ARA0168
TÓPICOS DE BIG DATA EM PYTHON

Aula 3 – Ecossistema *Hadoop*: Parte II

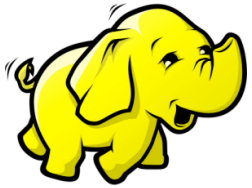
Universidade Estácio de Sá

Prof. Simone Gama

simone.gama@estacio.br

They say an elephant never forgets. Well, you are not an elephant. Take notes, constantly. Save interesting thoughts, quotations, films, technologies… the medium doesn't matter, so long as it inspires you.

Aaron Koblin



Ecossistema *Hadoop* : MapReduce

MapReduce (Mapear e Reduzir)

Em um modelo de programação, o *MapReduce* pode ser definido por três fases principais, a saber:

1. **Mapeamento** (*Map*)
2. **Embaralhar e Classificar** (*Shuffle and Sort*)
3. **Reduzir** (*Reduce*)

Estácio

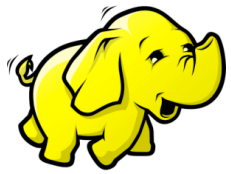
ARA0168
TÓPICOS DE BIG DATA EM PYTHON

3.1 – Ferramentas Python para Algoritmos em Big Data: *Collections*

Universidade Estácio de Sá

Prof. Simone Gama

simone.gama@estacio.br

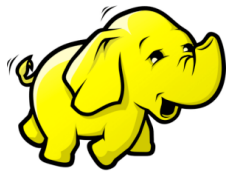


Ecossistema *Hadoop*: Ferramentas Python

Packages Collections

O módulo de coleções (***collections***)[1] em Python fornece diferentes tipos de contêineres. Um **Container** é um objeto usado para armazenar objetos diferentes e fornecer uma maneira de acessar os objetos contidos e iterar sobre eles. Alguns dos contêineres *built-in* são *Tuple*, *List*, *Dictionary*, e *Sets*.

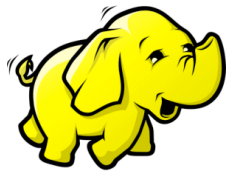
[1]Fonte: collections — Container datatypes — Python 3.10.6 documentation



Ecossistema *Hadoop*: Ferramentas Python

Packages Collections

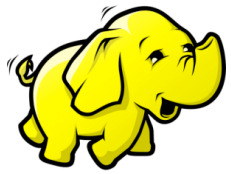
namedtuple()	**factory function** for creating tuple subclasses with named fields
deque	**list-like** container with fast appends and pops on either end
ChainMap	**dict-like** class for creating a single view of multiple mappings
Counter	**dict** subclass for **counting** hashable objects
OrderedDict	**dict** subclass that remembers the order entries were added



Ecossistema *Hadoop*: Ferramentas Python

Packages Collections

defaultdict	**dict** subclass that calls a factory function to supply missing values
UserDict	wrapper around **dictionary** objects for easier dict subclassing
UserList	wrapper around **list** objects for easier list subclassing
UserString	wrapper around **string** objects for easier string subclassing



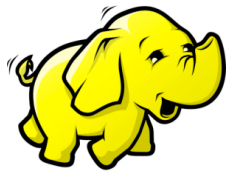
Ecossistema *Hadoop*: Ferramentas Python

Packages Collections: Counter

Um **Counter** é uma subclasse do dicionário. Ele é usado para manter a contagem dos elementos em um iterável na forma de um dicionário não ordenado onde a chave representa o elemento no iterável e o valor representa a contagem desse elemento no iterável.

Sintaxe:

```
class collections.Counter([iterable-or-mapping])
```



Ecossistema *Hadoop*: Ferramentas Python

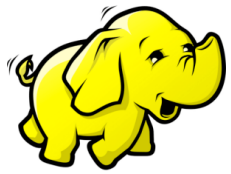
Packages Collections: Counter

Example 1:

```
palavra = "mississipi"
contador = { }

for i in palavra:
  if not contador.get(i):
    contador[i] = palavra.count(i)

print(contador)
```



Ecossistema *Hadoop*: Ferramentas Python

Packages Collections: Counter

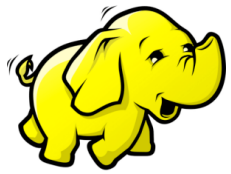
Example 2:

```
from collections import Counter

print(Counter("mississippi"))
```

Result:

```
Counter({'i': 4, 's': 4, 'p': 2, 'm': 1})
```



Ecossistema *Hadoop*: Ferramentas Python

Packages Collections: Counter

Example 3:

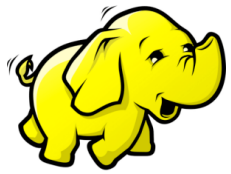
```
from collections import Counter

itens = Counter("mississippi")
itens = Counter(itens)
print(itens)

itens.update("missouri")
print(itens)
```

Result:

```
Counter({'i': 4, 's': 4, 'p': 2, 'm': 1})
Counter({'i': 6, 's': 6, 'm': 2, 'p': 2, 'o': 1, 'u': 1, 'r': 1})
```



Ecossistema *Hadoop*: Ferramentas Python

Packages Collections: Counter

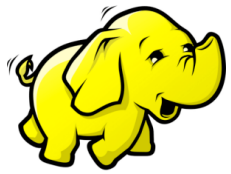
Example *Counter*:

```
from collections import Counter

# With sequence of items
print(Counter(['B','B','A','B','C','A','B',
               'B','A','C']))

# with dictionary
print(Counter({'A':3, 'B':5, 'C':2}))

# with keyword arguments
print(Counter(A = 3, B = 5, C = 2))
```



Ecossistema *Hadoop*: Ferramentas Python

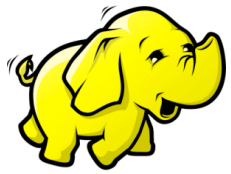
Packages Collections: Counter

Example *Counter*:

```
from collections import Counter


lista = Counter(['B','B','A','B','C','A','B',
                 'B','A','C'])

frequente = lista.most_common(2)
print("Elemento + comum ", frequente)
```



Ecossistema *Hadoop*: Ferramentas Python

Packages Collections: Counter

Pratice *Counter*:

1. Deseja-se contar a ocorrência de palavras de arquivos. Leia os arquivos "`contar1.txt`" e "`contar2.txt`" faça a contagem de palavras, somando as contagens das chaves em comum. Exemplo:

```
{"Maria": 2, "João": 6,
"Juca": 4, "Fabio": 10}
```

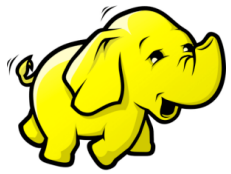
contar1.txt

```
{"Maria":   7,   "João":   4,
"Larissa":  23,  "Fabio":  0,
"Raimundo": 11}
```

contar2.txt

```
{'Larissa': 23, 'Raimundo':
11, 'João': 10, 'Fabio': 10,
'Maria': 9, 'Juca': 4}
```

Dicionário
Contador_Reduce



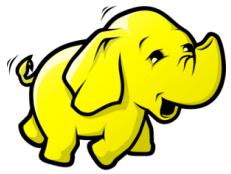
Ecossistema *Hadoop*: Ferramentas Python

Packages Collections: ChainMaps

A **ChainMap** encapsulates many dictionaries into a single unit and **returns a list of dictionaries**.

Sintaxe:

```
class collections.ChainMap(dict1, dict2)
```



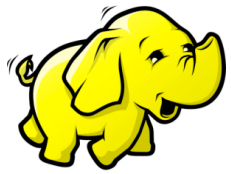
Ecossistema *Hadoop*: Ferramentas Python

Packages Collections: ChainMaps

A **ChainMap** encapsulates many dictionaries into a single unit and **returns a list of dictionaries**.

Sintaxe:

```
class collections.ChainMap(dict1, dict2)
```



Ecossistema *Hadoop*: Ferramentas Python

Packages Collections: ChainMaps

A **ChainMap** encapsulates many dictionaries into a single unit and **returns a list of dictionaries**.

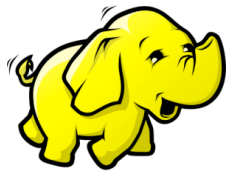
Example 1:

```
from collections import ChainMap

dict_A = {'a': 1, 'b': 2}
dict_B = {'c': 3, 'd': 4}
dict_C = {'e': 5, 'f': 6}

dict_maps = ChainMap(dict_A, dict_B, dict_C)

print(dict_maps)
```



Ecossistema *Hadoop*: Ferramentas Python

Packages Collections: ChainMaps

A **ChainMap** encapsulates many dictionaries into a single unit and **returns a list of dictionaries**.

Example 2:

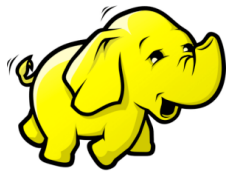
```
from collections import ChainMap

dict_A = {'a': 1, 'b': 2}
dict_B = {'c': 3, 'd': 4}
dict_C = {'e': 5, 'f': 6}

dict_maps = ChainMap(dict_A, dict_B, dict_C)
print(dict_maps)

dict_D = {'g' : 8}
cadeia = dict_maps.new_child(dict_D)
print(cadeia)
```

Add new dictionary to the previously created one!



Ecossistema *Hadoop*: Ferramentas Python

Packages Collections: ChainMaps

A **ChainMap** encapsulates many dictionaries into a single unit and **returns a list of dictionaries**.

Example 3:

```
from collections import ChainMap
import json

arquivo = open("contar1.txt", "r", encoding = "UTF-8")
contador1 = arquivo.read()
contador1 = json.loads(contador1)

arquivo = open("contar2.txt", "r", encoding = "UTF-8")
contador2 = arquivo.read()
contador2 = json.loads(contador2)

print(ChainMap(contador1, contador2))
```

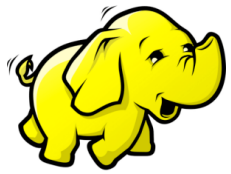
ARA0168
TÓPICOS DE BIG DATA EM PYTHON

3.2 – Ferramentas Python para Algoritmos em Big Data: *Map() and Filter()*

Universidade Estácio de Sá

Prof. Simone Gama

simone.gama@estacio.br



Ecossistema *Hadoop*: Ferramentas Python

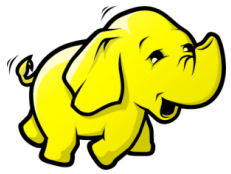
Function Map()

Usa-se a função integrada do Python **`map()`** para aplicar uma função a cada item em um iterável (como uma lista ou um dicionário) e retornar um novo iterador para recuperar os resultados.

`map()` retorna um objeto **`map`** (um iterador). Também pode-se passar o objeto map à função list(), ou outro tipo de sequência, para criar um iterável.

Sintaxe:

```
map(function, iterable, [iterable 2, iterable 3, ...])
```



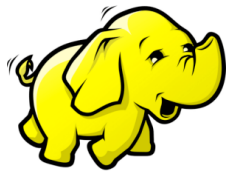
Ecossistema *Hadoop*: Ferramentas Python

Function Map()

Example 1:

```
import math

lista1 = [1, 4, 9, 16, 25]
lista2 = map(math.sqrt, lista1)
print (list(lista2))
```



Ecossistema *Hadoop*: Ferramentas Python

Function Map()

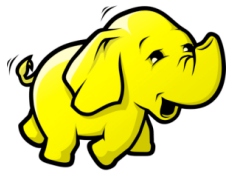
Example 1:

```
import math

lista1 = [1, 4, 9, 16, 25]
lista2 = map(math.sqrt, lista1)
print (list(lista2))
```

Equivalent list comprehension

```
lista2 = [math.sqrt(x) for x in lista1]
```



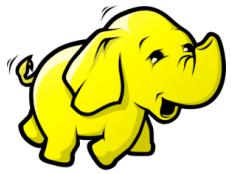
Ecossistema *Hadoop*: Ferramentas Python

Function Filter()

Filtra os elementos de uma sequência. O processo de filtragem é definido a partir de uma função que o programador passa como primeiro argumento da função. Assim, **`filter()`** só "deixa passar" para a sequência resultante aqueles elementos para os quais a chamada da função que o usuário passou retornar `True`.

Sintaxe:

```
filter(function_def, iterable, [iterable 2, iterable 3, ...])
```



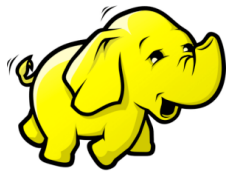
Ecossistema *Hadoop*: Ferramentas Python

Function Filter()

Example 2:

```
def maior_zero(x):
    return x > 0

valores = [100, 8, -1, 3, 5, -9, -12]
print (list(filter(maior_zero, valores)))
```



Ecossistema *Hadoop*: Ferramentas Python

Function Filter()

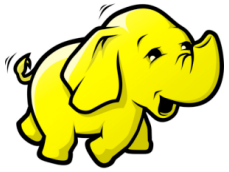
Example 2:

```
def maior_zero(x):
    return x > 0

valores = [100, 8, -1, 3, 5, -9, -12]
print (list(filter(maior_zero, valores)))
```

Equivalent list comprehension

```
[x for x in valores if x > 0]
```

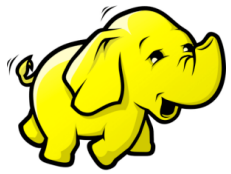


Ecossistema *Hadoop*: Ferramentas Python

Function Lambda()

A **lambda** function is a small anonymous function. A **lambda** function can take any number of arguments, but can only have one expression.

It works the same as the *Filter()* function, but without help or a function call.



Ecossistema *Hadoop*: Ferramentas Python

Function Lambda()

Example 3:

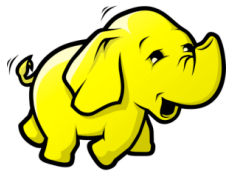
```
valores = [100, 8, -1, 3, 5, -9, -12]
print (list(filter(lambda x: x > 0, valores)))
```

using lambda

using Filter

```
def maior_zero(x):
    return x > 0

valores = [100, 8, -1, 3, 5, -9, -12]
print (list(filter(maior_zero, valores)))
```

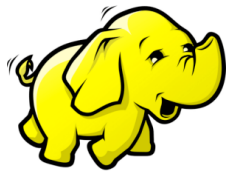


Ecossistema *Hadoop*: Ferramentas Python

Function Lambda()

Example 3:

```
valores = [100, 8, -1, 3, 5, -9, -12]
print (list(filter(lambda x: x > 0, valores)))
```



Ecossistema *Hadoop*: Ferramentas Python

Function Lambda()

Example 4:

```
numbers = [10, 12, 21, 33, 45, 55]
mapped_numbers = list(map(lambda x: x * 2 + 3, numbers))
print(mapped_numbers)
```

Estácio

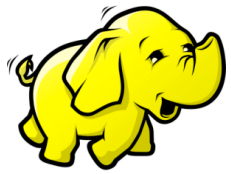
ARA0168
TÓPICOS DE BIG DATA EM PYTHON

3.3 – Ferramentas Python para Algoritmos em Big Data: *Expressões Regulares*

Universidade Estácio de Sá

Prof. Simone Gama

simone.gama@estacio.br



BIG DATA: Expressões Regulares

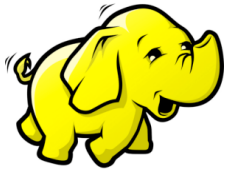
Regex

- Na ciência da computação, uma **expressão regular** (***regex*** ou ***regexp***) provê uma forma concisa e flexível de identificar cadeias de caracteres de interesse, como **caracteres particulares**, **palavras** ou **padrões de caracteres**.
- Expressões regulares são escritas numa linguagem formal que pode ser interpretada por um processador de expressão regular, um programa que serve um gerador de **analisador sintático** ou **examina o texto e identifica as partes que casam com a especificação dada**.

BIG DATA: Expressões Regulares

Regex

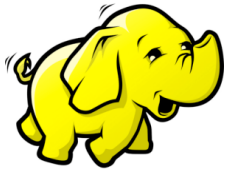
Ou seja, uma **Expressão Regular** (ER) é um método formal de se especificar um **padrão de texto**.



BIG DATA: Expressões Regulares

Regex: Importância

O uso atual de expressões regulares inclui **procura** e **substituição** de texto em editores de texto e linguagens de programação, **validação de formatos de texto** (validação de protocolos ou formatos digitais), **realce de sintaxe** e **filtragem de informação**.



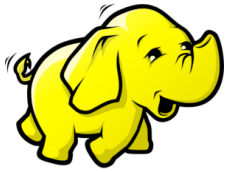
BIG DATA: Expressões Regulares

Regex: Escapes de Caracteres

Talvez o mais conhecido ícone de expressões regulares seja a **barra invertida** (\). Ela indica que o próximo caractere é um caractere especial ou se ele deve ser interpretado literalmente.

Caractere com escape	Descrição
\b	Em uma classe de caractere, corresponde a um backspace.
\t	Corresponde a uma tabulação.
\v	Corresponde a uma tabulação vertical

Veja a Tabela completa de caracteres de Escape: https://learn.microsoft.com/pt-br/dotnet/standard/base-types/character-escapes-in-regular-expressions



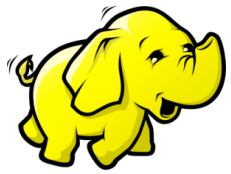
BIG DATA: Expressões Regulares

Regex: Escapes de Caracteres

Talvez o mais conhecido ícone de expressões regulares seja a **barra invertida** (\). Ela indica que o próximo caractere é um caractere especial ou se ele deve ser interpretado literalmente.

Caractere com escape	Descrição
\b	Em uma classe de caractere, corresponde a um backspace.
\t	Corresponde a uma tabulação.
\v	Corresponde a uma tabulação vertical

Veja a Tabela completa de caracteres de Escape: https://learn.microsoft.com/pt-br/dotnet/standard/base-types/character-escapes-in-regular-expressions



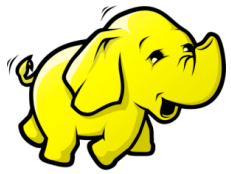
BIG DATA: Expressões Regulares

Package re

re.match - Determina se a expressão regular combina com o início da string. Retorna a posição da string buscada.

```
import re

string = "frase digitada e testada"
resultado = re.match('frase', string)
```



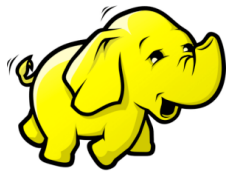
BIG DATA: Expressões Regulares

Package re

re.search - Varre a string, procurando o primeiro local onde a expressão regular tem correspondência. Retorna a posição da string buscada.

```
import re

string = "frase digitada e testada"
resultado = re.search('testada', string)
```



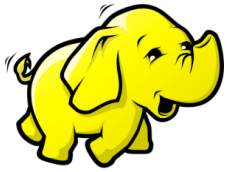
BIG DATA: Expressões Regulares

Package re

re.findall - Encontra todas as sub strings que têm correspondência com a expressão regular, e as retorna como uma lista com as strings encontradas.

```
import re

string = "frase digitada e testada"
resultado = re.findall('ta', string)
```



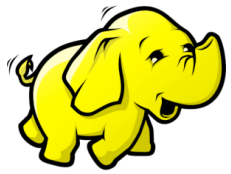
BIG DATA: Expressões Regulares

Package re

re.finditer - Encontra todas as substrings que têm correspondência com a expressão regular, e as retorna dentro de um ***iterador***.

```
import re

string = "frase digitada e testada"
for i in re.finditer('ta', string):
  print(i)
```

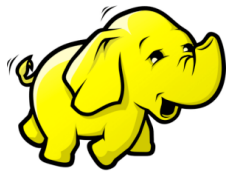


BIG DATA: Expressões Regulares

Padrões em *Regex*

Alguns padrões de símbolos podem ser usados para facilitar as buscas.

Padrão	Significado
^	Nega a expressão OU marca o início de uma string
$	Marca o final de uma string
+	Prolonga o caractere anterior
*	Prolonga o caractere anterior, mas ele também pode não existir
?	Diz que há dúvida se o caractere anterior a ele existe
.	Substitui qualquer caractere (apenas um)
[A-Z]	Procura qualquer caractere maiúsculo de **A** a **Z**
[a-z]	Procura qualquer caractere minúsculo de **a** a **z**
[0-9]	Procura qualquer dígito de **0** a **9**
[125]	Procura os dígitos **1**, **2** ou **5**
[^0-9]	O ^ nega a expressão a seguir, logo, procura tudo que não for número.
[abc]	Procura os caracteres **a**, **b** ou **c**
[^A-Z]	Procura tudo que não for letra maiúscula
[a-zA-Z]	Procura tudo que for letra
\s	Procura espaços
\w	Procura caracteres, exceto os especiais
\d	Também procura qualquer dígito de **0** a **9**



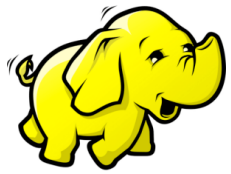
BIG DATA: Expressões Regulares

Padrões em *Regex* - Exemplo 1:

```
import re

lista = [ 'www.google.com', 'https://www.google.com', 'google.com.br' ]
for string in lista:
    print(re.findall("^www", string))
```

O que retorna o código acima?



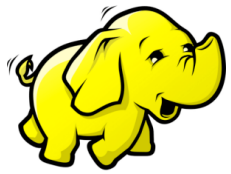
BIG DATA: Expressões Regulares

Padrões em *Regex* - Exemplo 2:

```
import re

lista = [ 'www.google.com', 'https://www.google.com', 'google.com.br' ]
for string in lista:
    print(re.findall("com$", string))
```

E agora, o que retorna o código acima?

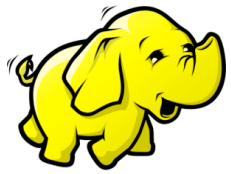


BIG DATA: Expressões Regulares

Padrões em *Regex* - Exemplo 3:

```
import re
nomes = ['nadia', 'marcos', 'norman', 'nivia']
padrao = '^n...a$'

for palavra in nomes:
    resultado = re.match(padrao, palavra)
    if resultado:
        print("Padrão encontrado!")
    else:
        print("Padrão NÃO encontrado!")
```



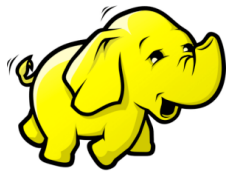
BIG DATA: Expressões Regulares

Padrões em *Regex* - Exemplo 3:

```
import re
nomes = ['nadia',
padrao = '^n...a$'

for palavra in nomes:
    resultado = re.match(padrao, palavra)
    if resultado:
        print("Padrão encontrado!")
    else:
        print("Padrão NÃO encontrado!")
```

Qualquer sequência de cinco letras começando com **n** e terminando com **a**.

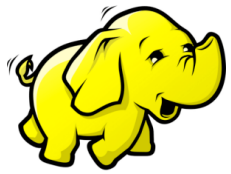


BIG DATA: Expressões Regulares

Padrões em *Regex* - Exemplo 4:

```
import re
def text_match(text):
        padrao = '^a(b*)$'
        if re.search(padrao,  text):
                return 'Correspondência encontrada!'
        else:
                return('Correspondência Não encontrada!')

print(text_match("ac"))
print(text_match("abc"))
print(text_match("a"))
print(text_match("ab"))
print(text_match("abb"))
print(text_match("abba"))
```



BIG DATA: Expressões Regulares

Padrões em *Regex* - Exemplo 4:

```
import re
def text_match(text):
        padrao = '^a(b*)$'
        if re.search(padrao,  text):
                return 'Correspondência encontrada!'
        else:
                return('Correspondência Não encontrada!')

print(text_match("ac"))
print(text_match("abc"))
print(text_match("a"))
print(text_match("ab"))
print(text_match("abb"))
print(text_match("abba"))
```

String que tenha um “**a**” seguido de zero ou mais “**b’s**”.

Dúvidas?

Bibliografia

- FACELI, Katti. **Inteligência Artificial Uma abordagem de aprendizado de máquina.** 2ª Ed. Rio de Janeiro: LTC, 2021. Disponível em:

Bibliografia Auxiliar

- *MapReduce* com Python: MapReduce with Python. An introduction to the MapReduce… | by Technologies In Industry 4.0 | Python in Plain English
- Collections Python:
  - Python Collections Module – GeeksforGeeks
  - Counter - Python Module of the Week (pymotw.com)