UNIVERSIDADE FEDERAL DE UBERLÂNDIA
Engenharia de Software

Capítulo 9, 10, 11 e 13
Organização de arquivos, web scraping, depuração e arquivos word e PDF

Alunos:

Luiz Fernando	11611EEL014
Guilherme Costa	11611EEL025
João Pedro Teixeira	11621EAU007
Aline Mendonça	11611ETE008

Professor: Marcelo Rodrigues

Uberlândia
2017

Conteúdo

1 Introdução

Este trabalho realizado tem como finalidade aplicar os conhecimentos obtidos em comunicação e manipulação de arquivos, como visto nos capítulos anteriores, em arquivos *word*, *PDF* e com processos na internet. Exibiremos as resoluções dos projetos práticos propostos pelo livro.

2 Organizando Arquivos

Aprendemos nos capítulos anteriores a criar e escrever em arquivos novos em python, porém essa tarefa pode ser muito repetitiva a exaustiva. Neste capítulo, temos como finalidade aumentar a capacidade de comunicação com outros tipos de arquivos, como *.pdf*, *.jpg*, *.txt* etc, além de automatizar tais trabalhos exaustivos. Introduziremos as soluções dos projetos práticos desse capítulo.[1]

2.1 Selective copy

O intuito deste exercício é criar um programa que procure, em uma pasta fornecida pelo usuário, arquivos com uma determinada extensão, também fornecida pelo usuário, e os copiem para outro local desejado, conforme o código na figura 1.

```
import shutil, os
pasta = input("Digite o diretorio de pesquisa: ")
destino = input("Digite o local para onde deseja copiar os arquivos: ")
extensao = input("Digite a extensao do arquivo desejado: ")
os.mkdir(destino+'\\New Folder')
count = 0
for filename in os.listdir(pasta):
    if filename.endswith(extensao):
        shutil.copy(pasta + '\\' + filename, str(destino)+ '\\New Folder')
        count+=1
if count == 0:
    os.rmdir(destino+'\\New Folder')
    print("Nao ha arquivos com essa extensao.")
```

Figura 1: Resolução do exercício *Selective copy*

Inicialmente o programa pede para que o usuário entre com o diretorio de pesquisa, o local de cópia e a extensão desejada. Em seguida uma pasta *New Folder* é criada no local de cópia. Todos os arquivos do diretório de pesquisa são verificados no laço *for* e se caso a extensão, que é verificada pela função *filename.endswith()*, for a correta, o arquivo é copiado para a *New Folder* por meio da função *shutil.copy()*. Se caso não houver arquivos com a extensão desejada, a pasta *New Folder* é excluida e uma mensagem impressa na tela.

2.2 Deleting Unneeded Files

O objetivo do exercício é criar um programa que verifique cada arquivo dentro de uma pasta cujo endereço é fornecido pelo usuário e, se caso o peso do arquivo for superior à 100MB, o mesmo será excluído. O código está ilustrado na figura 2.

[1] SWEIGART, 2015, p.213

```
import shutil, os
pasta = input("Digite o diretorio de pesquisa: ")
for filename in os.listdir(pasta):
    if os.path.getsize(pasta + '\\' + filename)>100000000:
        os.unlink(pasta + '\\' + filename)
    else:
        print("Nao existem arquivos com peso superior a 100MB")
```

Figura 2: Resolução do exercício *Deleting Unneeded Files*

Primeiramente o programa pede para que o usuário entre com o endereço da pasta na qual deseja realizar a busca. Em seguida, no laço *for*, o peso de cada arquivo é verificado pela função *os.path.getsize()*, e se caso superior a 100MB, a condição do *if* é atendida e o arquivo é excluido pela função *os.unlink()*, caso contrário é impresso na tela que não existem arquivos acima desse peso.

2.3 Filling in the gaps

O intuito do exercício é criar um programa que, quando fornecido o diretório de uma determinada pasta, o mesmo verifica se há arquivos numerados nesse local. Encontrados os arquivos, se caso houver alguma falha na numeração (ex: *Spam001.txt*, *Spam003.txt*, *Spam004.txt*), o programa renomeia todos os arquivos à frente dessa falha de forma a corrigí-la (transforma *Spam003.txt* em *Spam002.txt*, em seguida *Spam004.txt* em *Spam003.txt*). De forma sintetizada: Doc001, Doc003, Doc004 → Execução do código → Doc001, Doc002, Doc003.

Tais funcionalidades foram conseguidas por meio do uso de combinações de laços e iterações, expressões regulares e manipulação de arquivos com python, conforme ilustrado na figura 3.

```
import shutil, os, re
diretorio=input("Digite o diretório: ")
Regex = re.compile(r'^(.*?)(\d)(\d)(\d)(.*?)$')
for file in os.listdir(diretorio):
    mo = Regex.search(file)
    if mo == None:
        continue
    beforePart = mo.group(1)
    afterPart = mo.group(5)
    file = os.path.abspath(file)
    n1, n2, n3 = 0, 0, 1
    while True:
        if n3 == 10:
            n3 = 0
            n2 += 1
        elif n2 == 10:
            n2 = 0
            n1 += 1
        nome = beforePart + str(n1) + str(n2) + str(n3) + afterPart
        if not os.path.exists(diretorio+'\\'+nome):
            m1, m2, m3 = n1, n2, n3
            while True:
                if m3 == 10:
                    m3 = 0
                    m2 += 1
                elif m2 == 10:
                    m2 = 0
                    m1 += 1
                m3 += 1
                nome2 = beforePart + str(m1) + str(m2) + str(m3) + afterPart
                if not os.path.exists(diretorio + '\\' + nome2):
                    break
                velho = diretorio + '\\' + os.path.sep + nome2
                novo = diretorio + '\\' + os.path.sep + nome
                os.rename(velho, novo)
                print('Renomeando %s para %s: ' %(nome2, nome))
                nome = nome2
            break
        n3 += 1
```

Figura 3: Resolução do exercício *Filling in the gaps*

3 Depuração

Este capítulo aborda algumas ferramentas e técnicas para encontrar a causa raíz de bugs no código do programa e corrigí-los mais rapidamente e com menos esforço. Felizmente, existem formas para identicar o que exatamente está sendo realizado pelo código e onde está o erro. Em geral, quanto mais cedo forem encontrados os bugs, mais fácil será a correção. Mostraremos agora a resolução do projeto prático dessa seção.[2]

3.1 Debugging Coin Toss

O programa a seguir destina-se a ser um lance de moeda, simples jogo de adivinhação, onde o jogador recebe duas suposições. No entanto, o código possui vários erros. Devemos executar o programa algumas vezes para localizar os bugs que o impedem de funcionar corretamente.

```
>>> import random
guess = ''
while guess not in ('heads', 'tails'):
    print('Guess the coin toss! Enter heads or tails:')
    guess = input()
toss = random.randint(0, 1) # 0 is tails, 1 is heads
if toss == guess:
    print('You got it!')
else:
    print('Nope! Guess again!')
    guesss = input()
    if toss == guess:
       print('You got it!')
    else:
        print('Nope. You are really bad at this game.')
```

Figura 4: Programa do projeto com bugs *Debugging Coin Toss*

No programa sugerido pelo capítulo vemos uma incoerência ao notar que os parâmetros que são pedidos ao usuário é de tipo diferente do passado à função, sendo que o pedido é do tipo char e o passado é do tipo *int*. Devemos consertar o programa a fim de deixar os parâmetros com o mesmo tipo e tirar os bugs. O programa a seguir é o mesmo apresentado acima, mas com os erros já corrigidos e pronto para funcionar.

[2]SWEIGART, 2015, p.232

```
>>> import random
guess = ''
while guess not in (int(1), int(0)):
    print('Guess the coin toss! Enter 1 for heads or 0 for tails:')
    guess = int(input())
toss = random.randint(0, 1) # 0 is tails, 1 is heads
if toss == guess:
    print('You got it!')
else:
    print('Nope! Guess again!')
    guess = int(input())
    if toss == guess:
       print('You got it!')
    else:
        print('Nope. You are really bad at this game.')
```

Figura 5: Programa do projeto corrigido *Debugging Coin Toss*

4 Web Scraping

Neste capítulo, iremos usar a linguagem python para comunicar com a internet, conseguindo baixar processos online e realizando decisões conforme interpretamos tais processos. Algumas bibliotecas adicionais foram utilizadas, como a *webbrowser*, *Requests*, *Beautiful Soup* e *Selenium*. Segue abaixo as resoluções dos exercícios práticos. [3]

4.1 Image Site Downloader

O intuito deste exercício é criar um programa que dado um tema pelo usuário, o programa entre em um site de compartilhamento de imagens préviamente estabelecido , nesse caso o *Flickr*, e baixe todas as correspondências para o HD do usuário. O código é ilustrado na figura 6.

[3] SWEIGART, 2015, p.262

```
import requests, bs4
from selenium import webdriver
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

busca = input('O que deseja procurar? ')
url = ('https://www.flickr.com/search/?text=%s' % busca)

browser = webdriver.Firefox()
browser.get(url)
delay = 3
WebDriverWait(browser, delay).until(EC.presence_of_element_located(browser.find_element_by_id('

soup = bs4.BeautifulSoup(browser.page_source, "html.parser")


elements = soup.select('body #content main .main.search-photos-results \
                .view.photo-list-view.requiredToShowOnServer \
                .view.photo-list-photo-view.requiredToShowOnServer.awake \
                .interaction-view')
print(elements)
```

Figura 6: Resolução do exercício *Image Site Downloader*

Inicialmente o usuário digita o tema que está buscando. Em seguida é aberto uma aba no navegador que realiza a busca e baixa as correspondências, nesse programa utilizou-se um delay de 3s para a busca prevendo a possíveis problemas com a lentidão da internet.

Para saber se o código foi executado com sucesso, ao final da execução é mostrado na tela do usuário os links que estão sendo baixados.

4.2 Command Line Emailer

O intuito deste exercício é criar um programa que dado um email pelo usuário e um texto, o programa envie esse texto para o email fornecido. O código é ilustrado na figura 7.

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys

usuarioemail = input('Digite seu email ')
usuariosenha = input('Digite sua senha ')
emailpara = input('Digite o email do destinatario ')
assuntoemail = input('Digite o assunto do email ')
textoemail = input('Digite o corpo do email ')
browser = webdriver.Firefox()
type(browser)
browser.get('http://www.gmail.com')
emailElem = browser.find_element_by_id('Email')
emailElem.send_keys('', usuarioemail)
emailElem.submit()
senhaElem = browser.find_element_by_id('Password')
senhaElem.send_keys('', usuariosenha)
senhaElem.submit()
novoElem = browser.find_element_by_class_name('z0')
novoElem.click()
paraElem = browser.find_element_by_name("Para")
paraElem.send_keys('', emailpara)
assuntoElem = browser.find_element_by_name('Assunto')
assuntoElem.send_keys('', assuntoemail)
corpoemailElem = browser.find_element_by_css_selector("div[aria-label='Corpo da Mensagem']")
corpoemailElem.send_keys('', textoemail)
corpoemailElem.submit()
enviaElem = browser.find_element_by_xpath("//div[text()='Send']")
enviaElem.click()
```

Figura 7: Resolução do exercício *Command Line Emailer*

Inicialmente o usuário fornecerá uma serie de informações, como email do destinatário, seu email e etc. Após esses dados serem fornecidos e armazenados em variáveis, o programa acessa um site previamente definido, nesse caso o Gmail, e realiza login no email do usuário.

Realizado o acesso, o programa cria um novo email e preenche os dados que também foram fornecidos pelo usario, destinatário, assunto e corpo do email, com isso feito o programa envia o email.

4.3 2048

O intuito desse programa é simular o envio de teclas para o navegador por meio do selenium. O código é ilustrado na figura 8.

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys

browser = webdriver.Firefox()
type(browser)
browser.get('https://gabrielecirulli.github.io/2048/')
tecla = browser.find_element_by_class_name(game-container)
tecla.send_keys (Keys.UP)
tecla.send_keys (Keys.RIGHT)
tecla.send_keys (Keys.DOWN)
tecla.send_keys (Keys.LEFT)
tecla.submit()
```

Figura 8: Resolução do exercício *2048*

O código funciona acessando um site onde o jogo está disponível, com esse acesso realizado é enviado a seguinte ordem de teclas: Cima, Direita, Baixo, Esquerda, simulando um circulo. Essa simulação é possível devido ao modulo Keys presente no selenium, que permite simular o envio de teclas para o navegador.

5 Trabalhando com documentos em PDF e WORD

Arquivos em *word* (extensão mais usual: *.docx*) e em *PDF* (extensão *.pdf* são os tipos de arquivos mais usados no cotidiano. Aprender a manipular esses arquivos pode ser uma tarefa realmente complicada, pois a leitura desses arquivos são feitas em modo binário, o que compromete a acessibilidade. Para isso, usaremos algumas bilbiotecas como *PyPDF2* e *python-docx*. Mostraremos algumas resoluções dos problemas práticos propostos pelo livro.[4]

5.1 PDF Paranoia

Devemos criar um programa que acesse um diretório dado pelo usuário e que abra cada arquivo PDF e que o criptografe com uma senha informada pelo usuário e o adcione o sufixo *encripted.pdf*. O programa deve ser capaz de verificar as subpastas que existem no diretório. Tendo feito isso, devemos também criar um *script* que seja capaz de descriptografar um programa em PDF neste diretório com uma senha informada pelo usuário. Caso essa senha seja válida para algum arquivo PDF, o *script* deve copiar esse PDF descriptografado e deletar o criptografado, caso a senha seja incorreta, o *script* deverá mostrar uma mensagem na tela para o usuário tentar outra senha. Para melhor a compreensão, dividimos a resolução em duas partes: *Encryptor* (figura 9) e *Decrypter* (figura 10).

[4]SWEIGART, 2015, p.317

```
import PyPDF2, os
Dir = input('Insira o diretorio: ')
lista = os.walk(Dir)
senha = input("Insira a senha: ")
for folderName, subfolders, filenames in lista:
    for filename in filenames:
        diretorio = folderName + '\\' + filename
        if not diretorio.endswith('_encrypted.pdf'):
            arquivo_pdf = PyPDF2.PdfFileReader(open(diretorio, 'rb'))
            pdfEscritor = PyPDF2.PdfFileWriter()
            for num_paginas in range(arquivo_pdf.numPages):
                pdfEscritor.addPage(arquivo_pdf.getPage(num_paginas))
            pdfEscritor.encrypt(senha)
            pdfFinal = open(folderName + '\\' + filename + '_encrypted.pdf', 'wb')
            pdfEscritor.write(pdfFinal)
            pdfFinal.close()
```

Figura 9: Encryptor: *PDF Paranoia*

O *encryptor* tem como função básica colocar uma senha nos arquivos *PDF's* de um diretório informado. Para isso, usamos o laço *for* para percorrer todos esses arquivos e a função *if not .endswith('_encrypted.pdf')* irá verificar se o arquivo já não foi atribuído uma senha. Caso não, o *script* captura as páginas de um arquivo qualquer "*arquivo_pdf*"e copia para um arquivo *pdfFinal* através da função *pdfEscritor.addPage()*, dentro do *for*. Depois de copiado, atribuímos a senha na função *pdf.Escritor.encrypt()* e fechamos os arquivos.

```
import os, PyPDF2
Dir = input(r'Insira o diretorio: ')
lista = os.walk(Dir)
escritor_pdf = PyPDF2.PdfFileWriter()
senha = input('Insira a senha: ')
i = 0
for folderName, subfolders, filenames in lista:
    for filename in filenames:
        diretorio = folderName + '\\' + filename
        if diretorio.endswith('_encrypted.pdf'):
            arquivo_pdf = PyPDF2.PdfFileReader(open(diretorio, 'rb'))
            if arquivo_pdf.isEncrypted:
                if arquivo_pdf.decrypt(senha):
                    for num_paginas in range(arquivo_pdf.numPages):
                        pagina = arquivo_pdf.getPage(num_paginas)
                        escritor_pdf.addPage(pagina)
                    pdfcriado = open(folderName + '\\decrypted_' + filename, 'wb')
                    escritor_pdf.write(pdfcriado)
                    os.unlink(diretorio)
                    pdfcriado.close()
                    arquivo_pdf.close()
                    i = 1
        else:
            print('Diretório não encontrado')
            break
if i == 0:
    print("Senha incorreta!")
```

Figura 10: Decrypter: *PDF Paranoia*

A ideia do *decrypter* é semelhante ao *encryptor*. O que de fato muda é que temos um *else* para o *diretorio.endswith()*, com a intenção de verificar se o diretório foi encontrado. Usamos a função *arquivo_pdf.isEncrypted* para verificar se o "*arquivo_pdf*"possui uma senha e tentamos acessá-lo com a senha informada pelo usuário e em seguida, usamos a função *.decrypt()* para isso. Ela retorna 1 caso a senha for correta e 0 caso não for (por isso usamos uma variável *i*, a qual mostra esse raciocínio em "*if i == 0*"). Caso a senha for correta, copiaremos esse arquivo PDF em um outro e o adcionamos o sufixo "*_decrypted*"e apagamos o que estava com senha, conforme pedido no enunciado.

5.2 Custom Invitations as Word Documents

O problema inicial é criar um programa que, dado uma lista de convidados *guests.txt*, devemos criar uma lista de convidados em um arquivo *.docx*. A resolução se encontra na figura 11.

```
import docx
from docx.enum.text import WD_BREAK #Biblioteca para mudar de página
from docx.shared import Pt #Biblioteca para mudar a fonte
from docx.enum.text import WD_ALIGN_PARAGRAPH #Biblioteca para centralizar o parágrafo
arquivo = open('guests.txt', 'r')
convidados = arquivo.readlines()
documento = docx.Document()
i = 0
for guest in convidados:
    documento.add_paragraph('It would be a pleasure to have the company of').alignment = WD_ALIGN_PARA
    paragrafo = documento.add_paragraph(guest)
    paragrafo.add_run('at 11010 Memory Lane on the Evening of')
    paragrafo.alignment = WD_ALIGN_PARAGRAPH.CENTER
    documento.add_paragraph('May 16th').alignment = WD_ALIGN_PARAGRAPH.CENTER
    documento.add_paragraph('at 7 o clock').alignment = WD_ALIGN_PARAGRAPH.CENTER
    i = i + 3
    documento.paragraphs[i].runs[0].add_break(docx.enum.text.WD_BREAK.PAGE)
    i += 1
    style = documento.styles['Normal']
    font = style.font
    font.name = 'Brush Script MT'
    font.size = Pt(29)
documento.save('Convidados.docx')
```

Figura 11: Resolução do exercício *Custom Invitations as Word Documents*

Usamos a função *.alignment = WD_ALIGN_PARAGRAPH.CENTER* para alinharmos o parágrafos. O *i* é incrementado 4 a cada repetição do laço *for* pois sabemos que existem quatro parágrafos. Deve-se por atenção ao último trecho do laço *for* conforme mostrado na figura 12, pois é esta parte que altera o tamanho e o formato da fonte para *Brush Script MT*. Usamos a extensão *.font* para especificarmos qual fonte usaremos e *.size* para indicar o tamanho e, por fim, salvamos o arquivo como *Convidados.docx* na mesma pasta de diretório do programa.

```
style = documento.styles['Normal']
font = style.font
font.name = 'Brush Script MT'
font.size = Pt(29)
```

Figura 12: Trecho do código do *Custom Invitations as Word Documents*

Referências

6 Bibliografia

[1] SWEIGART, Al; *Automate the boring stuff with Python*, Practical Programming for Total Beginners, 2015.