

Expressões regulares, ler e escrever arquivos em Python

Alunos:

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

Professor: Marcelo Rodrigues

Conteúdo

1	Introdução	2
2	Expressões Regulares	2
2.1	Strong Password Detection	2
2.2	Regex Version of strip()	2
3	Leitura e escrita em arquivos	3
3.1	Mad Libs	3
3.2	Regex Search	4
4	Bibliografia	5

1 Introdução

O objetivo principal deste trabalho realizado é manipular arquivos, como escrever, ler e procurar padrões usando expressões regulares.

2 Expressões Regulares

Com a mesma ideia de procurar por textos ao apertar CTRL-F em navegadores e leitores de texto, as expressões regulares tem por finalidade encontrar padrões em palavras e localizá-las em um determinado texto. Neste capítulo, exibiremos resoluções dos projetos práticos¹

2.1 Strong Password Detection

O primeiro problema é escrever um programa que receba uma senha informada pelo usuário e o algoritmo deverá decidir se a senha é fraca ou forte, considerando que uma senha é forte se há pelo ou menos 8 dígitos, contendo ambas letras maiúsculas e minúsculas e no mínimo um número. É necessário testar a senha com vários outros padrões para validar a força.

```
1 import re
2 senha = (input("Digite a senha: "))
3 passRegex = re.compile(r'^(?=.*[A-Z])(?=.*[0-9])(?=.*[a-z]).{8,}$')
4 verificar = passRegex.search(senha)
5 print(verificar)
6 if (not verificar):
7     print("Senha fraca")
8 else:
9     print("Senha forte")
```

Figura 1: Resolução do exercício *Strong Password Detection*

Primeiramente temos uma busca, na senha digitada, por letras maiúsculas, números, letras minúsculas e a verificação se contém mais de 8 caracteres por meio do padrão escrito no `re.compile()` e pelo uso do `search()`, que retorna `match` quando todos os requisitos são encontrados ou então nulo quando pelo menos um dos requisitos não for encontrado. Em seguida, é impresso na tela o resultado.

2.2 Regex Version of strip()

O intuito deste exercício é criar um programa que, por meio de expressões regulares, exerça o papel da função `strip()`. Por conseguinte, temos que o programa deve retirar todos os dígitos desejados (sejam espaços em branco ou símbolos) das partes inicial e final da string. Caso esses dígitos estejam contidos no meio da string (entre a parte inicial e a parte final), de lá não devem ser removidos. Observamos tais características na figura 2.

O efeito desejado é conseguido pelo uso dos símbolos `^` e `$` que indicam que o padrão encontrado deve estar no início e no final da string, respectivamente. Os dígitos são verificados por meio do `[+ arg +]`. Ademais, após encontrados os padrões desejados, que no caso são os

¹SWEIGART, 2015, p.171

```

1  import re
2  string = input("Digite a string: ")
3  arg = input("Digite o argumento a ser retirado do início e do fim da string")
4  string2 = re.compile(r'(^['+ arg +']*)|(['+ arg +']*$)')
5  string2 = string2.sub('', string)
6  print("Strip:" + string2)

```

Figura 2: Resolução do exercício *Regex Version of strip()*

digitados pelo usuário, os mesmos são substituídos por nada através da função *regex sub()* e, em seguida, o resultado é impresso na tela.

3 Leitura e escrita em arquivos

Mesmo que queiramos salvar dados em variáveis, devemos nos atentar que elas apenas serão gravadas enquanto o programa estiver rodando. Entre outras palavras, o interesse deste capítulo é expor métodos de escrever, ou seja, armazenar as variáveis e assim como interpretá-las também. Com a mesma metodologia do capítulo anterior, exibiremos os algoritmos dos projetos práticos².

3.1 Mad Libs

Devemos criar um programa que leia palavras inseridas pelo usuário, tendo adjetivo, substantivo, advérbio e verbo. Este programa deverá incrementar essas palavras na frase *"The ADJECTIVE panda walked to the NOUN and then VERB. A nearby NOUN was unaffected by these events."* assim como também salvar essa frase em um arquivo *.txt*. Na figura 1, encontra-se uma possível resolução do problema em questão.

```
1  import shelve
2  shelf = shelve.open('Words')
3  lista = []
4  adj = input('Enter an adjective:')
5  lista.append(adj)
6  noun1 = input('Enter a noun:')
7  lista.append(noun1)
8  verb = input('Enter a verb:')
9  lista.append(verb)
10 noun2 = input('Enter with other noun:')
11 lista.append(noun2)
12 shelf['words'] = lista
13 print('The', shelf['words'][0], 'panda walked to the', shelf['words'][1],
14       'and then ', shelf['words'][2], '. A nearby ', shelf['words'][3],
15       ' was unaffected by these events.')
```

Figura 3: Resolução do exercício *Mad Libs*

O exercício pede para que inserimos:

adjective: silly

noun: chandelier

verb: screamed

noun: pickup truck

Atribuindo essas condições para as variáveis descritas, obtemos:

The silly panda walked to the chandelier and then screamed . A nearby pickup truck was unaffected by these events.

Notemos que a frase em questão foi imprimida na tela e foi salva no arquivo *"Words.txt"*. Neste algoritmo, usamos o *shelve* (arquivar, em inglês) para salvarmos as palavras inseridas pelo

²SWEIGART, 2015, p.194

usuário de forma que possamos usar elas na linha 13. A variável *"shelve"* pode ser interpretada com um dicionário com uma única chave ("words") e múltiplos valores (palavras digitadas). Quando usamos a *shelve*, criamos três arquivos novos com os nomes *words.bak*, *words.dir* e *words.dat*, os quais são arquivos binários que salvam as palavras como previamente dito³.

3.2 Regex Search

Neste problema, devemos escrever um programa que abra todos os arquivos de extensão *.txt* de um diretório e que procure por uma expressão regular informada pelo usuário e mostrá-la na tela. A figura 2 mostra uma possível solução.

```
1  import re
2  import os
3  try:
4      directory = 'C:\\Users\\Name_of_User\\Desktop\\' + input('Enter with your search: ')
5      folder = os.listdir(directory)
6      i = 0
7      search = input('Enter with your search: ')
8      for direc in folder:
9          if direc.endswith('.txt'):
10             file = open(directory + direc)
11             reg_exp = re.compile(r"{}".format(search), re.I)
12             mo = reg_exp.search(file.read())
13             if mo:
14                 print('Your search is on the file:', direc)
15                 i += 1
16             file.close()
17     if i == 0:
18         print('Search not found')
19 except:
20     print('The file does not exist! Try change the variable "directory" on the code\n')
```

Figura 4: Resolução do exercício *Regex Search*

O usuário em questão deverá modificar a variável *directory* para alterar o destino. Usamos a função *os.listdir()* para criar uma lista com todos os nomes do arquivo e a função *.endswith()* para comparar a extensão final do arquivo com *.txt* na linha 9. Feito isso, o arquivo será aberto no modo de leitura conforme a linha 10 e começará a pesquisa da expressão regular nas linhas 11 e 12. Se *mo* não for *None*, então mostraremos na tela o resultado e incrementaremos a variável *i*. Caso a pesquisa não for encontrada, o *i* será 0 e a pesquisa não foi encontrada. Caso houver uma falha na abertura do arquivo, o usuário será informado que houve um erro pela função *except*.

³Extensões assim não prejudicam a interpretação do que o *shelve* faz, pois foram planejados para facilitar o salvamento de informações, pois no contexto computacional, é mais fácil salvar em modo binário

Referências

4 Bibliografia

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