Desenvolvimento e Introdução do Grupo/1ª Parte

Grupo:

Carlos Alberto 11611ETE022

Douglas Almeida 11611EAU013

Laura Ribeiro 11611EAU015

Marcela Coury 11611EAU001

O professor Marcelo Rodrigues, da disciplina Engenharia de Software do 1º Semestre de 2017, propôs a leitura do livro "*Automate the Boring Stuff with Python*" e que, em primeiro momento, fosse estudado os primeiros 6 capítulos do livro e logo após a leitura, fosse escolhido dois capítulos do 7 ao 18. Foi decidido que o nosso grupo de 4 pessoas, sendo Carlos Alberto, Douglas Almeida, Laura Ribeiro e Marcela Coury; fosse divido em duplas para que essas, escolhessem um dos capítulos para ser realizado a leitura e a discussão sobre o que foi abordado, o tema, curiosidades, etc. Para isso, seria necessário aprender Python, e além do livro, utilizamos plataformas de aprendizagem, como o Codecademy, o sites Panda e PythonBrasil, dentre outros.

Cada dupla iria discutir entre si e depois passar as informações para a outra, para que assim, houvesse o conhecimento compartilhado e debatido. Além disso, o grupo se reuniu em torno de 5 vezes, nestas duas semanas (12/04 a 19/04; 19/04 a 25/04) sendo que nas duas últimas reuniões, foi direcionado para o uso do Github, Github Desktop e LaTex. Todos os passos do grupo foram colocados em quadros e listas do Trello, outra plataforma que o Professor sugeriu que utilizássemos.

Para o capítulo 7, *Pattern Matching with Regular Expressions*, a dupla Carlos e Marcela executou o desenvolvimento de alguns dos projetos e em resumo, explicaram funções e efetuaram comentários em linhas dos códigos.

- ➢ Biblioteca "re" - Import re

re.compile() cria um objeto regex

group() retorna a string atual (que achou), colocando números dentro e delimita os grupos; com o 0 ou sem nada, retorna a string inteira.

Exemplos:

```
import re
phoneNumRegex = re.compile(r'\d\d\d-\d\d\d-\d\d\d\d')
mo = phoneNumRegex.search('My number is 415-555-4242.')
print('Phone number found: ' + mo.group())
```

```
import re
phoneNumRegex = re.compile(r'(\d\d\d)-(\d\d\d-\d\d\d\d)')
mo = phoneNumRegex.search('My number is 415-555-4242.')
print(mo.group(1))      #1 grupo que esta entre parenteses
print(mo.group(2))      #2 grupo que esta entre parenteses
print(mo.group(0))      #grupo todo
print(mo.group())
```

```
mo.groups()                     #groups todos juntos
('415', '555-4242')
>>> areaCode, mainNumber = mo.groups()
>>> print(areaCode)
415
>>> print(mainNumber)
555-4242
```

```
>>> batRegex = re.compile(r'Bat(man|mobile|copter|bat)')
#todas começam com Bat
>>> mo = batRegex.search('Batmobile lost a wheel')
>>> mo.group()    #achou o correspondente e retornou o
ultimo
'Batmobile'
>>> mo.group(1)   #retornou o segundo nome
```

```
>>> batRegex = re.compile(r'Bat(wo)?man')  # ?opcional
aparte antes ()
>>> mo1 = batRegex.search('The Adventures of Batman')
>>> mo1.group()
'Batman'
>>> mo2 = batRegex.search('The Adventures of Batwoman')
>>> mo2.group()
'Batwoman'
```

```
>>> batRegex = re.compile(r'Bat(wo)*man')  # *repete
quantas vezes quiser ou não aparece nenhuma vez
```

```
>>> mo3 = batRegex.search('The Adventures of
Batwowowowoman')
>>> mo3.group()
'Batwowowowoman'
```

```
>>> batRegex = re.compile(r'Bat(wo)+man')   # + aparce pelo
menos 1 vez
>>> mo1 = batRegex.search('The Adventures of Batwoman')
>>> mo1.group()
'Batwoman'
>>> mo2 = batRegex.search('The Adventures of
Batwowowowoman')
>>> mo2.group()
'Batwowowowoman'

>>> mo3 = batRegex.search('The Adventures of Batman')
>>> mo3 == None
True
```

Repetições: coloca o numero de vezes entre {}, para inicio e fim especificado {_,_}, para inicio sem fim {_,} e para fim sem inicio {,_}

```
>>> haRegex = re.compile(r'(Ha){3}')
>>> mo1 = haRegex.search('HaHaHa')
>>> mo1.group()
'HaHaHa'
```

Repetição com ‘?’ : conta a menor possibilidade.

Exemplo:

```
nongreedyHaRegex = re.compile(r'(Ha){3,5}?') vai contar e
mostrar so HaHaHa (3 vezes).
```

- **findall()**

Vai retornar todos os que dão certo. Diferente do search() que retorna só o primeiro que da certo.

```
Search():>>> phoneNumRegex = re.compile(r'\d\d\d-\d\d\d-
\d\d\d\d')
```

```
>>> mo = phoneNumRegex.search('Cell: 415-555-9999 Work:
212-555-0000')
>>> mo.group()
'415-555-9999'
```

```
Findall():>>> phoneNumRegex = re.compile(r'\d\d\d-\d\d\d-
\d\d\d\d') # has no groups
>>> phoneNumRegex.findall('Cell: 415-555-9999 Work:
212-555-0000')
['415-555-9999', '212-555-0000']
```

Table 7-1: Shorthand Codes for Common Character Classes

Shorthand character class	Represents
\d	Any numeric digit from 0 to 9.
\D	Any character that is *not* a numeric digit from 0 to 9.
\w	Any letter, numeric digit, or the underscore character. (Think of this as matching "word" characters.)
\W	Any character that is *not* a letter, numeric digit, or the underscore character.
\s	Any space, tab, or newline character. (Think of this as matching "space" characters.)
\S	Any character that is *not* a space, tab, or newline.

```
>>> xmasRegex = re.compile(r'\d+\s\w+')
>>> xmasRegex.findall('12 drummers, 11 pipers, 10 lords, 9
ladies, 8 maids, 7
swans, 6 geese, 5 rings, 4 birds, 3 hens, 2 doves, 1
partridge')
['12 drummers', '11 pipers', '10 lords', '9 ladies', '8
maids', '7 swans', '6
geese', '5 rings', '4 birds', '3 hens', '2 doves', '1
partridge']
```

Obs: \d+ = 1 ou mais números; \s = espaço; \w+ = 1 ou mais letras.

Pode-se criar também uma classe, colocando entre []. Exemplo: r'[aeiouAEIOU]'.

Se quiser que apareça os que não foram especificados no [], é só colocar ^ antes das letras. Ex: r'[^aeiouAEIOU]'.

```
. = retorna a letra antes das outras letras que vierem
depois do ponto. Exemplo:
>>> atRegex = re.compile(r'.at')
>>> atRegex.findall('The cat in the hat sat on the flat
mat.')
['cat', 'hat', 'sat', 'lat', 'mat']
```

.* = retorna o que vem depois, exceto se passar de linha.
Exemplo:

```
>>> nameRegex = re.compile(r'First Name: (.*) Last Name: (.*)')
>>> mo = nameRegex.search('First Name: Al Last Name: Sweigart')
>>> mo.group(1)
'Al'
>>> mo.group(2)
'Sweigart'
```

('.*', re.DOTALL)= retorna tudo, inclusive se tiver nas outras linhas. Exemplo:

```
>>> newlineRegex = re.compile('.*', re.DOTALL)
>>> newlineRegex.search('Serve the public trust.\nProtect the innocent.
\nUphold the law.').group()
'Serve the public trust.\nProtect the innocent.\nUphold the law.'
```

Para ignorar se as letras estão em maiúsculo ou minúsculo, basta colocar como segundo argumento re.IGNORECASE ou re.I

Exemplo:

```
>>> robocop = re.compile(r'robocop', re.I) #se colocar ROBOCOP ou RoBoCop, ... da certo.
```

- The ? matches zero or one of the preceding group.
- The * matches zero or more of the preceding group.
- The + matches one or more of the preceding group.
- The {n} matches exactly *n* of the preceding group.
- The {n,} matches *n* or more of the preceding group.
- The {,m} matches 0 to *m* of the preceding group.
- The {n,m} matches at least *n* and at most *m* of the preceding group.
- {n,m}? or *? or +? performs a nongreedy match of the preceding group.
- ^spam means the string must begin with *spam*.
- spam$ means the string must end with *spam*.
- The . matches any character, except newline characters.
- \d, \w, and \s match a digit, word, or space character, respectively.
- \D, \W, and \S match anything except a digit, word, or space character, respectively.
- [abc] matches any character between the brackets (such as *a*, *b*, or *c*).
- [^abc] matches any character that isn't between the brackets.

Substituição: substitui o que vem depois da palavra

```
>>> namesRegex = re.compile(r'Agent \w+')
>>> namesRegex.sub('CENSORED', 'Agent Alice gave the secret
documents to Agent Bob.')
'CENSORED gave the secret documents to CENSORED.'
```

```
>>> agentNamesRegex = re.compile(r'Agent (\w)\w*')
>>> agentNamesRegex.sub(r'\1****', 'Agent Alice told Agent
Carol that Agent
Eve knew Agent Bob was a double agent.')
A**** told C**** that E**** knew B**** was a double agent.'
```

Expressões regulares mais longas

Utilizano re.VERBOSE como segundo argumento, ele compila sem contar os comentários.

```
phoneRegex = re.compile(r'''(
(\d{3}|\(\d{3}\))? # area code
(\s|-|\.)? # separator
\d{3} # first 3 digits
(\s|-|\.) # separator
\d{4} # last 4 digits
(\s*(ext|x|ext.)\s*\d{2,5})? # extension
)''', re.VERBOSE)
```

Para combinar as expressões de ignorar o maiúsculo, minúsculo, pegar as linhas... utiliza o | .

```
>>> someRegexValue = re.compile('foo', re.IGNORECASE |
re.DOTALL | re.VERBOSE)
```

Projeto 1:

Strong Password Detection
Write a function that uses regular expressions to make sure the password string it is passed is strong. A strong password is defined as one that is at least eight characters long, contains both uppercase and lowercase characters, and has at least one digit. You may need to test the string against multiple regex patterns to validate its strength.

```
import re

password = re.compile(r'(\w){8}')
senha = input('Digite uma senha(8 digitos): ')
if len(senha) == 8:
    if re.search(r'[a-z]',senha):                #Verifica se
tem letras minúsculas
        if re.search(r'[A-Z]',senha):            #Verifica se
tem letras maiúsculas
            if re.search(r'[0-9]',senha):      #Verifica se
tem números
                print('Senha forte: ',senha)
            else:
                print('Senha fraca')
        else:
            print('Senha fraca')
    else:
        print('Senha fraca')
else:
    if len(senha) < 8:
        print('Senha curta')
    else:
        print('Senha longa')
```

Projeto 2:

Regex Version of strip()
Write a function that takes a string and does the same thing as the strip()string method. If no other arguments are passed other than the string to strip, then whitespace characters will be removed from the beginning and end of the string. Otherwise, the characters specified in the second argument to the function will be removed from the string.

```
import re

frase = input('Escreva uma palavra: ')
digito = input('Remover: ')
x = re.compile(r'^['+digito+']*')
nova_frase = x.sub('',frase)
y = re.compile(r'(['+digito+']$)')
string = y.sub('',frase)
print('Frase sem a letra no comeco(se tiver): ',nova_frase)
print('Frase sem a letra no final(se tiver): ',string)
```

O capítulo 8, *Reading and Writing Files,* foi escolhido pela dupla Douglas e Laura; em que foi realizado o desenvolvimento de resumos e palavras-chaves do capítulo, além da realização de alguns projetos e definição de certas funções.

O tema abordado, de fato, é a maneira de como criar, armazenar, ler, escrever, deletar/apagar, arquivos utilizando o Python.

Um arquivo tem duas propriedades “chaves”:

- *Filename*: geralmente escrito como uma única palavra, informando o nome do projeto e o tipo.

  Exemplo: projects.docx

- *Path*: específica a localização de um arquivo no computador, o endereço do arquivo, o “caminho”.

Como exemplo, temos: C:\Users\asweigart\Documents.

Obs.: No Windows, os caminhos são escritos usando barras invertidas (\) como o separador, entre os nomes de pastas. OS X e Linux, no entanto, use a barra (/).

No capítulo 8 foram reutilizados os conceitos aprendidos no capítulo anterior com a adição de técnicas de manipulação de arquivo. Observa-se que há duas maneiras de endereçar um arquivo:

1) Por caminho absoluto (*absolute path*) onde o caminho para o arquivo sempre começa a partir do diretório raíz;

2) por caminho relativo (*relative path*) em que o caminho para o arquivo é especificado a partir do diretório a partir do qual o programa é executado.

Figure 8-2 is an example of some folders and files. When the current working directory is set to *C:\bacon*, the relative paths for the other folders and files are set as they are in the figure.

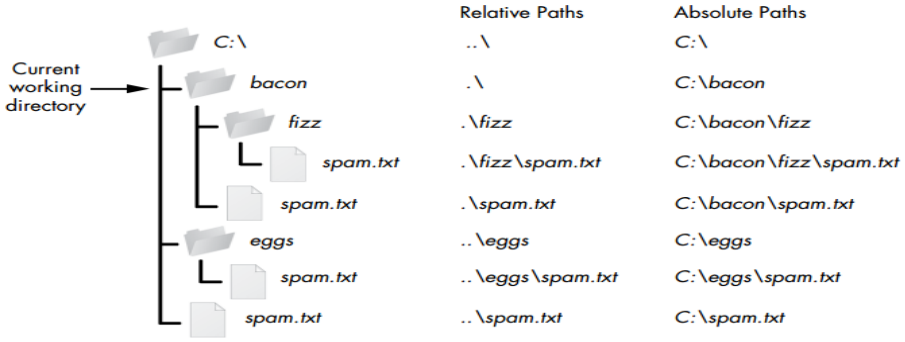

Figure 8-2: The relative paths for folders and files in the working directory C:\bacon

➢ *The File Reading/Writing Process*

Uma das três maneiras de ler ou escrever arquivos em Python, é a partir das seguintes funções:

1. Chama-se **open()** a função que abre um arquivo;

2. Chama-se **read()** ou **write()** para ler ou escrever no Arquivo;

3. **Close** para fechar o arquivo que foi chamado em open() (todo arquivo aberto deve ser fechado).

Nome de Método	Uso	Explicação
`open`	`open(nome_arquivo,'r')`	Abre um arquivo chamado nome_arquivo e o usa para leitura. Retorna uma referëncia para um objeto *file*.
`open`	`open(nome_arquivo,'w')`	Abre um arquivo chamado nome_arquivo e o usa para escrita. Retorna uma referëncia para um objeto *file*.
`close`	`ref_arquivo.close()`	Utilização do arquivo referenciado pela variável ref_arquivo terminou.

Tabela retirada da versão do livro online: Como pensar como um Cientista da Computação.