Trabalho de Engenharia de Software

Vilson Magner Carrijo Santiago (11521EEL011)

27 de Abril de 2016

1 Introdução

Neste trabalho de Engenharia de Software, irei apresentar um tutorial sobre como criar novos arquivos e até mesmo organizá-los utilizando a linguagem de programação Python, com o objetivo de automatizar tarefas. Como base, utilizarei o Capítulo 9 do livro do Al Sweigart, "Automate the boring stuff with Python".

2 Capítulo 9: Organizando Arquivos

2.1 Mostrando extensão de arquivos

Com OS X e Linux, provavelmente seu navegador de arquivos mostrará as extensões dos arquivos (ex.: .pdf, .txt, .jpg). Já no Windows, por padrão, não mostrará, portanto, siga os seguintes passos para alterar:

(1) Vá em Iniciar
(2) Painel de Controle
(3) Aparência e Personalização
(4) Opções de Pasta
(5) Na aba de visão, abaixo da opção avançada, desmarque "Esconder extensões de tipos de arquivos conhecidos".

2.2 Copiando pastas e arquivos

Nesta parte do tutorial, iremos mostrar como copiar, mover, renomear e deletar arquivos utilizando o Python. Para utilizar as funções, será necessário o comando: import shutil.
O comando shutil.copy(fonte, destino) irá copiar o arquivo no caminho fonte para a pasta no destino caminho. Exemplo:

```
import shutil, os
os.chdir('C:\\')
```

shutil.copy('C:\\spam.txt', 'C:\\delicious')
'C:\\delicious\\spam.txt'
shutil.copy('eggs.txt', 'C:\\delicious\\eggs2.txt')
'C:\\delicious\\eggs2.txt'

A primeira chamada da função shutil.copy copia o arquivo C:\\spam.txt na pasta C:\\delicious e, na segunda chamada dessa função, realiza o mesmo processo, porém o arquivo eggs.txt terá um novo nome, eggs2.txt.

A segunda função shutil.copytree(fonte, destino) irá realizar quase o mesmo processo, porém em vez de ser um arquivo, será uma pasta inteira, sendo que dentro desta poderá ter mais pastas e, também, arquivos. Exemplo:

import shutil, os
os.chdir('C:\\')
shutil.copytree('C:\\bacon', 'C:\\baconbackup')

A pasta bacon será copiada para a pasta baconbackup e, caso baconbackup não existe, irá criar automaticamente.

2.3 Movendo e renomeando arquivos e pastas

A função shutil.move(origem, destino) irá mover um arquivo ou pasta para o destino, como em:

import shutil
shutil.move('C:\\bacon.txt', 'C:\\eggs')
Resultado: 'C:\\eggs\\bacon.txt'

É importante ressaltar que, caso o nome do arquivo ou pasta que você quiser mover já exista no destino desejado, será sobrescrito no arquivo ou pasta existente, portanto, é necessário tomar cuidado com a função shutil.move().

2.4 Deletando arquivos e pastas permanentemente

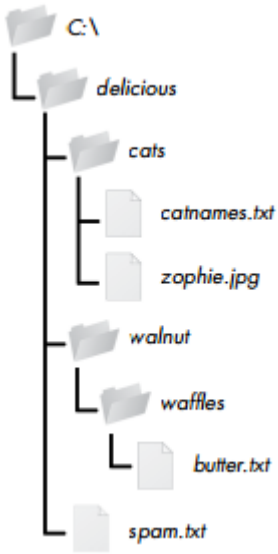
- Chamando os.unlink(path), deletará o arquivo no path
- Chamando os.rmdir(path), deletará a pasta no path
- Chamando shutil.rmtree(path), deletará a pasta no path com tudo que está dentro da mesma.

Exemplo de programa:
import os
for filename in os.listdir():
    if filename.endswith('.rxt'):
os.unlink(filename)

Neste programa, todos os arquivos que terminam com '.rxt' serão deletados, portanto, antes de utilizar as funções que deletam arquivos ou pastas, utilize o 'print(filename)' para mostrar tudo que está dentro de um destino, com o objetivo de não excluir algo importante.

2.5 Percorrendo diretórios

Digamos que você queira renomear cada arquivo em alguma pasta e também todos os arquivos em todas as subpastas dessa pasta. Ou seja, você quer andar pela árvore de diretórios, tocando cada arquivo que você vá. Escrever um programa para fazer isso poderia ficar complicado; felizmente, Python fornece uma função para lidar com este processo para você. Vamos olhar para o C:\delicious com o seu conteúdo:



Aqui está um exemplo de programa que usa a função os.walk () na árvore de diretórios:

```
import os
for pastaNome, subpastas, nomedosarquivos in os.walk('C:\\delicious'):
    print('A pasta atual é ' + pastaNome)
    for subpasta in subpastas:
        print('SUBPASTA DE ' + pastaNome + ': ' + subpasta)
    for nomedoarquivo in nomedosarquivos:
        print('Arquivo dentro de ' + pastaNome + ': '+ nomedoarquivo)
    print('')
```

2.6 Lendo arquivos .ZIP

Para ler arquivos do tipo .ZIP será necessário criar um objeto ZipFile. Objetos ZipFile são conceptualmente semelhantes aos objetos de arquivo que você viu

retornado pela função open () no capítulo anterior (8), ou seja, eles são valores através do qual o programa interage com o arquivo. Para criar um objeto ZipFile e realizar a leitura do arquivo .ZIP, siga as instruções do código abaixo:
import zipfile, os
os.chdir('C:\\') # move to the folder with example.zip
exampleZip = zipfile.ZipFile('example.zip')
exampleZip.namelist()
O que irá mostrar: ['spam.txt', 'cats/', 'cats/catnames.txt', 'cats/zophie.jpg']
spamInfo = exampleZip.getinfo('spam.txt')
spamInfo.file_size
O que irá mostrar: 13908
spamInfo.compress_size
O que irá mostrar: 3828
exampleZip.close()

O objeto ZipFile tem uma lista de nomes para listar todos arquivos dentro de um do tipo .ZIP, que se chama namelist(), utilizada no programa acima. Além disso, o getinfo() é um método para mostrar as informações do objeto ZipFile, por exemplo o tamanho dele (file_size), que também foi utilizado acima.

2.7 Extraindo arquivos .ZIP

Neste tópico, vamos aprender como extrair arquivos .ZIP, o que é bastante simples. Veja só o programa abaixo:

import zipfile, os
os.chdir('C:\\') # move to the folder with example.zip
exampleZip = zipfile.ZipFile('example.zip')
exampleZip.extractall()
exampleZip.close()

No programa acima, o conteúdo de example.zip será extraído para C:\\. Mas, também, podemos extrair somente um arquivo específico para outro local, como o exemplo que segue:

exampleZip.extract('spam.txt')
exampleZip.extract('spam.txt', 'C:\\some\\new\\folders')
exampleZip.close()

2.8 Criando arquivos .ZIP

Para criar arquivos .ZIP, o código é facilmente compreendido. Primeiramente abriremos o objeto ZipFile no modo de escrita 'w', passando o 'w' como segundo argumento, semelhante ao abrir um documento utilizando a função open(). No método de escrita, o Python realizará uma compressão do arquivo no destino e

adicioná-lo ao ZipFile, como no seguinte código:

```
import zipfile
newZip = zipfile.ZipFile('new.zip', 'w')
newZip.write('spam.txt', compress_type=zipfile.ZIP_DEFLATED)
newZip.close()
```

2.9 Projeto: renomear arquivos que possuem datas no estilo americano para datas no estilo europeu

Ensinaremos agora como trocar vários arquivos que possuam datas no estilo americano (MM-DD-YYYY) para arquivos com datas no estilo europeu (DD-MM-YYYY). Normalmente, essa tarefa pode ser bastante cansativa e entediante, porém, vamos aqui utilizar o Python para fazê-la. Com isso, nosso programa fará:

- Procurará todos arquivos que tenham o padrão de datas americano.
- Mudará todos esses para o padrão de datas europeu.

Passo 1:
Segue o programa abaixo. Atenção aos comentários realizados, pois eles são autoexplicativos.:

```
#! python3
# renameDates.py - Renomear arquivos de nomes com datas no formato americano MM-DD-YYYY
# para o europeu DD-MM-YYYY.
–recorte–
# Utilize o for sobre todos arquivos no diretório.
for amerFilename in os.listdir('.'):
    mo = datePattern.search(amerFilename)
    # Pular arquivos sem datas.
    if mo == None:
        continue
# Obter as diferentes partes do nome do arquivo.
beforePart = mo.group(1)
monthPart = mo.group(2)
dayPart = mo.group(4)
yearPart = mo.group(6)
afterPart = mo.group(8)
–recorte–
```

Passo 2: Nesta etapa, utilizaremos de conceitos já vistos anteriormente, como o shutil.move() para renomear arquivos. Também utilizaremos a concatenação de string com as variáveis feitas no passo anterior, com o objetivo de adequar ao estilo europeu de datas (DD-MM-YYYY). Segue abaixo o programa:

```
#! python3
```

```
# renameDates.py - Renomear nomes de arquivos com o formato de datas americano MM-DD-YYYY
# para o europeu DD-MM-YYYY.
–recorte–
# Formar o estilo europeu de nomes de arquivos.
euroFilename = beforePart + dayPart + '-' + monthPart + '-' + yearPart + afterPart
# Obter os caminhos dos arquivos por completo.
absWorkingDir = os.path.abspath('.')
amerFilename = os.path.join(absWorkingDir, amerFilename)
euroFilename = os.path.join(absWorkingDir, euroFilename)
# Renomeie os arquivos.
print('Renomeando "%s" to "%s"...' % (amerFilename, euroFilename))
#shutil.move(amerFilename, euroFilename) # Retire o comentário depois do teste.
```

2.10 Exercícios para praticar

Durante todo o tutorial, podemos aprender bastante como realizar tarefas corriqueiras e entediantes utilizando o Python como modo de automatizá-las. Realizamos um projeto no final importante sobre como mudar datas do estilo Americano para o estilo Europeu, porém, com o conceito dado nesse projeto, é possível realizar qualquer mudança de nome que contenham um certo padrão. Portanto, vamos agora propor aos leitores que realizem as questões abaixo:

1. Qual a diferença entre shutil.copy() e shutil.copytree()?
2. Qual função é usada para renomear arquivos?
3. Objetos ZipFile possuem o método close(), como os objetos arquivo possuem o método close(). Qual método ZipFile é equivalente ao método do objeto arquivo open()?
4. Qual função é utilizada para procurar arquivos que terminam com um certo formato? (Ex.: '.txt').
5. Qual cuidado, abordado nesse tutorial, devemos nos preocuparmos ao utilizar uma tarefa automatizada para renomear arquivos utilizando a função que é resposta do item (2)?

3 Finalizações do tutorial

Percebemos neste tutorial que a utilização do Python para realizar tarefas do cotidiano, as quais são extremamente entediantes, é um boa ideia. Com isso, no capítulo 9 do livro do Al Sweigart - "Automate the Boring stuff with Python" relatou essencialmente de organização e criação de arquivos, portanto, queremos mostrar, principalmente, a rapidez de desenvolvimente com essa linguagem de

programação e a facilidade em usá-la e, então, finalizamos por aqui nosso tutorial e esperamos que tenham compreendido os ensinamentos.

4 Referências Bibliográficas

SWEIGART, Al. **Automate the Boring Stuff with Python**. Disponível em: http://www.digitalbookocean.info/etext/12437-automate_the_boring_stuff_with_python_2015.pdf. Acesso em: 17/04/2016.