Desafio 1 - Consumo de APIs

Objetivos:

Se desenvolver através de atividade prática como consumir APIs - **Application Programming Interface** (Interface de Programação de Aplicação). Recurso extremamente utilizado na área de Engenheiro de Dados. Consiste em puxar as informações de alguma fonte de dados e trazê-la para dentro de sua infraestrutura.

Conhecimento exigido:

* Lógica de programação
  + Curso de Lógica: <https://www.youtube.com/watch?v=iF2MdbrTiBM>
* Linguagem de programação (**Python**, JavaScript, Java e etc)
  + Curso de Python: <https://www.youtube.com/watch?v=GQpQha2Mfpg>
* APIs
  + <https://www.youtube.com/watch?v=ghTrp1x_1As>

Softwares Necessários:

* IDE Visual Studio: <https://code.visualstudio.com/>
* Interpretador e Plugins do VS:
  + [Python - Visual Studio Marketplace](https://marketplace.visualstudio.com/items?itemName=ms-python.python)
  + [Code Runner - Visual Studio Marketplace](https://marketplace.visualstudio.com/items?itemName=formulahendry.code-runner)
  + [Flake8 - Visual Studio Marketplace](https://marketplace.visualstudio.com/items?itemName=ms-python.flake8)
* Plugin Chrome
  + [https://chrome.google.com/webstore/detail/json-viewer/gbmdgpbipfallnflgajpaliibnhdgobh](https://chrome.google.com/webstore/detail/json-viewer/gbmdgpbipfallnflgajpaliibnhdgobh?utm_source=ext_sidebar&hl=fr)

Bibliotecas Utilizadas:

* Requests: <https://pypi.org/project/requests/>
* Grequests: <https://pypi.org/project/grequests/>

Desafio:

**Descrição**

Eu como desenvolvedor tenho a necessidade de consumir algumas informações de uma API voltada para dados de Pokémons e assim realizar algumas extrações dessas informações.

**Recursos do Desafio**

Endpoint da API Principal: <https://pokeapi.co/docs/v2>

Endpoint da API Species: <https://pokeapi.co/api/v2/pokemon-species/>

Volume de dados: Puxar a primeira geração

Formato das Informações:

{

"id": 1,

"name": "bulbasaur",

"height": 7,

"weight": 69,

"types":"grass/poison", types": [ { "slot": 1, "type": { "name": "grass",

"moves","A/B/C", (PODE ESTOURAR OS CARACTERES) LIMITAR A 3 / NÃO ACEITAR NULOS

"power": "10/20/30", LIMITAR A 3 / NÃO ACEITAR NULOS

"hp" : 45,

"attack" :49,

"defense" : 49

"is\_legendary":false

}

Gerações:

| id | name | start | end |
| --- | --- | --- | --- |
| 1 | Kanto | 1 | 151 |
| 2 | Johto | 152 | 251 |
| 3 | Hoenn | 252 | 386 |
| 4 | Sinnoh | 387 | 493 |
| 5 | Unova | 494 | 649 |
| 6 | Kalos | 650 | 721 |
| 7 | Alola | 722 | 809 |
| 8 | Galar/Hisui | 810 | 905 |
| 9 | Paldea | 906 | 1015 |

Passos:

1 - Desenvolver uma função que faça a requisição na API do Pokemon, onde terá como assinatura do método a seguinte linha:

def grequests\_pokemon\_api(start:int, end:int):

#start - primeiro pokemon da lista

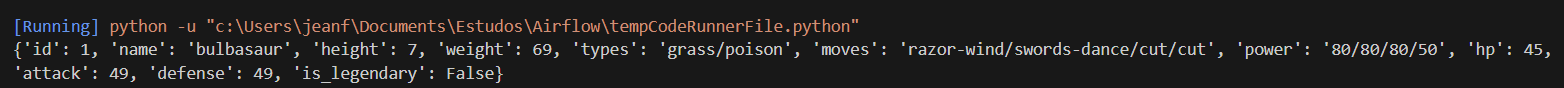
#end - ultimo pokemon da lista

#exemplo def grequests\_pokemon\_api(1, 151):



2 - Prepare os dados da consulta na API e transforme eles em: DataFrame, List, Class ou Dictionary. Qualquer uma das opções de armazenamento de estruturas.

3 - Imprima o resultado dessa consulta com os Pokemons da primeira geração.



Dica: No início reduza o escopo da consulta para **1** Pokémon para não perder tempo puxando muitos dados.

4 - Saída esperada:

{

"id":1,

"name":"bulbasaur",

"height":7,

"weight":69,

"types":"grass/poison",

"moves":"razor-wind/swords-dance/cut/cut",

"power":"80/80/80/50",

"hp":45,

"attack":49,

"defense":49,

"is\_legendary":false

}



Extras:

* Otimizar a consulta para o menor tempo possível:
  + Grequests
  + Pesquisar estrutura menos recurso de memória.
* Colocar logs de acompanhamento na execução.
  + import logging
* Realizar testes unitários.
  + PyTest
  + Unittest