ARA0168
TÓPICOS DE BIG DATA EM PYTHON

Aula 4 – Ecossistema *Hadoop*: Parte II

MapReduce

Universidade Estácio de Sá

Prof. Simone Gama

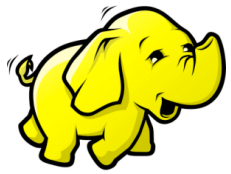
simone.gama@estacio.br

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*)



Ecossistema *Hadoop* : MapReduce

MapReduce (Mapear e Reduzir)

O fluxograma apresenta etapas de um processo MapReduce:

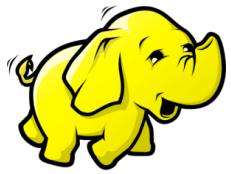
ARA0168
TÓPICOS DE BIG DATA EM PYTHON

4.1 – Ferramentas Python para Redução:
Reduce

Universidade Estácio de Sá

Prof. Simone Gama

simone.gama@estacio.br



Ecossistema *Hadoop* : Reduce

Função *Reduce* (Reduzir)

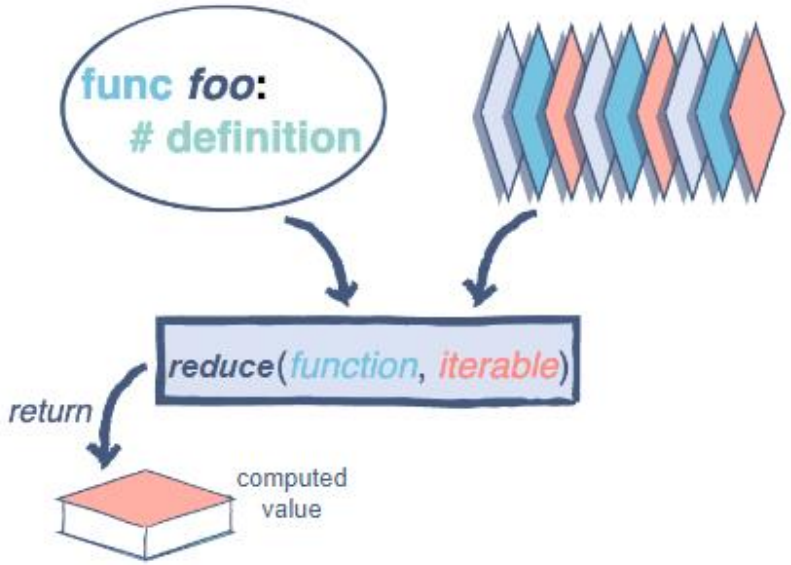
A função **`reduce()`**, disponível no módulo *built-in* **functools**[1], serve pra "reduzir" um iterável (como uma lista) a um único valor.

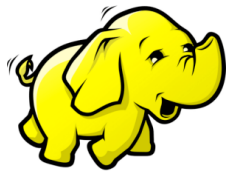
[1]Link: functools — Higher order functions and operations on callable objects — Python v3.0.1 documentation

Ecossistema *Hadoop* : Reduce

Função *Reduce* (Reduzir)

A ideia por trás do **`reduce()`** é pegar uma função existente, aplicá-la cumulativamente a todos os itens em um iterável e gerar um único valor final. Em geral, o reduce() do Python é útil para processar iteráveis **sem escrever loops for explícitos**.



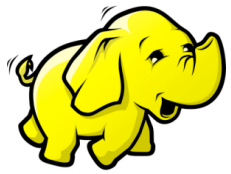


Ecossistema *Hadoop* : Reduce

Função *Reduce* (Reduzir)

Sintaxe da função **`reduce()`**:

```
reduce(function, iterable, initializer=None)
```



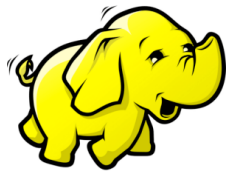
Ecossistema *Hadoop* : Reduce

Função *Reduce* (Reduzir)

Sintaxe da função **`reduce()`**:

```
reduce(function, iterable, initializer=None)
```

function: A função que deve ser aplicada aos elementos do iterável. Esta função deve receber dois argumentos.



Ecossistema *Hadoop* : Reduce

Função *Reduce* (Reduzir)

Sintaxe da função **`reduce()`**:

```
reduce(function, iterable, initializer=None)
```

`iterable`: pode ser uma lista, tupla ou qualquer outro tipo de dado iterável.

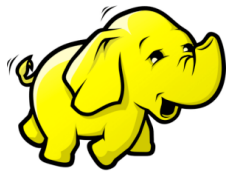
Ecossistema *Hadoop* : Reduce

Função *Reduce* (Reduzir)

Sintaxe da função **`reduce()`**:

```
reduce(function, iterable, initializer=None)
```

`initializer`: Um argumento opcional, que é o valor inicial do acumulador. Se não for fornecido, o primeiro item do iterável será usado como inicializador.

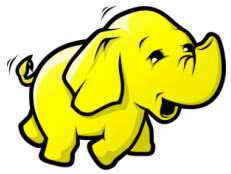


Ecossistema *Hadoop* : Reduce

Função *Reduce* (Reduzir)

1º Passo: Mapeando os maiores valores de listas em outra lista:

```
lista = [ [8, 2, 3], [4, 5, 6], [10, 8, 9] ]
lista2 = list(map(max, lista))
```

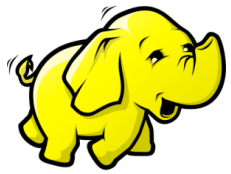


Ecossistema *Hadoop* : Reduce

Função *Reduce* (Reduzir)

2º Passo: Reduzindo os valores obtidos para a soma desses:

```
import operator
from functools import reduce

lista = [ [8, 2, 3], [4, 5, 6], [10, 8, 9] ]
lista2 = list(map(max, lista))
soma = reduce(operator.add, lista2)
```



Ecossistema *Hadoop* : Reduce

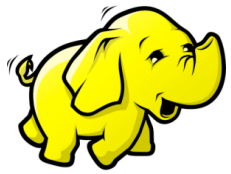
Função *Reduce* (Reduzir)

2º Passo: Reduzindo os valores obtidos para a soma desses:

```
import operator
from functools import reduce
```

No Python 3.x, se precisar usar **`reduce()`**, primeiro terá que importar a função para seu escopo atual usando uma instrução de importação de uma das seguintes maneiras:

1. **`import functools`** e então use nomes completos como functools.reduce().
2. **`from functools import reduce`** e então chame reduce() diretamente.

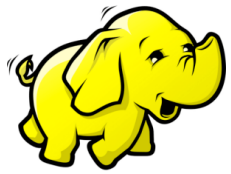


Ecossistema *Hadoop* : Reduce

Função *Reduce* (Reduzir)

Exemplo 1: Obtendo o maior elemento:

```
import functools

lista = [0, 3, 5, 8, 2]

print("O maior elemento é: ", end="")
print(functools.reduce(lambda a, b: a if a > b else b, lista))
```

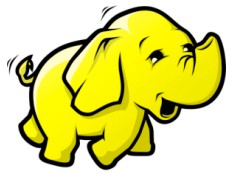


Ecossistema *Hadoop* : Reduce

Função *Reduce* (Reduzir)

Exemplo 2: Obtendo o produto de números de uma lista:

```
from functools import reduce

def mult(x, y):
    return x * y

minha_lista = [2, 4, 5, 3]

produto = reduce(mult, minha_lista)
print(produto)
```



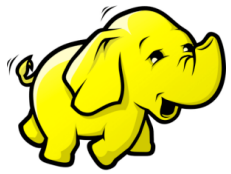
Ecossistema *Hadoop* : Reduce

Função *Reduce* (Reduzir)

Exemplo 2: Obtendo o produto de números de uma lista:

```
from functools import reduce

def mult(x, y):
    return x * y

minha_lista = [2, 4, 5, 3]

produto = reduce(mult, minha_lista)
print(produto)
```

É possível usar uma função própria para a redução.



Ecossistema *Hadoop* : Reduce

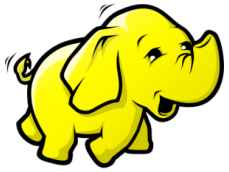
Função *Reduce* (Reduzir)

Exemplo 2: Obtendo o produto de números de uma lista:

Equivalente

```
from functools import reduce

minha_lista = [2, 4, 5, 3]

reduce(lambda x, y: x * y, minha_lista)
```

```
from functools import reduce

def mult(x, y):
    return x * y

minha_lista = [2, 4, 5, 3]

produto = reduce(mult, minha_lista)
print(produto)
```

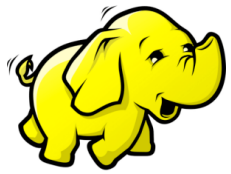


Ecossistema *Hadoop* : Reduce

Função *Reduce* (Reduzir)

Exemplo 3: Obtendo a maior palavra:

```
words = ['Python', 'Spark', "Aulas"]
maior_palavra = reduce(lambda x, y: x if len(x) > len(y) else y, words)
```

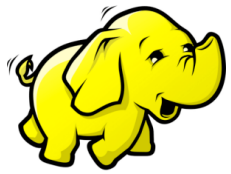


Ecossistema *Hadoop* : Reduce

Função *Reduce* (Reduzir)

Exemplo 4: União de conjuntos (sets):

```
sets = [{1, 2, 3}, {2, 3, 4}]
uniao = reduce(lambda x, y: x | y, sets)
print("União:", uniao)
```

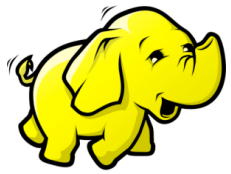


Ecossistema *Hadoop* : Reduce

Função *Reduce* (Reduzir)

Exemplo 5: *Reduce* com dicionários:

```
meu_dicionario = {'spark': 3, 'python': 2, "java": 2}
produto = reduce(lambda x, y: x * y, meu_dicionario.values())
print(produto)
```



Ecossistema *Hadoop* : Reduce

Função *Reduce* (Reduzir)

Exercícios para Praticar a função Reduce.

1. Seja uma lista de números, calcule:
    a) A **média** dessa lista de números.
    b) O **menor** elemento dessa lista
    c) A **soma** das elemento
2. Seja uma lista de strings:
    a) Concatene essa lista de strings.

Estácio

ARA0168
TÓPICOS DE BIG DATA EM PYTHON

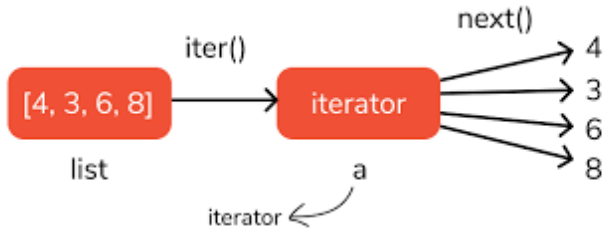
4.2 – Ferramentas Python para Redução:

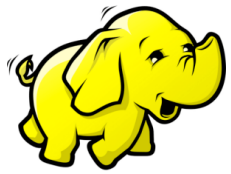
Yield

Universidade Estácio de Sá

Prof. Simone Gama

simone.gama@estacio.br

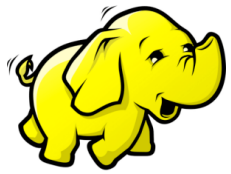




Ecossistema *Hadoop*: *Yield*

Função *Yield*

Em Python, **yield** é uma palavra-chave usada em funções geradoras para produzir um valor a ser retornado sem interromper a execução da função. Ao contrário do **return**, que finaliza a execução da função e retorna um valor, o **yield** permite que a função retorne um valor e pause sua execução, armazenando seu estado atual.

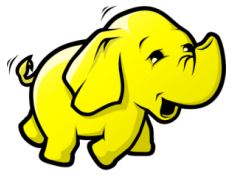


Ecossistema *Hadoop*: *Yield*

Função *Yield*

Quando a função é chamada novamente, ela **retoma a execução a partir do ponto em que foi interrompida** anteriormente, continuando a gerar valores.

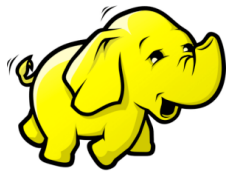
Isso permite que as **funções geradoras produzam sequências de valores infinitas** ou **muito grandes** que podem ser processados de forma eficiente um de cada vez, **sem exigir a alocação de uma grande quantidade de memória**.



Ecossistema *Hadoop*: *Yield*

Observe os seguintes geradores de listas de 0 a 9 (10 itens):

```
lista_compr = [x for x in range(10)]
print(lista_compr)

lista_gener = (x for x in range(10))
print(lista_gener)
```

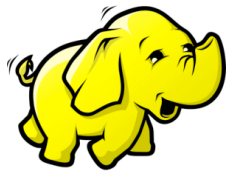


Ecossistema *Hadoop*: *Yield*

Observe os seguintes geradores de listas de 0 a 9 (10 itens):

```
lista_compr = [x for x in range(10)]
print(lista_compr)

lista_gener = (x for x in range(10))
print(lista_gener)
```

A 1ª lista é gerada por *list comprehension*.



Ecossistema *Hadoop*: *Yield*

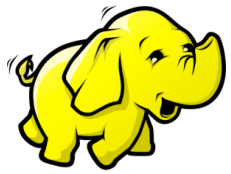
Observe os seguintes geradores de listas de 0 a 9 (10 itens):

```
lista_compr = [x for x in range(10)]
print(lista_compr)
```

```
lista_gener = (x for x in range(10))
print(lista_gener)
```

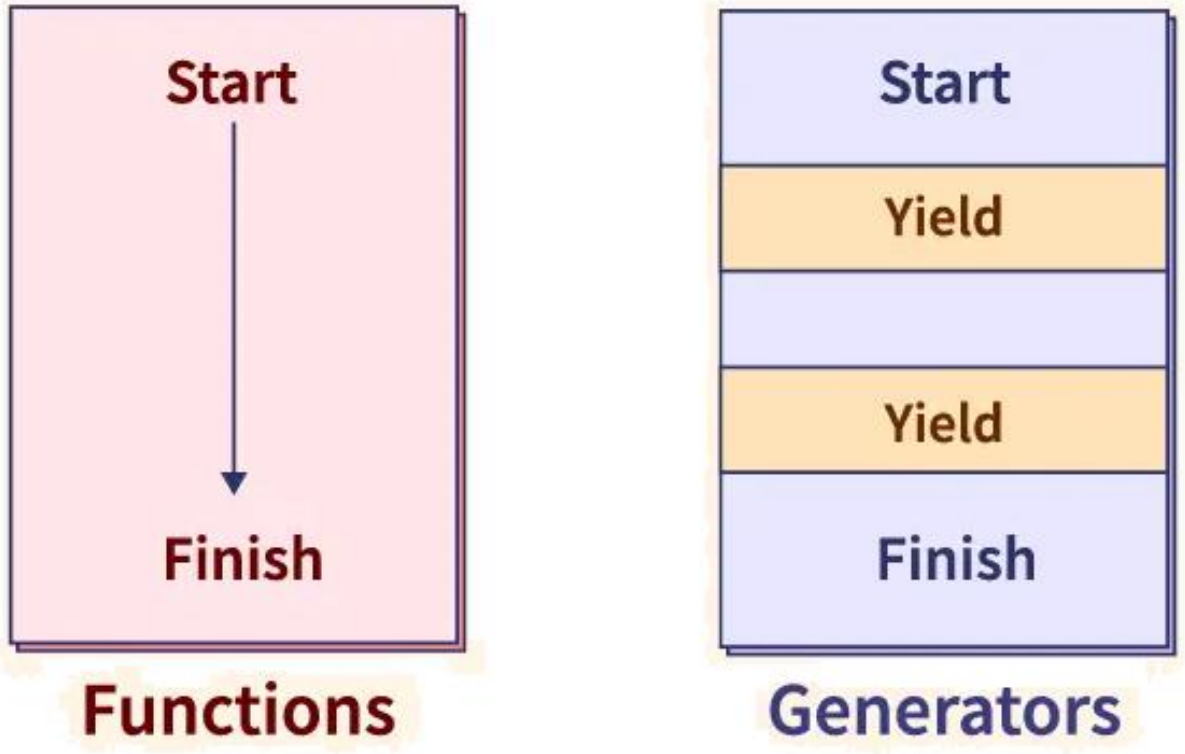
A 2ª lista é gerada **sob demanda**[2], ou seja, um objeto de itens.

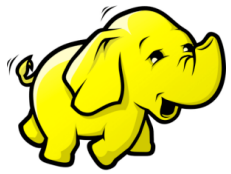
[2]sem usar a memória



Ecossistema *Hadoop*: *Yield*

Função *Yield*





Ecossistema *Hadoop*: *Yield*

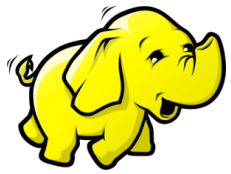
Função *Yield* – Exemplo 1

```
def contador(max):
    num = 0
    while num < max:
        yield num
        num += 1

for i in contador(5):
    print(i)
```

A função geradora "**contador**" retorna uma sequência de valores inteiros de **0 a max-1, um** de cada vez, usando o "**yield**".

O loop "**for**" na parte inferior do código chama a função "contador" e itera **sobre cada valor gerado**, imprimindo-o na tela.

O resultado seria a impressão de "0 1 2 3 4".

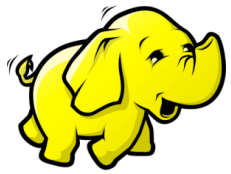


Ecossistema *Hadoop*: *Yield*

Função *Yield* – Exemplo 2

```
def utilizandoYield(x):
  for i in range(x):
    if(i % 2 == 0):
      yield 'par'
    else:
      yield 'impar'

lista = utilizandoYield(10)

for i in lista:
  print(i)
```

A função “**utilizandoYield**" verifica com cada gerador (0 a 10) se o mesmo é par ou ímpar e “abastece” a “lista” com o resultado encontrado.

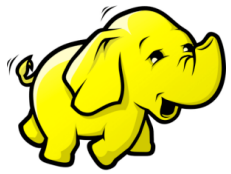


Ecossistema *Hadoop*: *Yield*

Função *Yield* – Exemplo 3

```
def quadrados(x, y):
  for i in range(x, y):
    yield i*i

if __name__ == '__main__':
  x, y = (1, 10)

for i in quadrados(x, y):
    print(i)
```

A função "**quadrados**" verifica com cada item do gerador (1 a 10) o cálculo do quadrado de cada número.

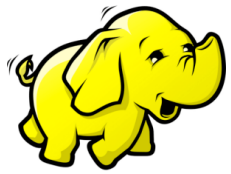


Ecossistema *Hadoop*: *Yield*

Função *Yield* – Exemplo 4

```
csv_gen = open("csv_arq.txt")
conta_linha = 0

for linha in csv_gen:
    conta_linha += 1

print(f"Total linhas {conta_linha}")
```

Leitura de arquivo grandes (Big Data).

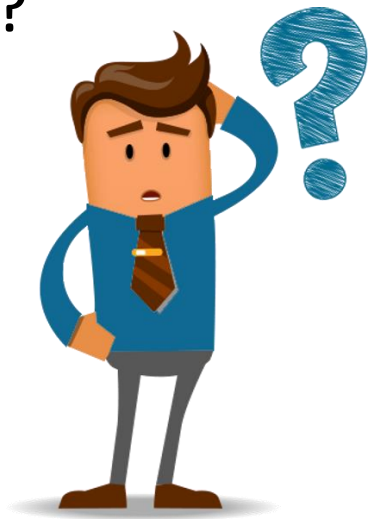


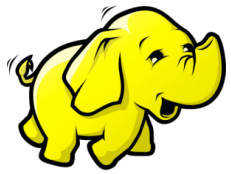
Ecossistema *Hadoop*: *Yield*

Função *Yield* – Exemplo 4

```
csv_gen = open("csv_arq.txt")
conta_linha = 0

for linha in csv_gen:
    conta_linha += 1

print(f"Total linhas {conta_linha}")
```

Esse código ainda funcionaria se o arquivo fosse muito grande? E se o arquivo for maior do que a memória disponível?





Ecossistema *Hadoop*: *Yield*

Função *Yield* – Exemplo 4

```
def csv_reader(file_name):
    for row in open(file_name, "r"):
      yield row

file_name = "csv_arq.txt"
print(csv_reader(file_name))
```

Uma opção é ler linhas do arquivo usando o `yield`. Basicamente se transformou `csv_reader()`em uma função geradora.
Esta versão abre um arquivo, **percorre cada linha** e retorna cada linha, em vez de retorná-la por inteiro.

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

ARA0168
TÓPICOS DE BIG DATA EM PYTHON

ANEXO I

Universidade Estácio de Sá

Prof. Simone Gama

simone.gama@estacio.br

ANEXO I

O que são processos Python

Um **processo** refere-se a um programa de computador.

Todo programa Python é um processo e tem um *thread* chamado thread principal usado para executar as instruções do programa.

Cada processo é, na verdade, uma **instância** do interpretador Python que executa instruções Python (código de bytes Python), que é um nível um pouco menor do que o código que você digita em seu programa Python.

ANEXO I

O que são processos Python

- Às vezes, precisamos criar novos processos para executar tarefas adicionais simultaneamente.
- Python fornece processos reais em nível de sistema através da **classe Process** no módulo de multiprocessamento.

ANEXO I

Processos

```
from multiprocessing import Process

def task(nome):
    print('Olá', nome)

if __name__ == '__main__':
    p = Process(target=task, args=('Python',))
    p.start()
    p.join()
```

ANEXO I

Processos

```
from multiprocessing import

def task(nome):
    print('Olá', nome)

if __name__ == '__main__':
    p = Process(target=task, args=('Python',))
    p.start()
    p.join()
```

Sempre que criamos novos processos, devemos proteger o ponto de entrada do programa.

ANEXO I

Processos

Uma tarefa pode ser executada em um novo processo criando uma instância da classe **Process** e especificando a função a ser executada no novo processo por meio do argumento "*target*".

```
from multiprocessing

def task(nome):
    print('Olá', nome)

if __name__ == '__main__':
    p = Process(target=task, args=('Python',))
    p.start()
    p.join()
```

ANEXO I

Processos

```
from multiprocessing import Process

def task(nome):
    print('Olá', nome)

if __name__ == '__main__':
    p = Process(target=task, args=('Python',))
    p.start()
    p.join()
```

Depois que o processo é criado, ele deve ser iniciado chamando o método **`start()`**.

ANEXO I

Processos

```
from multiprocessing import Process

def task(nome):
    print('Olá', nome)

if __name__ == '__main__':
    p = Process(target=task, args=('Python',))
    p.start()
    p.join()
```

Podemos então esperar que a tarefa seja concluída juntando-nos ao processo.

ANEXO I

Pool de Processos

Um **pool de processos** é um padrão de programação para gerenciar automaticamente um pool de processos de trabalho.

O pool é responsável por um número fixo de processos.

- Ele **controla quando eles são criados**, como quando eles são necessários.
- Ele também **controla o que eles devem fazer quando não estão sendo usados**, como fazê-los esperar sem consumir recursos computacionais.

ANEXO I

Exemplo de Pool de Processos para Multiplicação de Matrizes

```python
import numpy as np
import multiprocessing as mp

def multiply_row_col(row, col):
    return sum(row[i] * col[i] for i in range(len(row)))

def multiply_matrix(matrix1, matrix2):
    result = np.zeros((len(matrix1), len(matrix2[0])))

    with mp.Pool() as pool:
        for i in range(len(matrix1)):
            for j in range(len(matrix2[0])):
                result[i][j] = pool.apply(multiply_row_col, args=(matrix1[i], [row[j] for row in matrix2]))

    return result

if __name__ == '__main__':
    matrix1 = np.random.rand(3, 4)
    matrix2 = np.random.rand(4, 2)
    result = multiply_matrix(matrix1, matrix2)
```

ANEXO I

Exemplo de Pool de Processos para Raiz Quadrada

```python
import math
import multiprocessing as mp

def calculate_sqrt(num):
    return math.sqrt(num)

if __name__ == '__main__':
    nums = [2, 4, 6, 8, 10]
    with mp.Pool() as pool:
        result = pool.map(calculate_sqrt, nums)
```

Bibliografia ANEXO I

- **Multi Processamento em Python (doc):**
  - https://docs.python.org/3/library/multiprocessing.html
- **Pool de Processos**:
  - https://superfastpython.com/multiprocessing-pool-python/
- **Map de Processos**:
  - https://superfastpython.com/multiprocessing-pool-map/