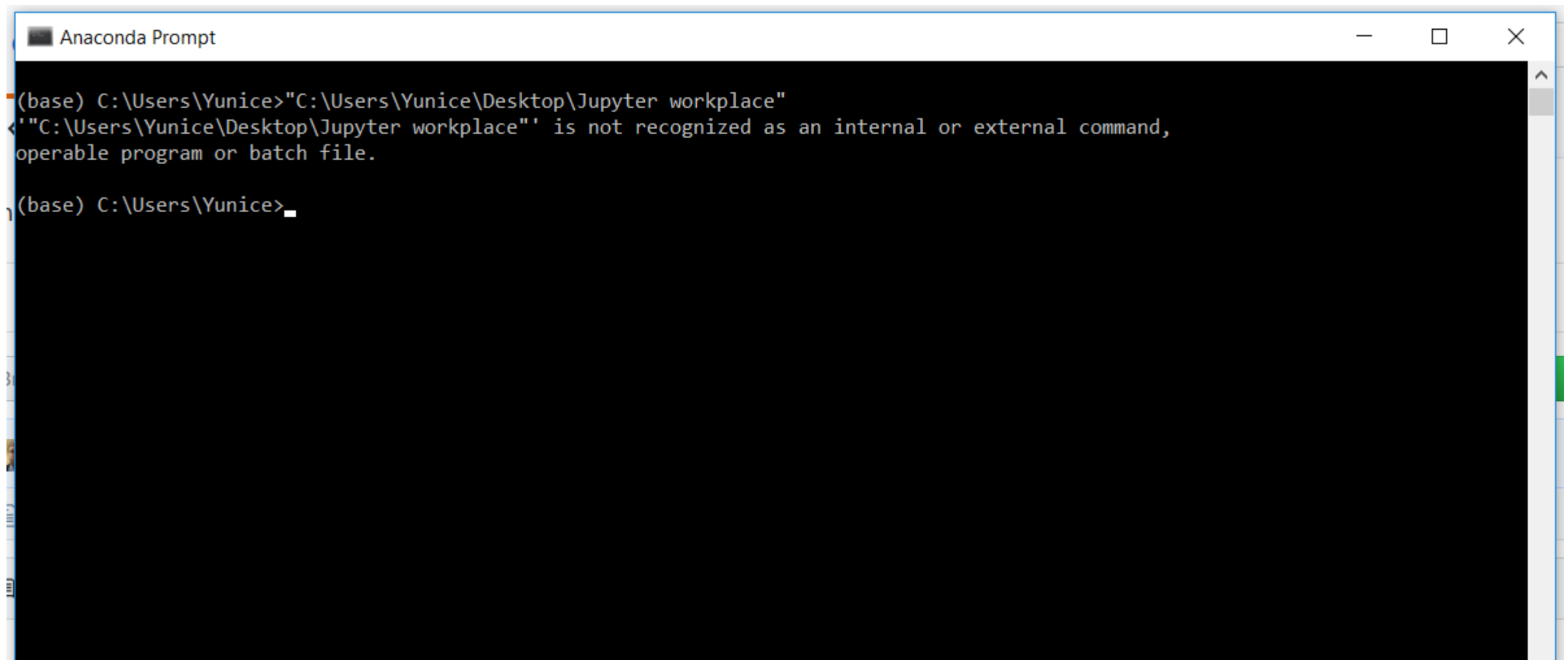


DigitalHouse >

DATA SCIENCE

Criando APIs - Python

**PRÉ-REQUISITO PARA A AULA
DE HOJE:
SABER EXECUTAR
SCRIPTS/PROGRAMAS PYTHON
PELO TERMINAL**

A screenshot of the Anaconda Prompt window. The title bar reads "Anaconda Prompt". The command prompt shows the user entering a command to run JupyterLab from the desktop. The system responds with an error message indicating that the command is not recognized. The prompt then returns to the user's input line.

```
(base) C:\Users\Yunice>"C:\Users\Yunice\Desktop\Jupyter workplace"  
<"C:\Users\Yunice\Desktop\Jupyter workplace" is not recognized as an internal or external command,  
operable program or batch file.  
(base) C:\Users\Yunice>_
```

Antes de entrarmos no assunto de APIs, vamos nos certificar que todos conseguem executar scripts Python através do Prompt de comando do sistema operacional de sua máquina.

Uma boa alternativa é usar o Anaconda Prompt.



```
C:\00 - PythonDev\scriptRecSys.py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

scriptRecSys.py
1 #!/usr/bin/env python
2 # coding: utf-8
3 import numpy as np
4 import pandas as pd
5
6 # ler arquivo de filmes
7 column_names = ['user_id', 'item_id', 'rating', 'timestamp']
8 df = pd.read_csv('u.data', sep='\t', names=column_names)
9 # ler arquivo de títulos
10 movie_titles = pd.read_csv("Movie_Id_Titles")
11 # merge dos arquivos
12 df = pd.merge(df, movie_titles, on='item_id')
13 # cálculo da média dos ratings
14 ratings = pd.DataFrame(df.groupby('title')['rating'].mean())
15 # cálculo da quantidade de ratings
16 ratings['num of ratings'] = pd.DataFrame(df.groupby('title')['rating'].count())
17 # matriz de interações
18 moviemat = df.pivot_table(index='user_id', columns='title', values='rating')
19 # entrada do número do filme pelo teclado
20 numero = input("Digite o índice do filme que você deseja saber a recomendação\n")
21 # busca o título do filme informado pelo índice/número
22 filme = movie_titles.loc[int(numero), 'title']
23 # apresenta filme escolhido na tela
24 print(filme)
25 # busca filme pelo título na matriz de interações
26 filme_user_ratings = moviemat[filme]
27 # calcula a correlação do filme em relação aos demais
28 similar_to_filme = moviemat.corrwith(filme_user_ratings)
29 # monta dataframe com os dados da correlação
30 corr_filme = pd.DataFrame(similar_to_filme, columns=['Correlation'])
31 # elimina vazios/NaN
32 corr_filme.dropna(inplace=True)
33 # agrega a quantidade de ratings para filtrá-lo posteriormente por este critério
34 corr_filme = corr_filme.join(ratings['num of ratings'])
35 # seleciona os filmes com a maior correlação, de acordo com a quantidade de ratings definida (100, inicialmente)
36 final = corr_filme[corr_filme['num of ratings'] > 100].sort_values('Correlation', ascending=False).head()
37
38 print(final)
39
```

Outro ponto importante para se trabalhar com o Terminal é utilizar um bom editor de texto que faça a marcação da sintaxe da linguagem, neste caso o Python. O Sublime Text é muito utilizado por programadores da área.



EXERCÍCIO 01




- Analisar o código do script **'scriptRecSys.py'** através de um editor de texto que formate a sintaxe do Python.
- Executar o script com sucesso pelo Terminal/Prompt do sistema operacional – obtendo a recomendação de filmes a partir de um índice informado (solução que fizemos na aula anterior)

CRIANDO APIs



No curso já aprendemos os conceitos de **Application Programming Interfaces** (APIs) e praticamos o consumo de dados através destas interfaces. Nesta aula iremos nos aprofundar no outro lado da questão – implementar APIs – provendo serviços a serem consumidos por terceiros.



WIKIPÉDIA
A enciclopédia livre

[Página principal](#)
[Conteúdo destacado](#)
[Eventos atuais](#)
[Esplanada](#)
[Página aleatória](#)
[Portais](#)

Artigo [Discussão](#)

[Ler](#) [Editar](#) [Editar código-fonte](#) [Ver histórico](#)

Interface de programação de aplicações [ocultar]

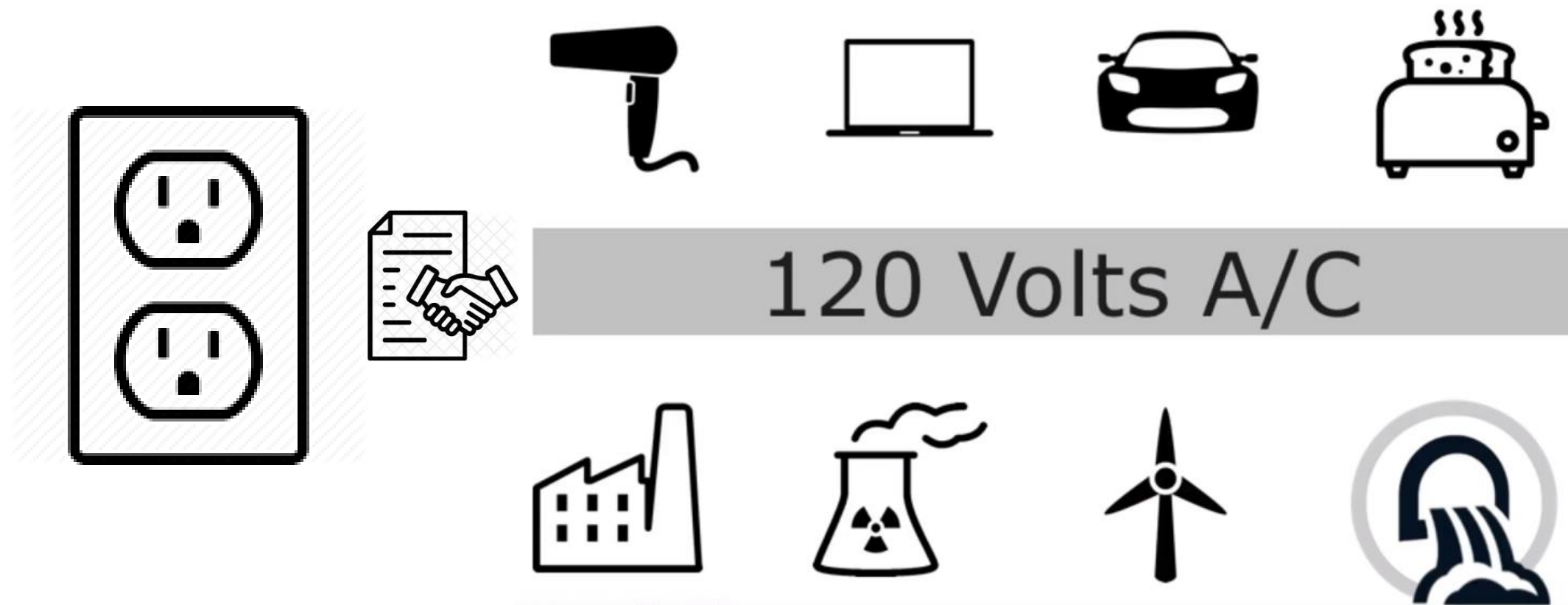
Origem: Wikipédia, a enciclopédia livre.

Nota: Para outros significados de *Api*, veja *Api (desambiguação)*.

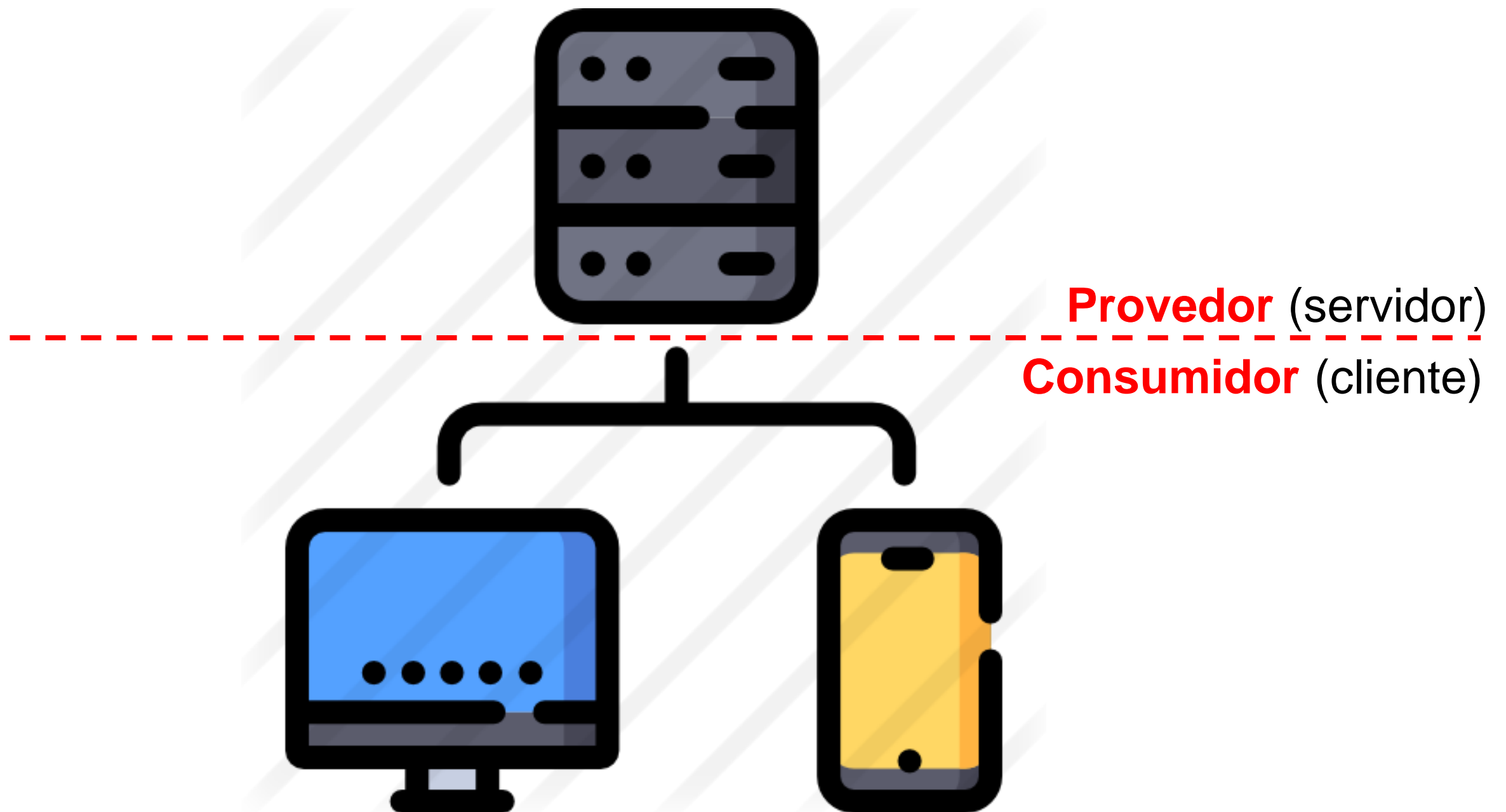
Interface de Programação de Aplicações ^(pt) ou **Interface de Programação de Aplicação** ^(pt-BR), cujo acrônimo **API** provém do Inglês ***A**pplication **P**rogramming **I**nterface*, é um conjunto de **rotinas** e padrões estabelecidos por um **software** para a utilização das suas funcionalidades por aplicativos que não pretendem envolver-se em detalhes da **implementação** do software, mas apenas usar seus serviços.^[1]

https://pt.wikipedia.org/wiki/Interface_de_programa%C3%A7%C3%A3o_de_aplica%C3%A7%C3%B5es

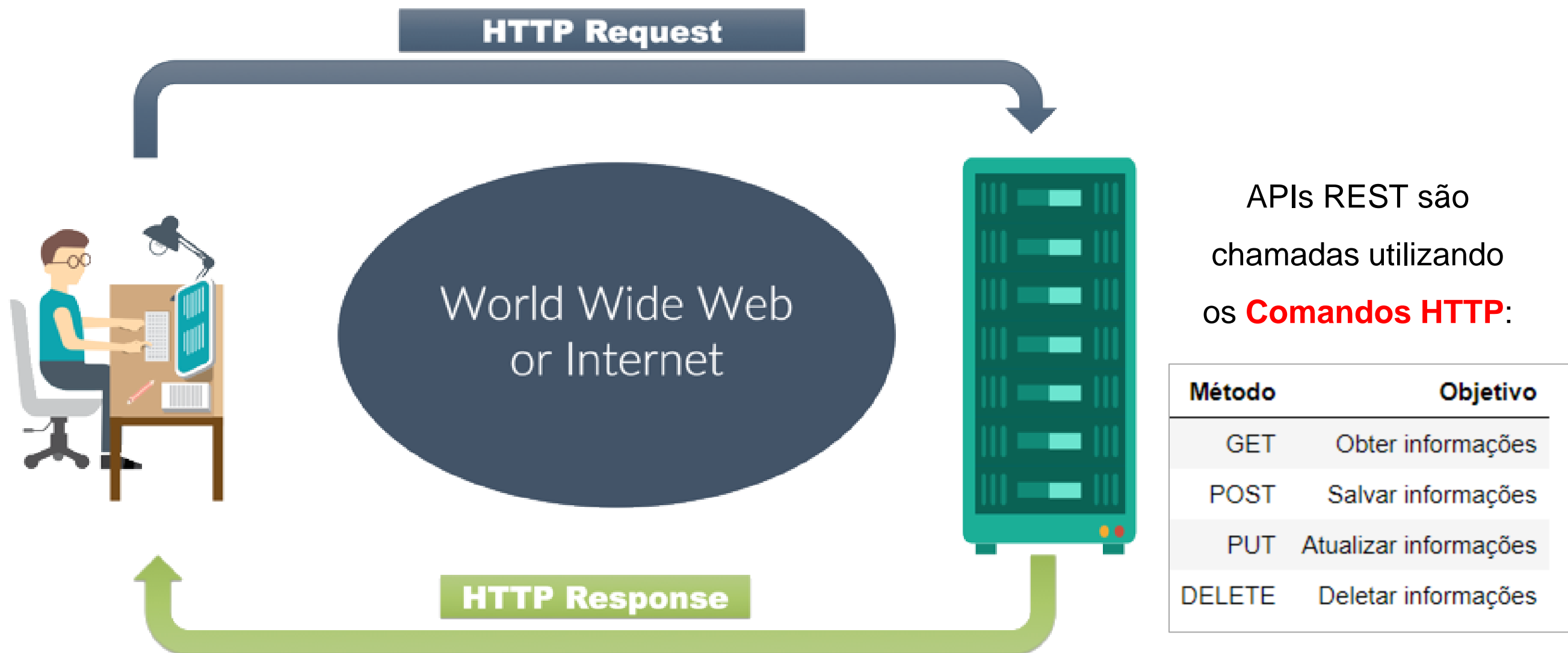
*“Interface de Programação Online ou Interface de Programação de Aplicação cujo provém do inglês “Application Programming Interface”, é um conjunto de **rotinas** e **padrões** estabelecidos por um **software** para a utilização das suas **funcionalidades** por aplicativos que **não pretendem envolver-se em detalhes da implementação do software**, mas apenas usar seus serviços.”*



Podem se entender as APIs como uma espécie de **Contrato** que define regras e comportamentos esperados entre duas partes na utilização de um serviço.



Pode se entender a utilização de APIs sob dois pontos de vista:
do **Provedor** (servidor) e do **Consumidor** (cliente) do serviço
em questão.

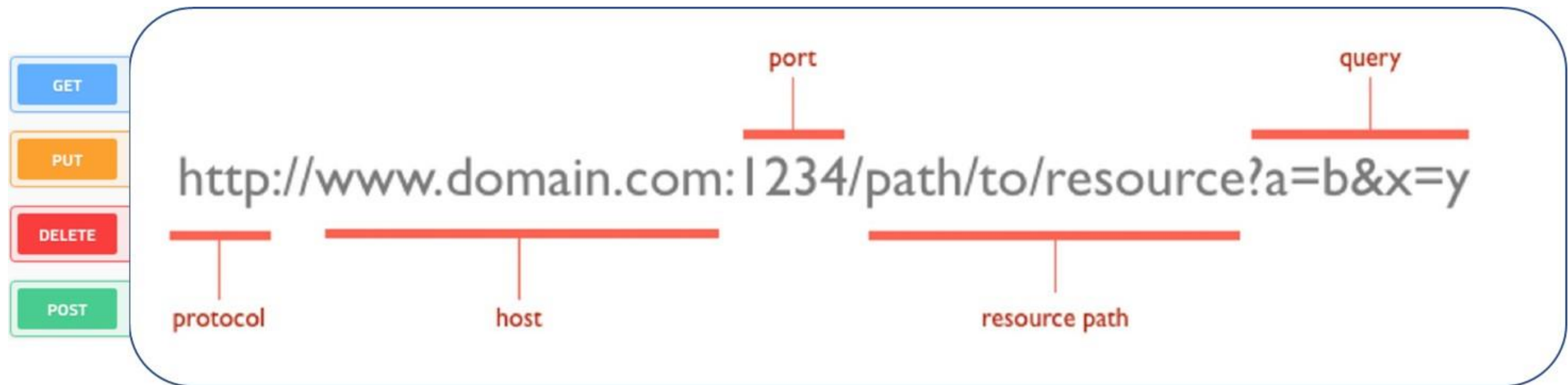


Códigos de Retorno

- 200 - tudo correu bem, e o resultado foi devolvido (se houver)
- 301- o servidor está redirecionando você para um terminal diferente. Isso pode acontecer quando uma empresa troca nomes de domínio ou um nome de terminal é alterado.
- 401- o servidor acha que você não está autenticado. Isso acontece quando você não envia as credenciais corretas para acessar uma API (falaremos sobre autenticação em uma postagem posterior).
- 400- o servidor acha que você fez um pedido incorreto. Isso pode acontecer quando você não envia os dados corretos, entre outras coisas.
- 403 - o recurso que você está tentando acessar é proibido - você não tem as permissões certas para visualizá-lo.
- 404 - o recurso que você tentou acessar não foi encontrado no servidor.

Método	Objetivo
GET	Obter Informações
POST	Inserir Informações
PUT	Atualizar Informações
DELETE	Apagar Informações





Como o servidor faz para saber o que está sendo pedido na solicitação?

Isso é especificado na URL (Uniform Resource Locator), uma espécie de caminho que indica onde um recurso pode ser encontrado.

protocol: Indica o protocolo que será utilizado para acessar o HTTP, FTP, HTTPS

host: Indica como encontrar na rede o servidor que tem o recurso.

port: Indica em qual porta TCP/IP está escutando o servidor **path**: Indica o caminho para localizar o recurso dentro do servidor

query: Indica qual é a consulta que está sendo realizada

XML (ontem)

```

1 <EmpRecord>
2   <Employee><-id>emp01
3   </-id>
4   <name>Alex</name>
5   <job>Developer</job>
6   <skills>python, C/C++, paskal</skills>
7 </Employee>
8 <Employee><-id>emp02
9 </-id>
10 <name>Bob</name>
11 <job>Tester</job>
12 <skills>lips, forton, REST
    APIs</skills>undefined</Employee>undefined</EmpRe
    cord>

```

JSON (hoje)

```

1 {
2   "EmpRecord": {
3     "Employee": [
4       {
5         "-id": "emp01",
6         "name": "Alex",
7         "job": "Developer",
8         "skills": "python, C/C++, paskal"
9       },
10      {
11        "-id": "emp02",
12        "name": "Bob",
13        "job": "Tester",
14        "skills": "lips, forton, REST APIs"
15      }
16    ]
17  }
18 }

```

YAML (amanhã)

```

1 ---
2 EmpRecord:
3   Employee:
4     - "-id": emp01
5       name: Alex
6       job: Developer
7       skills: python, C/C++, paskal
8     - "-id": emp02
9       name: Bob
10      job: Tester
11      skills: lips, forton, REST APIs
12

```

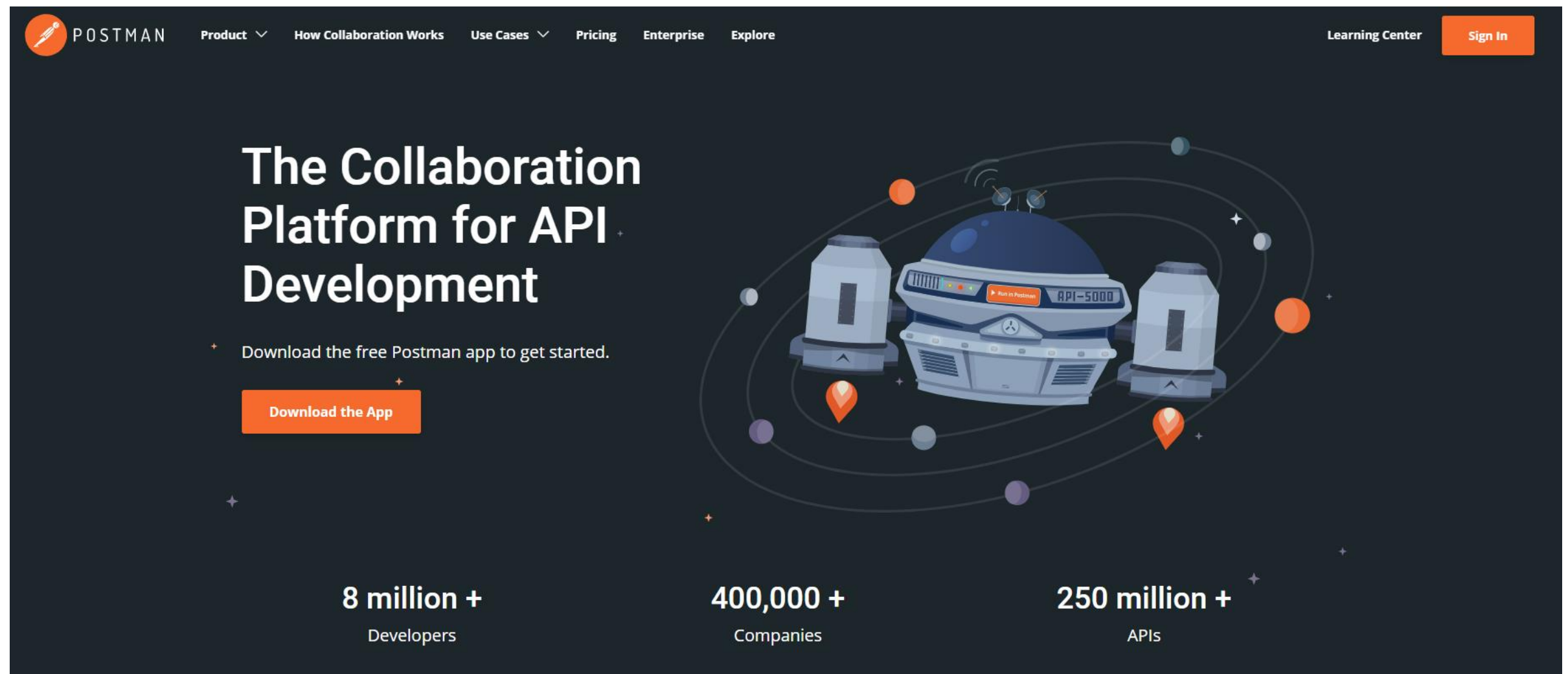
Anos atrás a comunicação entre serviços se dava por XML usando o padrão SOAP. Na era da mobilidade **JSON** passou a ser largamente usado. Hoje, aos poucos está sendo substituído por YAML, que é similar mas mais legível pelos olhos humanos. A estrutura de um json se parece muito com os **Dicionários** em **Python**.



Para o desenvolvimento de serviços, páginas e aplicativos Web – os dois Frameworks mais conhecidos e utilizados são o Flask e o Django. Existe um conteúdo enorme sobre o assunto na Web. Em suma, o Flask é um framework bem mais leve, flexível e simples. O Django é mais robusto e preparado para funções complexas, porém bem menos flexível para necessidades específicas.



Nos iremos utilizar o Flask em nossos exercícios. Pela simplicidade e flexibilidade, ele se adequa melhor ao contexto de Data Science em menor escala.



<https://www.postman.com/>

O **POSTMAN** é uma ferramenta *open source* muito útil em que você pode criar o seu próprio console e catalogar os serviços utilizados. Iremos utilizá-la na aula de hoje para executar os serviços criados.

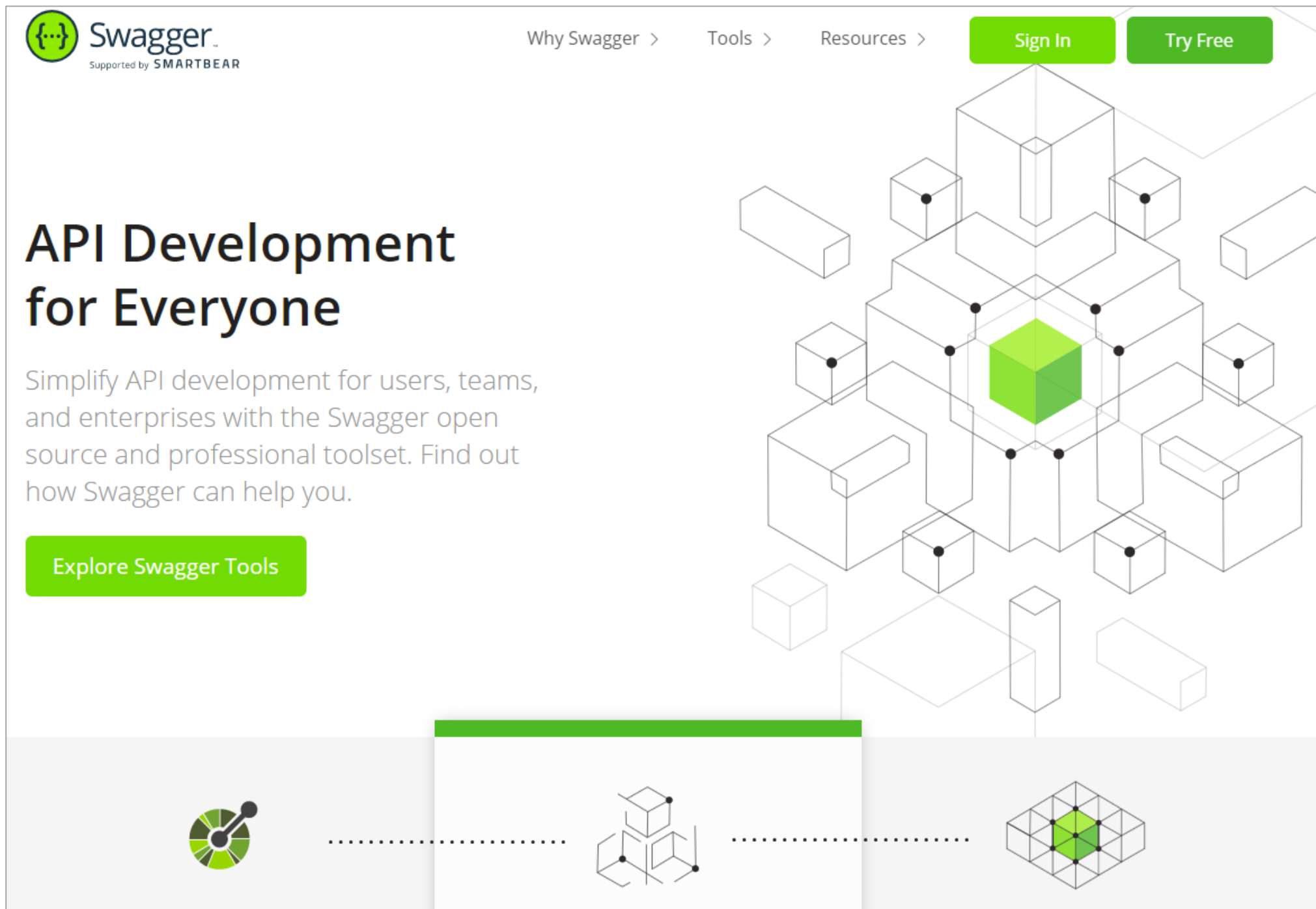
**CONSIDERAÇÕES
ADICIONAIS AO SE PROVER
UMA API PARA CONSUMO**



Dado que APIs – pela sua própria natureza – expõe serviços e dados de um organização, normalmente se faz necessário implementar algum mecanismo de **Controle de Acesso** / **Segurança** dos serviços. Isto geralmente implica, do lado do consumidor, na obtenção de um **Token de Acesso** para a utilização dos serviços.



Além dos requisitos de segurança, a disponibilização de APIs também implica normalmente em um **Controle da Utilização** (*billing*) dos serviços, para manutenção dos **SLAs** (*Service Level Agreements*) e monetização dos produtos em questão. Para o consumidor isto pode significar na assinatura de um plano com preços e volumes de consumo pré-acordados.



Em termos práticos para a construção de uma API, o **Swagger** é uma das ferramentas *open source* mais utilizadas no mercado para a disponibilização de serviços de uma maneira mais gráfica e intuitiva.

**AGORA VAMOS VER COMO
IMPLEMENTAR APIs COM
PYTHON**



EXERCÍCIO 02



- Abrir o arquivo '**cidades.py**' em seu editor de preferência. Iremos analisá-lo em conjunto.
- Abrir também o Notebook **api_flask_chamadas_v1.ipynb** no Jupyter.
- Iremos ver como são implementados serviços que fazem operações em um dicionário. Esta operação é feita em memória, mas é análoga a qualquer atividade que pode ser feita em uma base de dados ou outros serviços quaisquer.

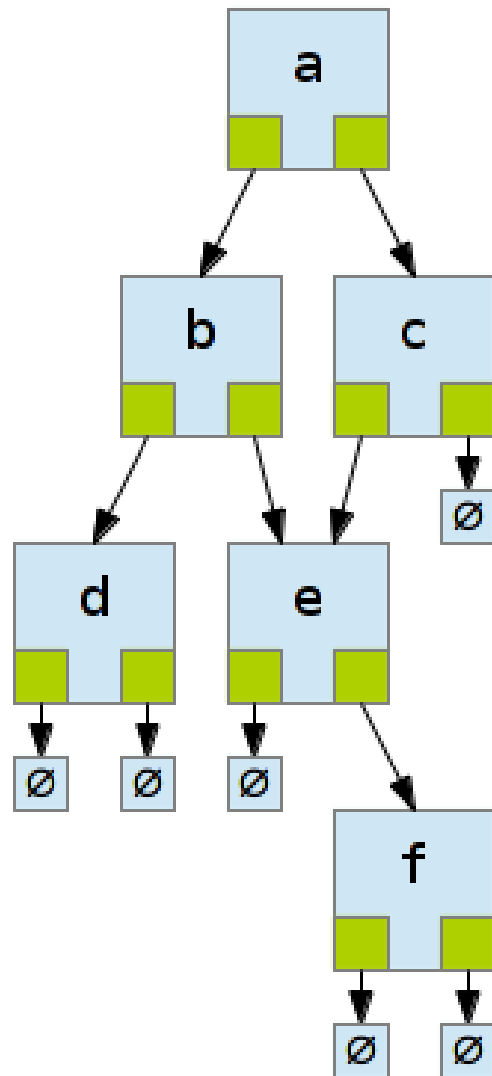
PERSISTÊNCIA DE MODELOS



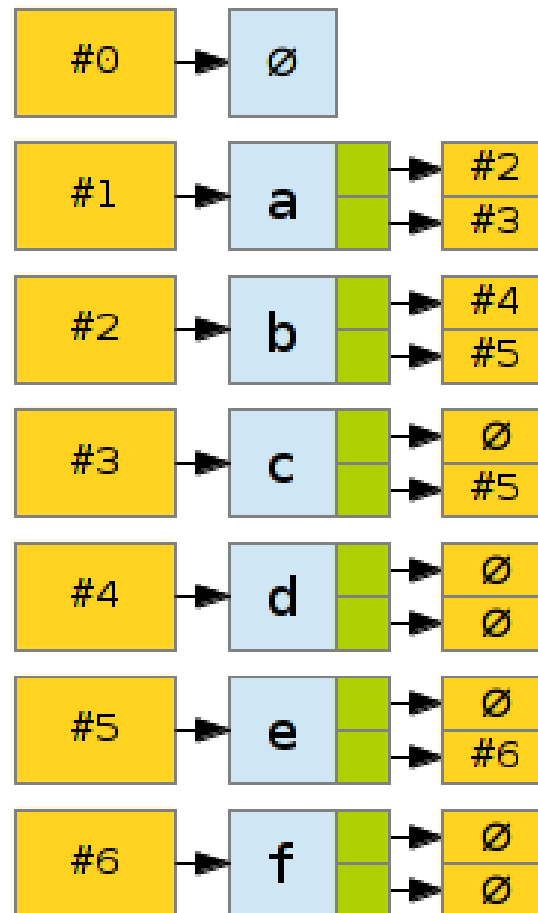
<https://docs.python.org/3/library/pickle.html>

O Pickle é uma biblioteca de SERIALIZAÇÃO de objetos do Python. Pode ser utilizado para gravar estruturas de dados, modelos preditivos, etc

Input Data



ser_serialize()



Output Format

```
node #1 { letter 'a';  
  left #2; right #3; }  
  
node #2 { letter 'b';  
  left #4; right #5; }  
  
node #3 { letter 'c';  
  left #5; right NULL; }  
  
node #4 { letter 'd';  
  left NULL; right NULL; }  
  
node #5 { letter 'e';  
  left NULL; right #6; }  
  
node #6 { letter 'f';  
  left NULL; right NULL; }
```

Your data as it appears
in Python's memory

```
['Is this the right room for an  
argument?', "No you haven't!",  
'When?', "No you didn't!", "You  
didn't!", 'You did not!', 'Ah!  
(taking out his wallet and paying)  
Just the five minutes.', 'You  
most certainly did not!', "Oh  
no you didn't!", "Oh no you  
didn't!", "Oh look, this isn't  
an argument!", "No it isn't!",  
"It's just contradiction!", 'It  
IS!', 'You just contradicted  
me!', 'You DID!', 'You did just  
then!', '(exasperated) Oh, this  
is futile!!', 'Yes it is!']
```

The pickle engine

Feed your Python
data to pickle.

Out comes the
pickled version of
your data.

Your
pickled
data



TL;DR: Never unpickle data from sources you don't trust. Otherwise you open your app up to a relatively simple way of remote code execution.

[HOME](#) [FEATURES](#) [PRICING](#) [TEAM](#) [BLOG](#) [CONTACT](#)

DANGEROUS PICKLES – MALICIOUS PYTHON SERIALIZATION

BY [EVAN SANGALINE](#) | OCTOBER 17, 2017

[Follow @sangaline](#) 283 [★ Star](#) 205

What's so dangerous about pickles?

Those pickles are very dangerous pickles. I literally can't begin to tell you how really dangerous they are. You have to trust me on that. It's important, Ok?

– "Explosive Disorder" by Pan Telaar

Before we get elbow deep in opcodes here, let's cover a little background. The Python standard library has a module called `pickle` that is

SEARCH

TAGS

[PYTHON \(19\)](#)[BROWSE ALL TAGS](#)

Joblib: running Python functions as pipeline jobs

Introduction



Joblib is a set of tools to provide **lightweight pipelining in Python**. In particular:

1. transparent disk-caching of functions and lazy re-evaluation (memoize pattern)
2. easy simple parallel computing

Joblib is optimized to be **fast** and **robust** on large data in particular and has specific optimizations for *numpy* arrays. It is **BSD-licensed**.

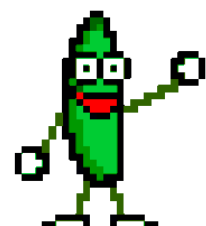
Documentation:	https://joblib.readthedocs.io
Download:	https://pypi.python.org/pypi/joblib#downloads
Source code:	https://github.com/joblib/joblib
Report issues:	https://github.com/joblib/joblib/issues

<https://joblib.readthedocs.io/en/latest/>

O Joblib também oferece funcionalidades de persistência e serialização de dados, além de ser excelente para a paralelização de tarefas e montagem de pipelines complexos.



EXERCÍCIO 03



- Vamos rodar o o Notebook **SklearnPipelines_persistence_vf.ipynb** para vermos o Pickle e Joblib em ação.

DigitalHouse >

DATA SCIENCE

APIs – Conceitos e Utilização