' ')
                except:
                        pass

        data.close()
except:
        print (' O arquivo de dados não esta na pasta')
```

Salvando dados em arquivo

Criando uma lista vazia chamada **man**
Criando uma lista vazia chamada **other**
Adicionando uma linha de código para remover espaço em branco indesejável da variável line_spoken

```
man = []
other = []

try:
        data = open('sketch.txt')
        for  each_line in data:
                                try:
                                        (role, line_spoken) = each_line.split(':', 1)
                                        line_spoken = line_spoken.strip()
                                        if role == 'Man':
                                                man.append(line_spoken)
                                        elif role == 'Other Man':
                                                other.append(line_spoken)
                                except ValueError:
                                        pass

        data.close()
except IOError:
                print (' O arquivo de dados não esta na pasta')

print (man)
print (other)
```

O método **strip** remove espaço em branco de uma string

Abra seu arquivo no modo gravação

Quando você usa o open() para trabalhar com um arquivo do disco, pode especificar o **modo de acesso**. Por padrão, open() usa **r** como ***modo de leitura***, portanto você não precisa especificá-lo. Para abrir um arquivo para ***gravação***, uso o modo w.

```
man = []
other = []
try:
    data = open('sketch.txt')
    for each_line in data:
        try:
            (role, line_spoken) = each_line.split(':', 1)
            line_spoken = line_spoken.strip()
            if role == 'Man':
                man.append(line_spoken)
            elif role == 'Other Man':
                other.append(line_spoken)
        except ValueError:
            pass
    data.close()
except IOError:
    print (' O arquivo de dados não esta na pasta')

try:
    man_file = open('man_data.txt','w')
    other_file = open('other_data.txt','w')

    print (man, file = man_file)
    print (other, file = other_file)

    man_file.close()
    other_file.close()
except IOError:
    print ('File error')
```

Abrir o arquivo em modo gravação

print() para salvar as listas nomeadas nos arquivos de disco

Não esqueça de fechar ambos os arquivos.

Lide com uma exceção E/S, caso uma ocorra.

Abra seu arquivo no modo gravação

Quando tudo o que você faz é ler dados nos arquivos, receber um **IOError** é chato, mas raramente perigoso, pois os dados ainda estão em seu arquivo.

É uma história diferente ao **gravar os dados nos arquivos**: se você precisar lidar com **IOError** **antes** do arquivo ser fechado, os dados gravados poderão ser corrompidos.

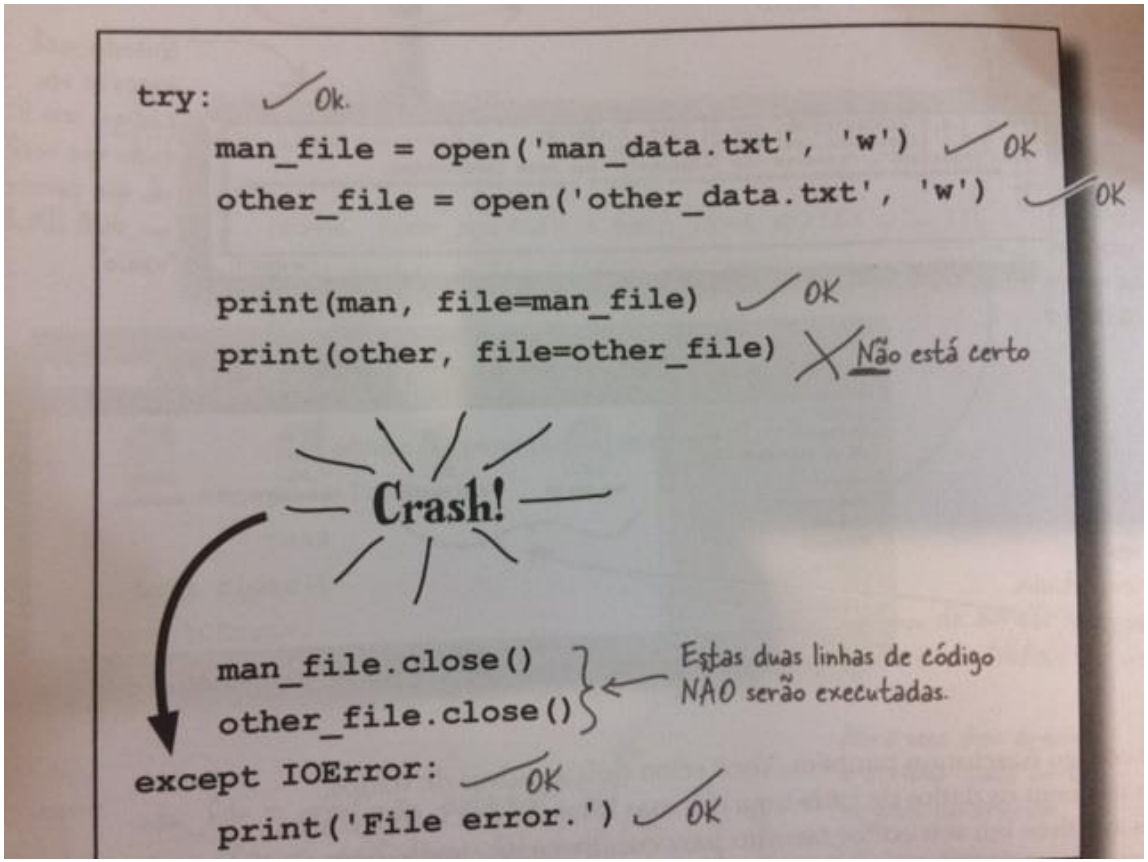


Figura 11 (2)

Quando se tem uma situação onde o código deve sempre ser executado não importando quais erros ocorram, pode-se adicionar o código abaixo ao conjunto **finally** de sua instrução **try**:

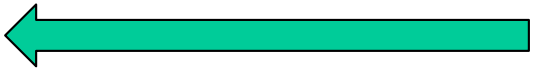
```
man = []
other = []
try:
    data = open('sketch.txt')
    for each_line in data:
        try:
            (role, line_spoken) = each_line.split(':', 1)
            line_spoken = line_spoken.strip()
            if role == 'Man':
                man.append(line_spoken)
            elif role == 'Other Man':
                other.append(line_spoken)
        except ValueError:
            pass
    data.close()
except IOError:
    print (' O arquivo de dados não esta na pasta')

try:
    man_file = open('man_data.txt','w')
    other_file = open('other_data.txt','w')

    print (man, file = man_file)
    print (other, file = other_file)

except IOError:
    print ('File error')
finally:
    man_file.close()
    other_file.close()
```

Abrir o arquivo em modo gravação



print() para salvar as listas nomeadas nos arquivos de disco

Lide com uma exceção E/S, caso uma ocorra.

Se não ocorrerem erros de execução, qualquer código no conjunto finally será executado
Igualmente, se um ***IOError*** ocorrer, o conjunto except será executado, e então o conjunto Finally
Aconteça o que acontecer, o código finally sempre será executado

Use with para trabalhar com arquivos

Uma vez que o uso padrão de **try / except / finally** é tão simples ao trabalhar com arquivos, o Python permite também trabalhar com with.

A instrução with, quando usada com arquivos, pode reduzir drasticamente a quantidade de código que você tem que escrever, pois evita a necessidade de incluir um conjunto finally para lidar com fechamento de arquivos de dados.

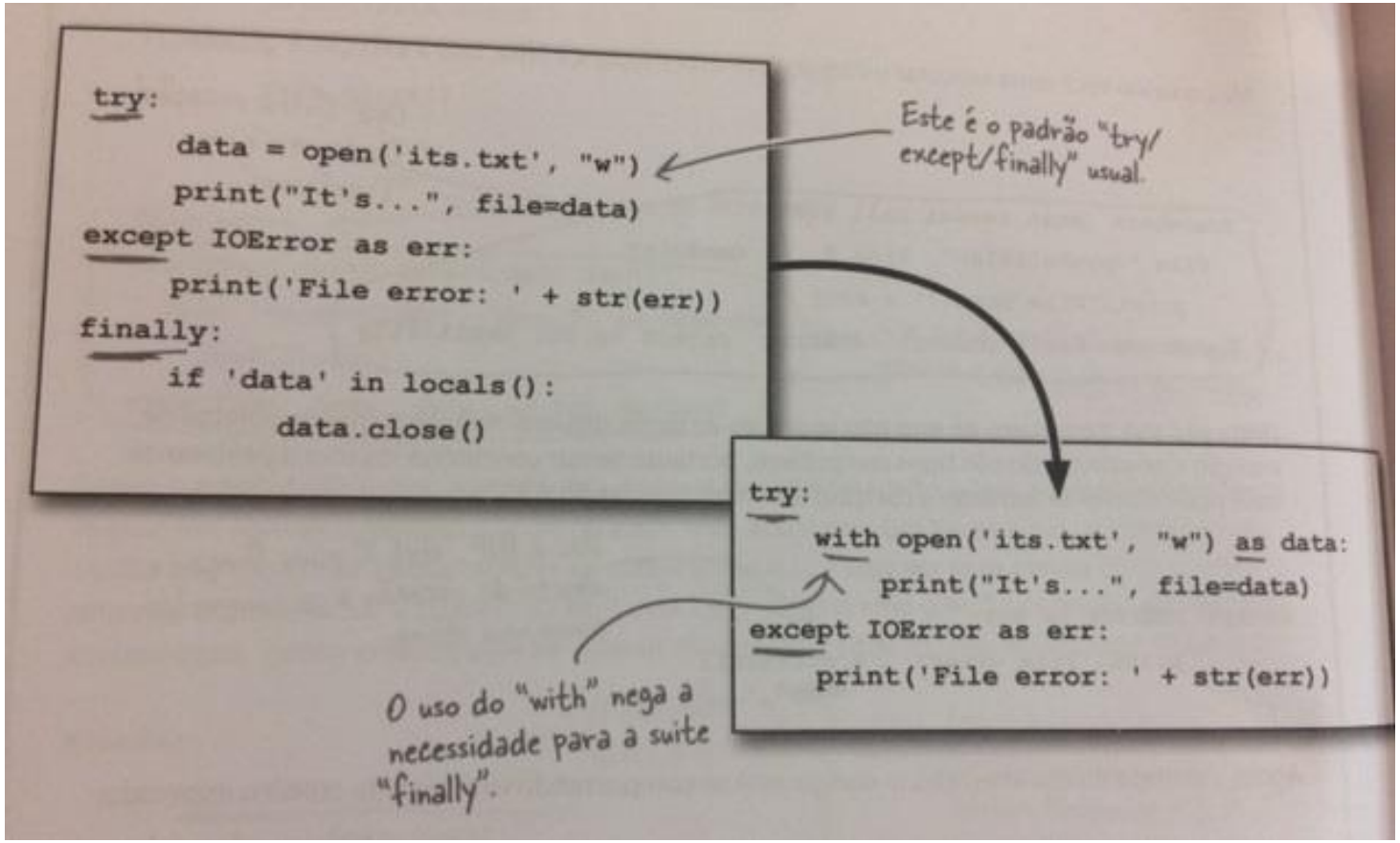


Figura 12 (2)

Quando se usa with, não precisa mais se preocupar em fechar os arquivos abertos, pois o interpretador Python automaticamente cuida disso.

```
man = []
other = []
try:
    data = open('sketch.txt')
    for each_line in data:
        try:
            (role, line_spoken) = each_line.split(':', 1)
            line_spoken = line_spoken.strip()
            if role == 'Man':
                man.append(line_spoken)
            elif role == 'Other Man':
                other.append(line_spoken)
        except ValueError:
            pass
    data.close()
except IOError:
    print (' O arquivo de dados não esta na pasta')

try:
    with open('man_data.txt','w') as man_file:
        print (man, file = man_file)
    with open('other_data.txt','w') as other_file:
        print (other, file = other_file)

except IOError:
    print ('File error')
```

Bibliografia

1) Administração de Redes com Scripts

Costa, Daniel Gouveia

Brasport

2) Python

Barry, Paul

Alta Books, 2012

3) Python3 – Conceitos e Aplicações

Banin, Sergio Luiz

Érica, 2018