**Relatório <008> - <Prática: Web Scraping com Python p/ Ciência de Dados (II)>**

<Kawan Machado>

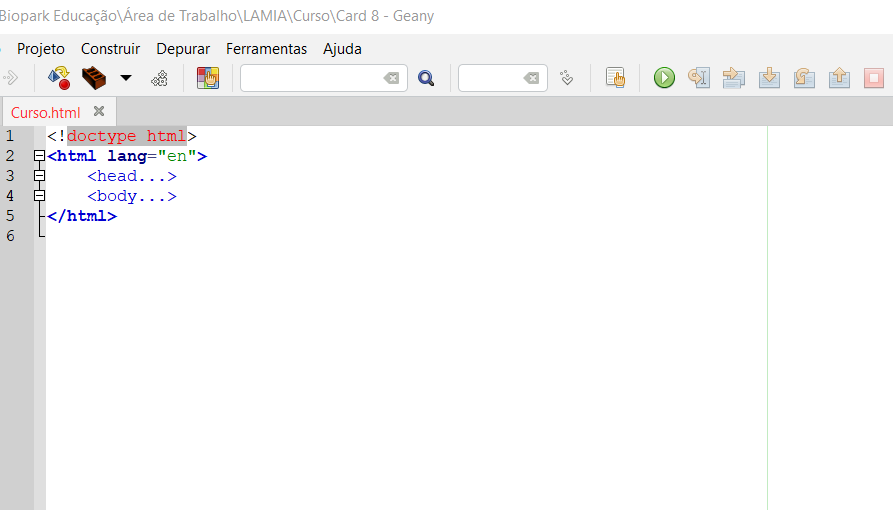
# Descrição da atividade

## Local HTML Scraping

### Basic HTML Structure, HTML Tags Explanation

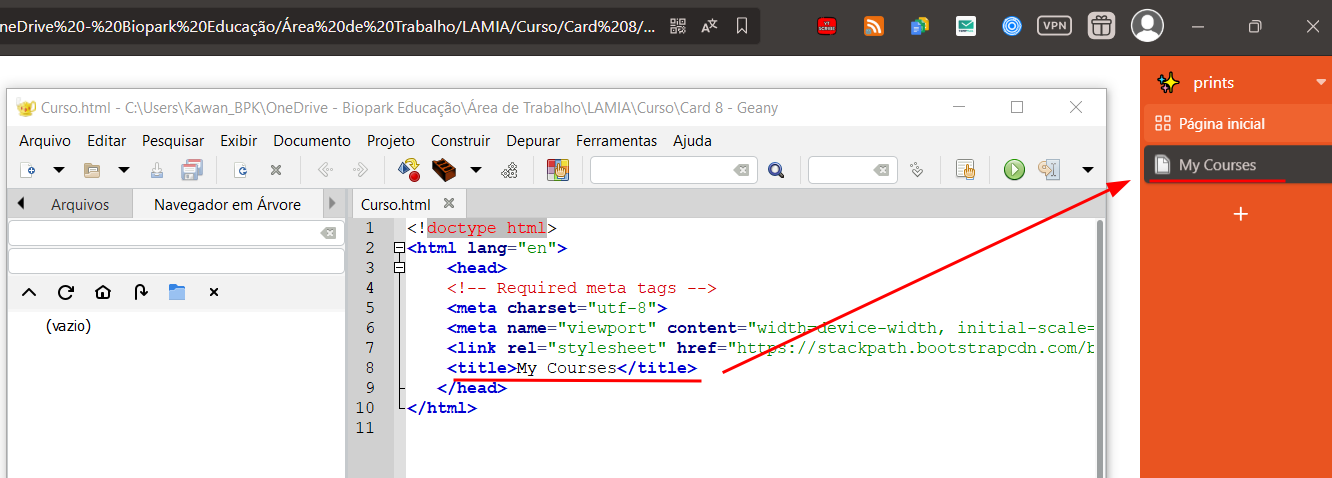
A estrutura padrão do HTML é primeiro definindo o tipo do documento, através do código ‘<! doctype html>’. Depois ele se subdivide, primeiro definindo onde estará contido o código html e sua língua (Português, Ingles, etc).

IMAGEM 1 – ESTRUTURA BASE

Elaborado pelo autor, 2025.

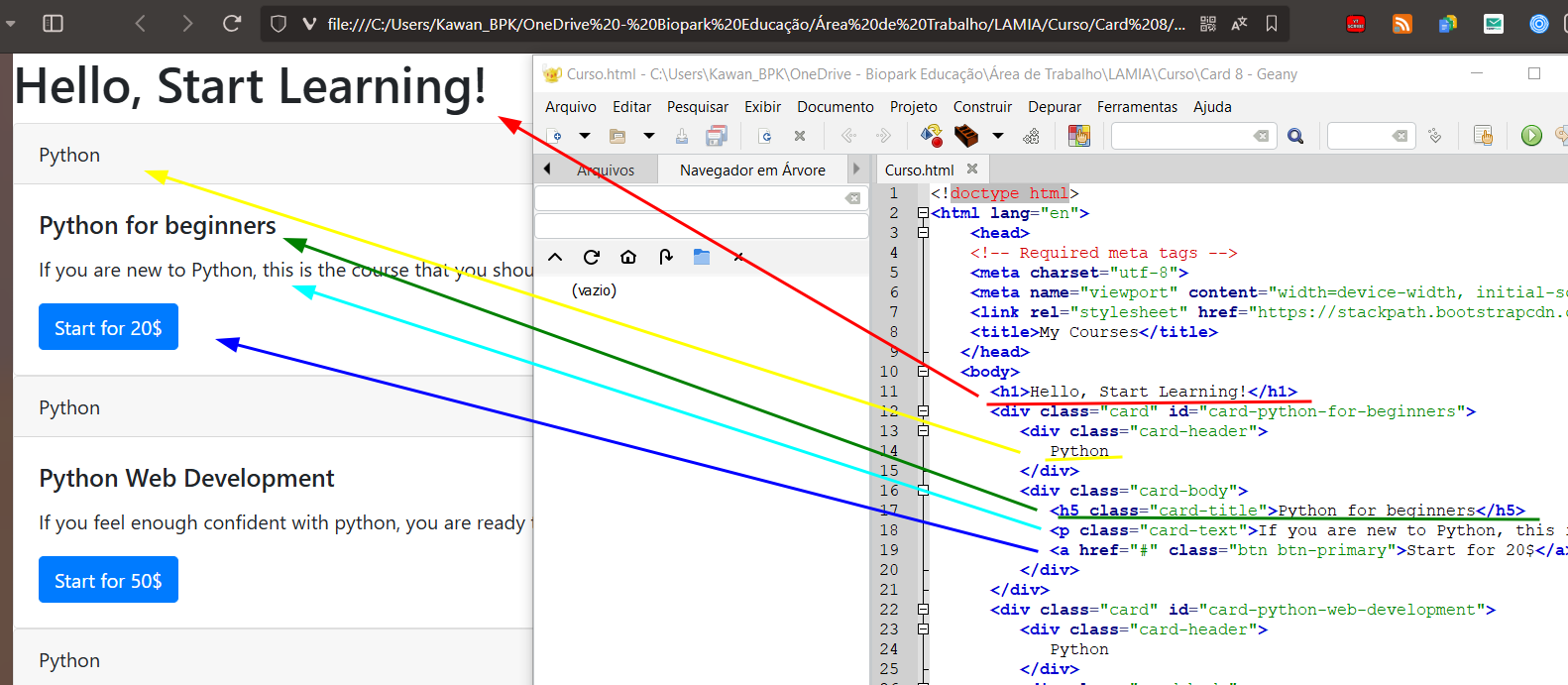
Como pode ser notado, o html funciona fazendo uso de tags, elas podem estar abrindo ‘< >’ e fechando ‘</ >. Tudo que está dentro daquele conjunto de tags irá executar naquele campo específico da página, por exemplo, a tag <tittle...> refere-se ao título que aparece em uma aba do site.

IMAGEM 2 – FUNCIONAMENTO DE TAGS

Elaborado pelo autor, 2025.

A tag ‘<body>’ é responsável por mostrar tudo que é visível no site, dentro do ‘body’ encontramos as tags ‘<div>’, elas basicamente vão definir a forma do site através do atributo ‘class’. Dentro da ‘div’ podemos destacar a tag ‘<a>’ que permite linkar outras páginas ou seja, ela é basicamente uma referência a outra página.

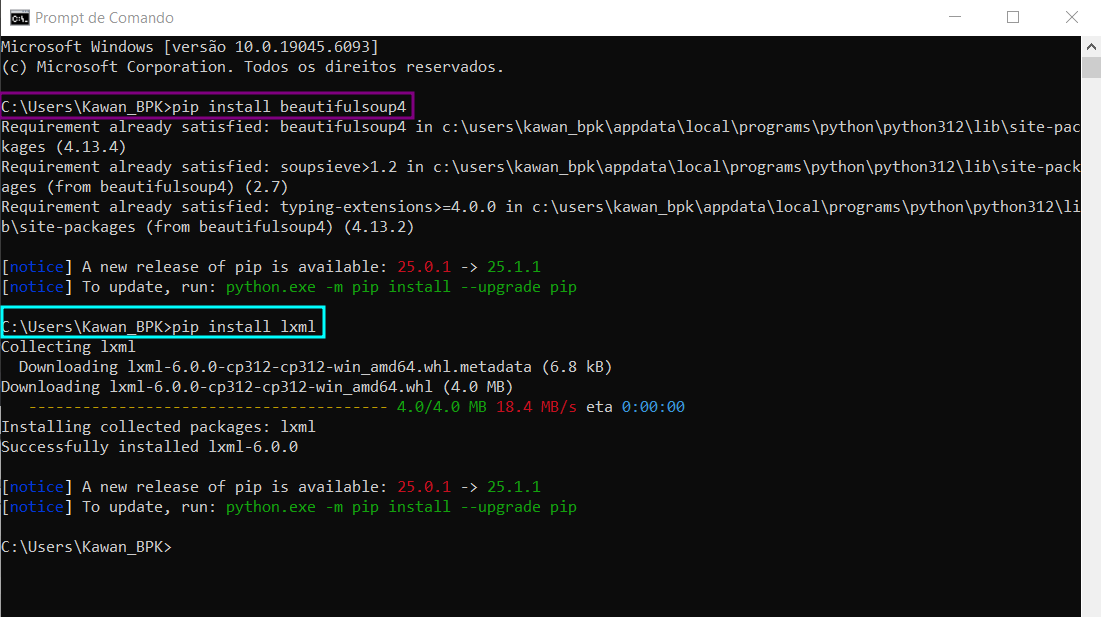
IMAGEM 3 – BODY E DIV

Elaborado pelo autor, 2025.

### Packages Installation

Para realizar um bom scrap, precisamos instalar duas bibliotecas para isso, a primeira é o beautifulsoup4 e a segunda é o lxml. Após a instalação, estamos prontos para iniciar um scrap.

IMAGEM 4 – INSTALAÇÃO DE BIBLIOTECAS

Elaborado pelo autor, 2025.

### Scraping Usage, Local files and Web Browser Inspect Tool

Depois de instalar a biblioteca, podemos baixar o arquivo html e trabalhar com ele de forma local, para isso precisamos importar o arquivo usando ‘with open()’, definir o comando ‘r’ para apenas leitura e atribuir um alias. Podemos usar o comando .read() para ler o html e depois disso aplicar o comando ‘BeatifulSoup()’ para visualizar o código html e conseguir trabalhar com o mesmo.

Com a página importada no código, começamos o scrap, para encontrar um elemento, utilizamos o comando .find() e .find\_all(). O primeiro retorna o primeiro item da busca enquanto o segundo retorna todos os itens da busca.

IMAGEM 5 – SCRAP

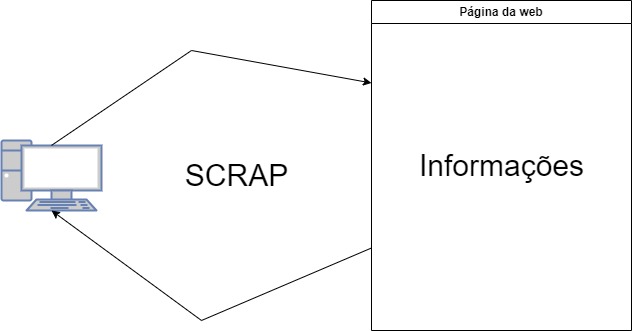
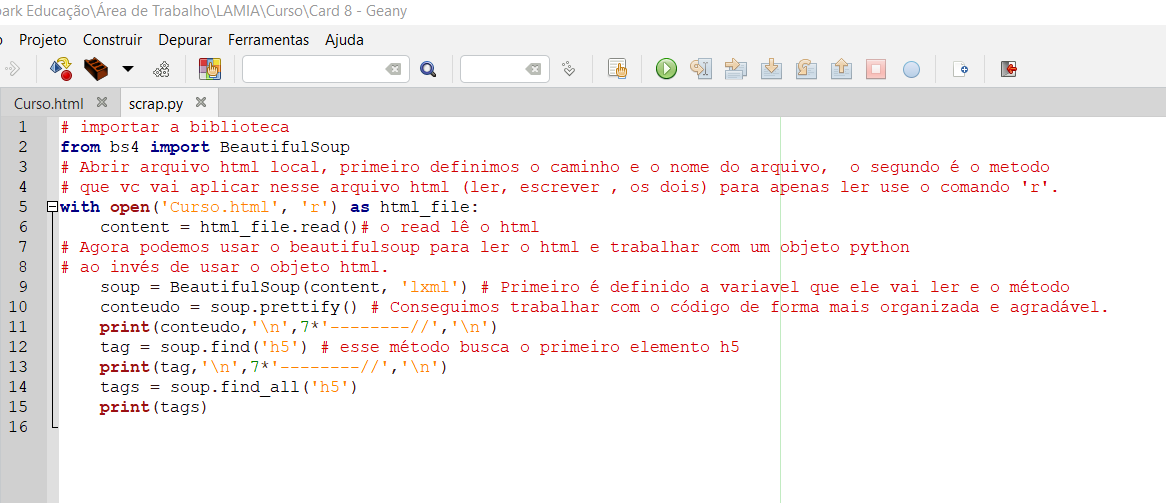
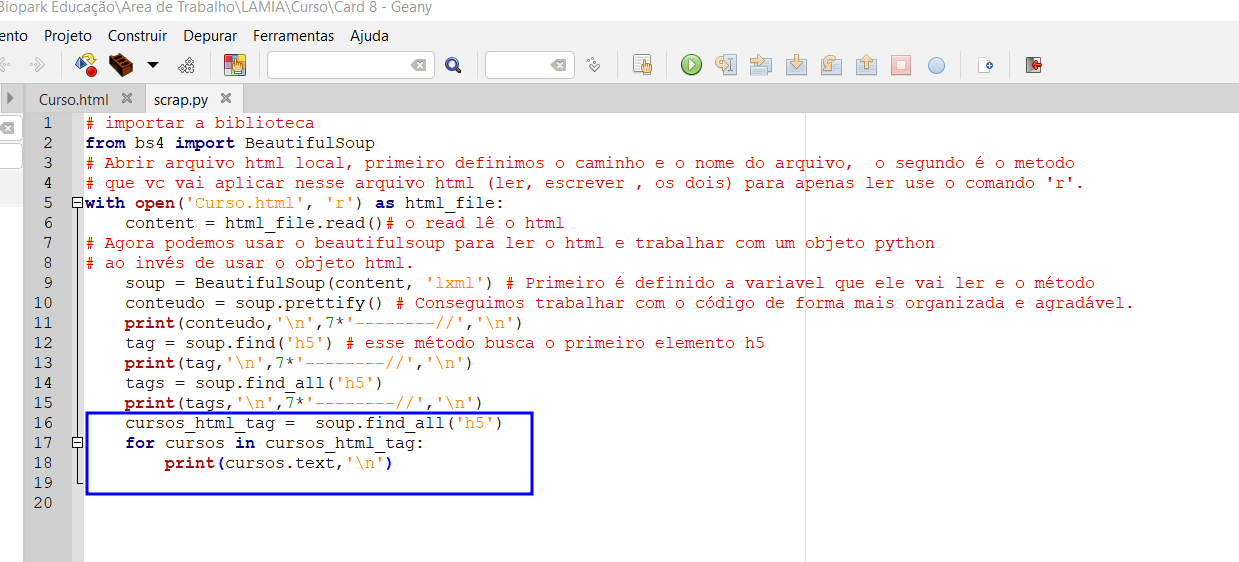
 Elaborado pelo autor, 2025.

IMAGEM 6 – FIND E FIND\_ALL

Elaborado pelo autor, 2025.

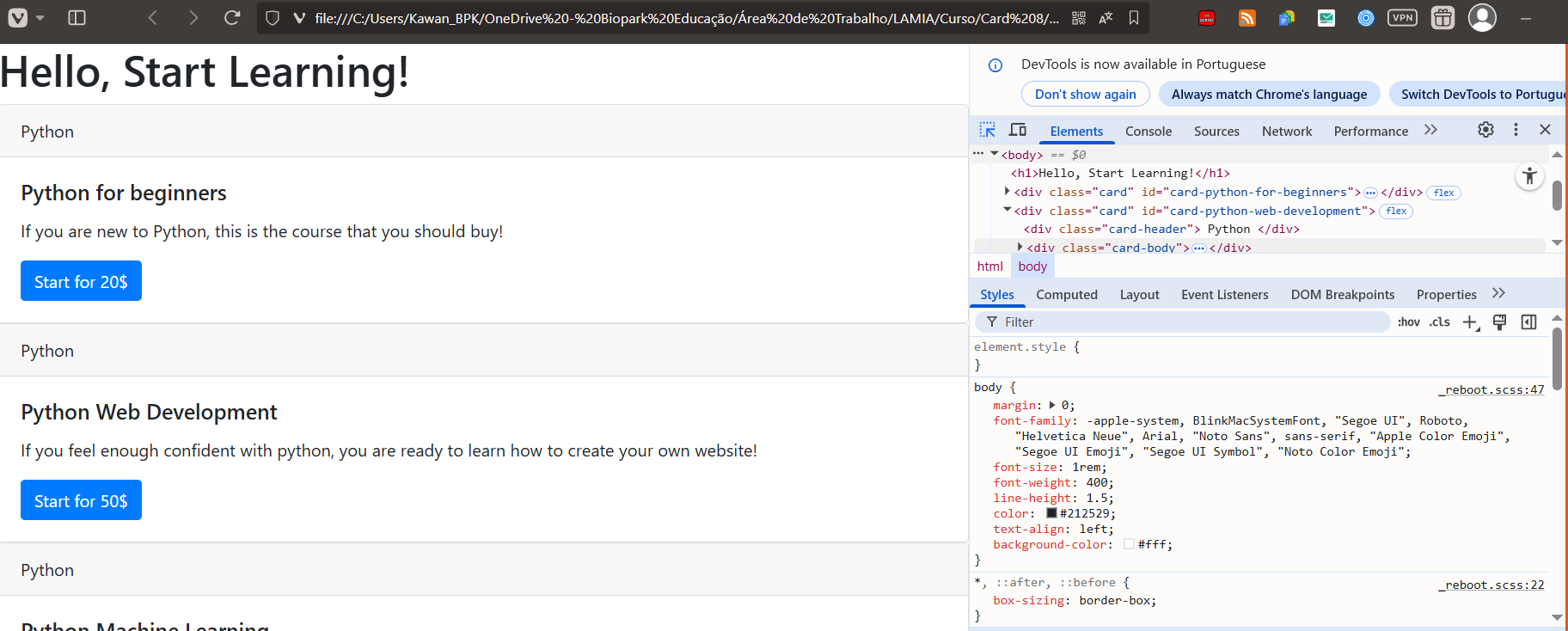
Nesse exemplo, sabemos que todos os elementos h5 contém os títulos dos cursos ofertados, podemos criar uma lista com todos os cursos ofertados, para isso armazenamos todos os resultados da busca ‘h5’ e iteramos sobre esse resultado um loop ‘for’, trazendo desse loop apenas os valores do tipo texto.

IMAGEM 7 – EXTRAINDO H5

Elaborado pelo autor, 2025.

Nesse exemplo, o arquivo html é simples e permite visualizar de forma fácil onde estão os elementos de cada código na página. No dia a dia é necessário fazer uso das ferramentas de análise das páginas para conseguir encontrar o elemento o qual se deseja extrair.

IMAGEM 8 – ANÁLISE DE UMA PÁGINA

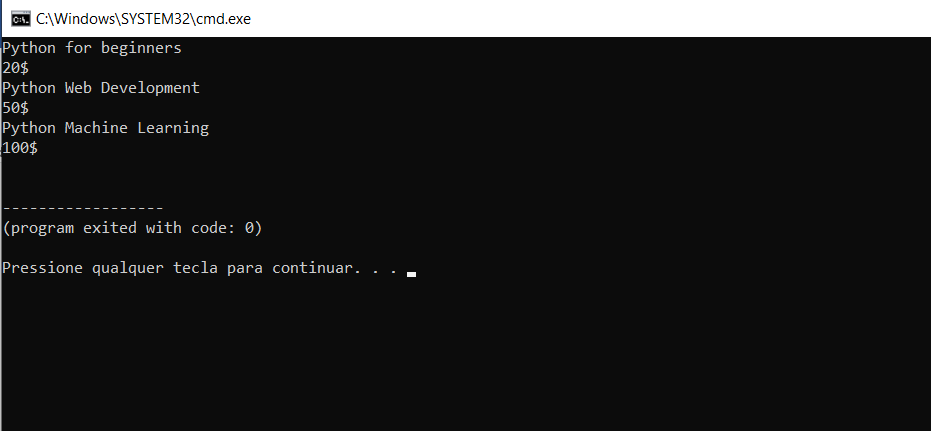
Elaborado pelo autor, 2025.

### Grab all Prices, Basic Scraping Project

Ainda nesse exemplo, digamos que queremos extrair o nome do curso e os preços, para isso devemos selecionar todas as ‘div’ de uma mesma classe, fazendo uso do comando ‘find\_all’ filtrando todas as ‘div’ do tipo ‘card’.

Com todas as ‘div’ contendo o valor que queremos extrair (curso e preço), podemos iterar sobre essa lista e extrair tanto os ‘textos’ contendo o nome do curso, quanto os textos contendo os valores de cada curso. O campo com o preço irá trazer todo o texto junto (exemplo, ‘*Start for 20$*’), para evitar isso, podemos usar o comando .split()[-1], o qual irá cortar o valor dos textos de trás para frente, dessa forma, mantendo apenas os valores numéricos.

IMAGEM 9 – PREÇOS E NOMES

Elaborado pelo autor, 2025.

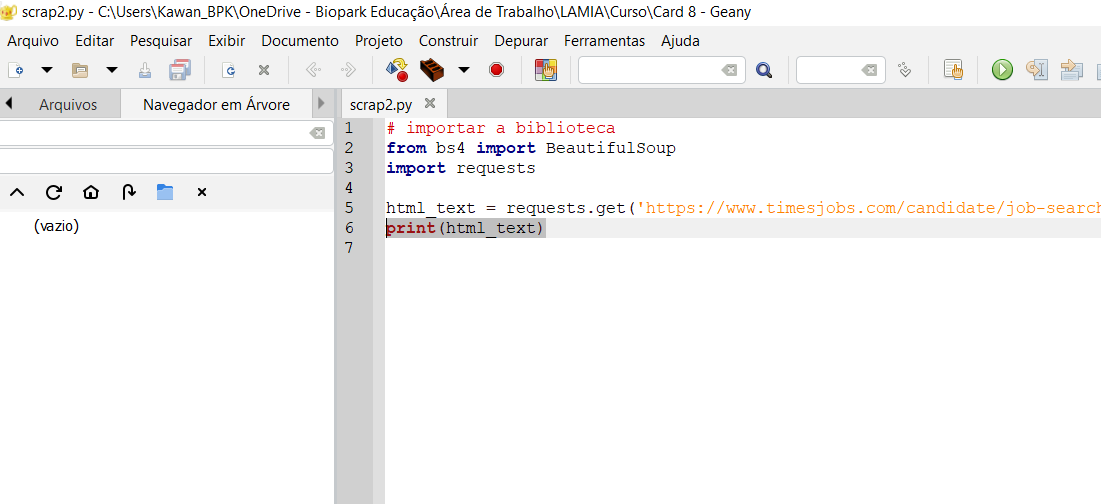
## Website Scraping

### Grab all Prices, Basic Scraping Project

Agora que conseguimos entender o básico, vamos começar a extrair informações de sites reais. Para isso vamos precisar da biblioteca ‘*requests’*, a qual podemos instalar usando o comando ‘*pip install requests’*.

Prosseguimos, o site que vamos realizar o scrap é o [*https://www.timesjobs.com*](https://www.timesjobs.com), vamos usar o método ‘get’, com o qual conseguimos pegar uma informação específica de um website como se fosse uma pessoa normal acessando a página, o que no nosso caso será o conjunto de vagas para a pesquisa da palavra ‘python’.

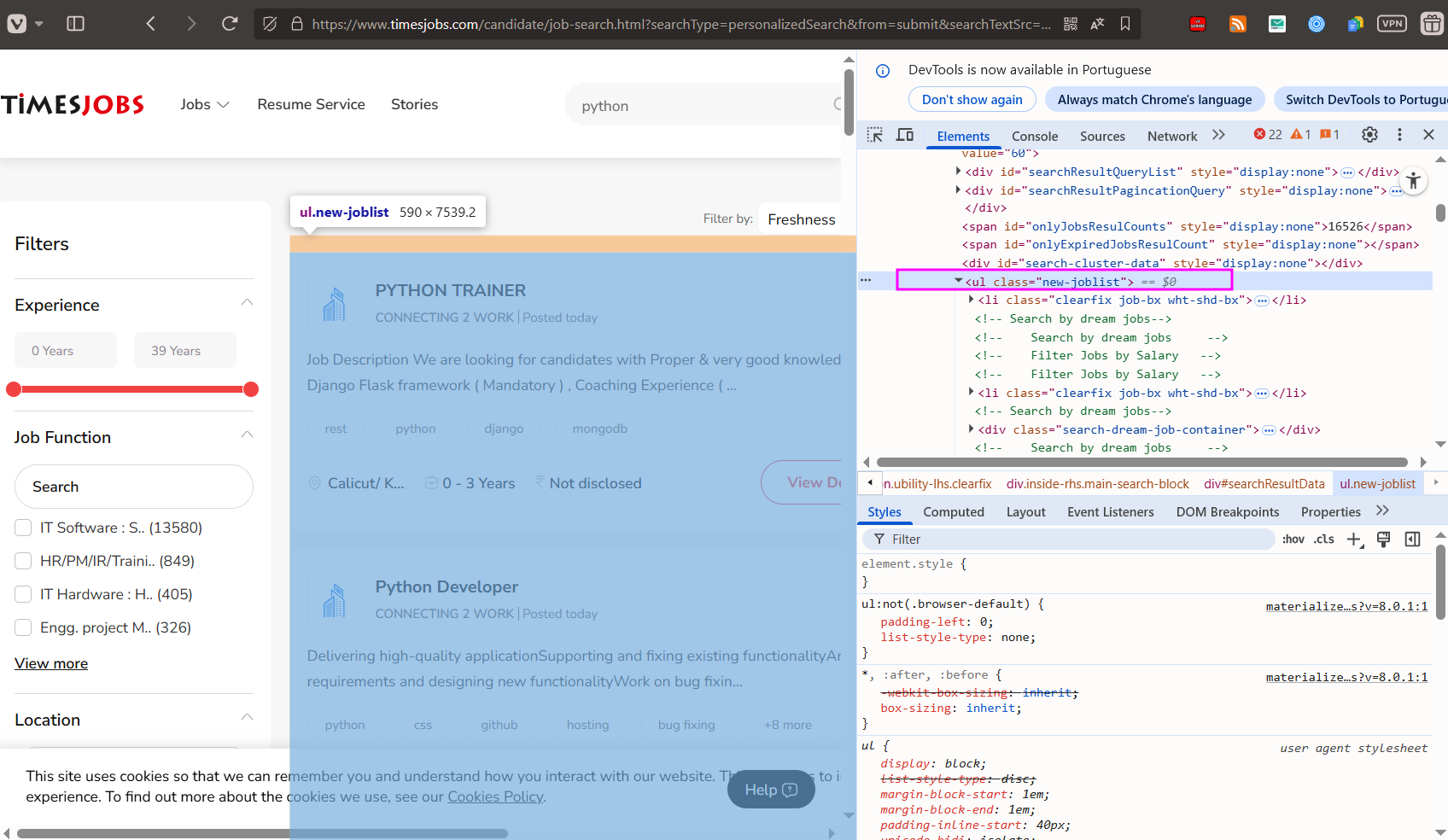
IMAGEM 10 – GET

Elaborado pelo autor, 2025.

### Scraping a Production Website, Best practices for pulling info

Para praticar como seria um scrap no mundo real, iremos usar o site https://www.timesjobs.com a qual se trata de um site de pesquisa de vagas, para o nosso exemplo iremos fazer um scrap da página com a pesquisa da palavra ‘python’ na busca.

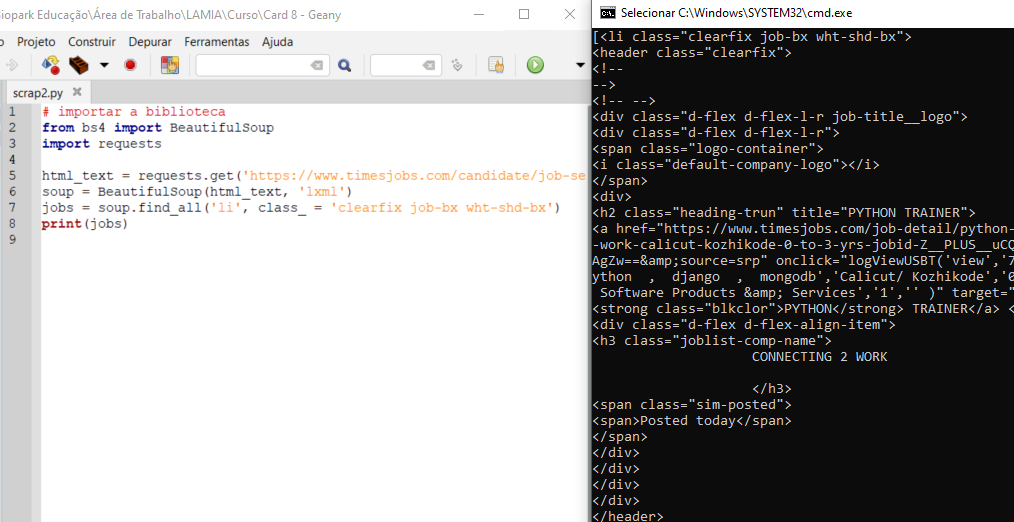
Primeiro, precisamos acessar a ferramenta de desenvolvedor do navegador para analisar o html da página, e encontramos a seção que contém todos os cards de pesquisa da vaga.

IMAGEM 11 – ENCONTRANDO ELEMENTOS  


Elaborado pelo autor, 2025.

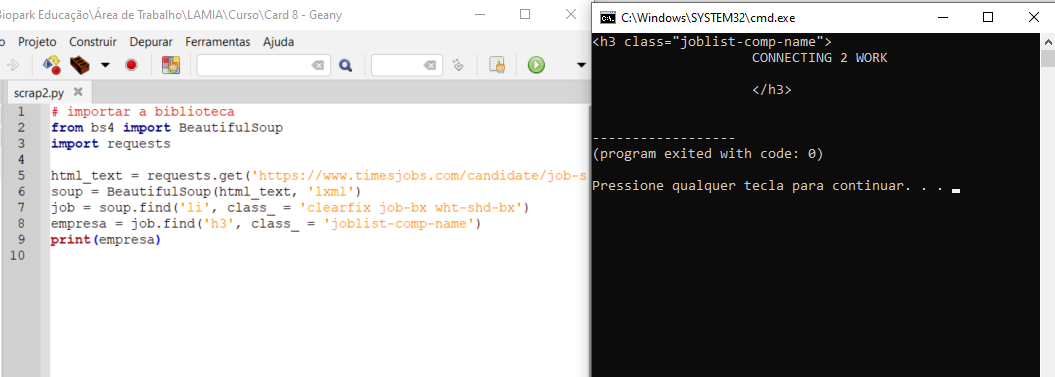
Observando o código conseguimos identificar que dentro do elemento ‘ul’ (unordered list) temos os elementos ‘li’ (list item) que representa cada card da busca de forma individual. Cada elemento ‘li’ pertence a classe ‘clearfix job-bx wht-shd-bx’, essa classe será inserida na busca para exibir todos os itens da lista.

IMAGEM 12 – BUSCA DENTRO DO CÓDIGO

Elaborado pelo autor, 2025.

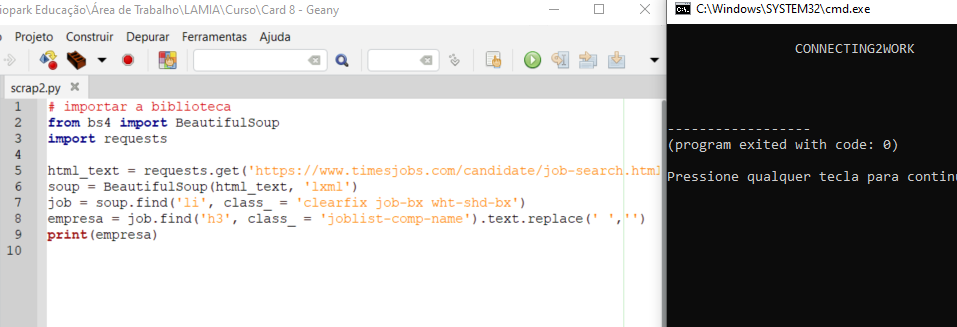
Para facilitar o alteramos o código para usar apenas ‘find’, com isso, garantimos que retorne apenas o primeiro resultado da busca, em seguida localizamos onde se encontra a informação do nome da empresa e fazemos a pesquisa da classe dentro da variável ‘job’.

IMAGEM 13 – BUSCANDO O NOME DA EMPRESA

Elaborado pelo autor, 2025.

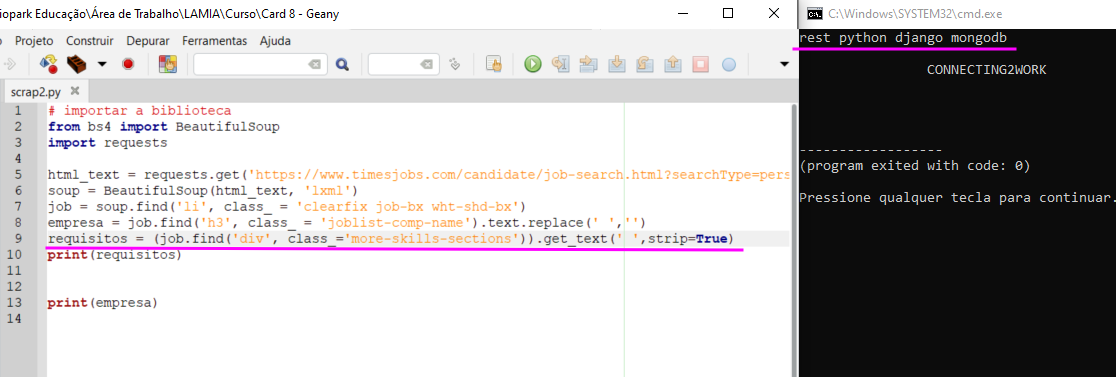
Podemos aplicar o método ‘.text’ junto com ‘.replace’. O primeiro irá trazer apenas o texto ao invés do elemento html todo e o segundo irá remover, no nosso caso, espaços por nenhum espaço.

IMAGEM 14 – MÉTODO TEXT E REPLACE

Elaborado pelo autor, 2025.

Aplicamos a mesma forma para encontrar os requisitos necessário para a vaga, a diferença, é que foi necessário encontrar a classe que continha as informações dos requisitos e aplicar a função ‘.get\_text()’.

IMAGEM 15 – REQUISITOS

Elaborado pelo autor, 2025.

Por fim, inserimos uma apresentação dos resultados usando f string com três aspas para imprimirmos as variáveis cada uma em uma linha. Abaixo o resultado final:

IMAGEM 16 – RESULTADO FINAL

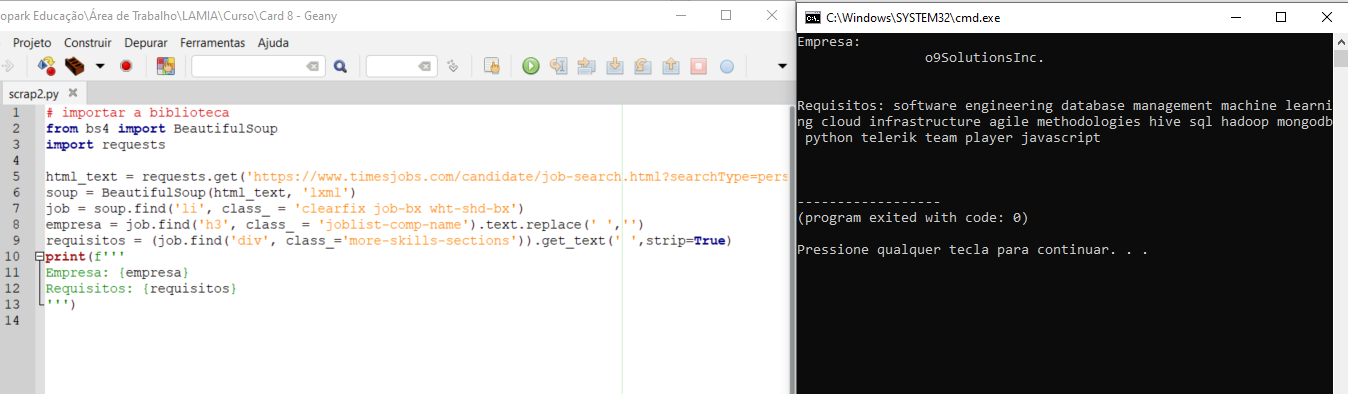
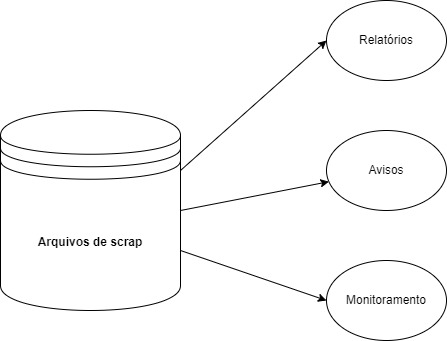
Elaborado pelo autor, 2025.

IMAGEM 17 – ALGUNS OBJETIVOS DO SCRAP

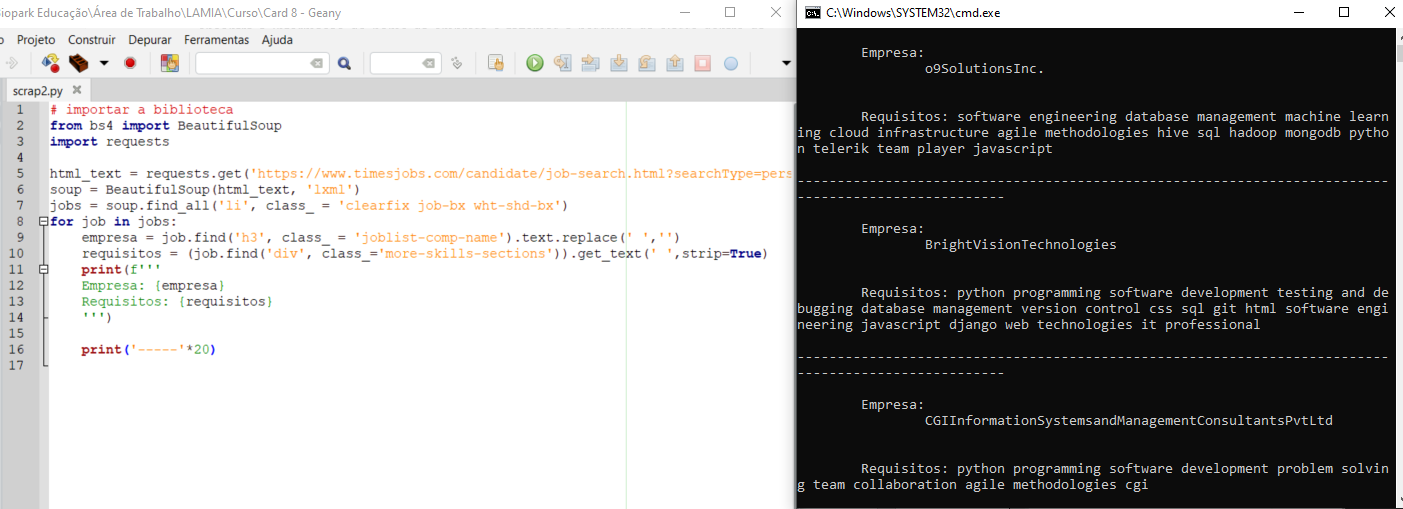
 Elaborado pelo autor, 2025.

### Looping through similar soup.find\_all() objects

Quando estamos tratando de um scrap, geralmente estamos buscando todas as informações contidas em determinado site, no nosso caso, queremos encontrar todos os trabalhos da página 1, para isso é necessário adaptar o código com um comando de loop ‘for’.

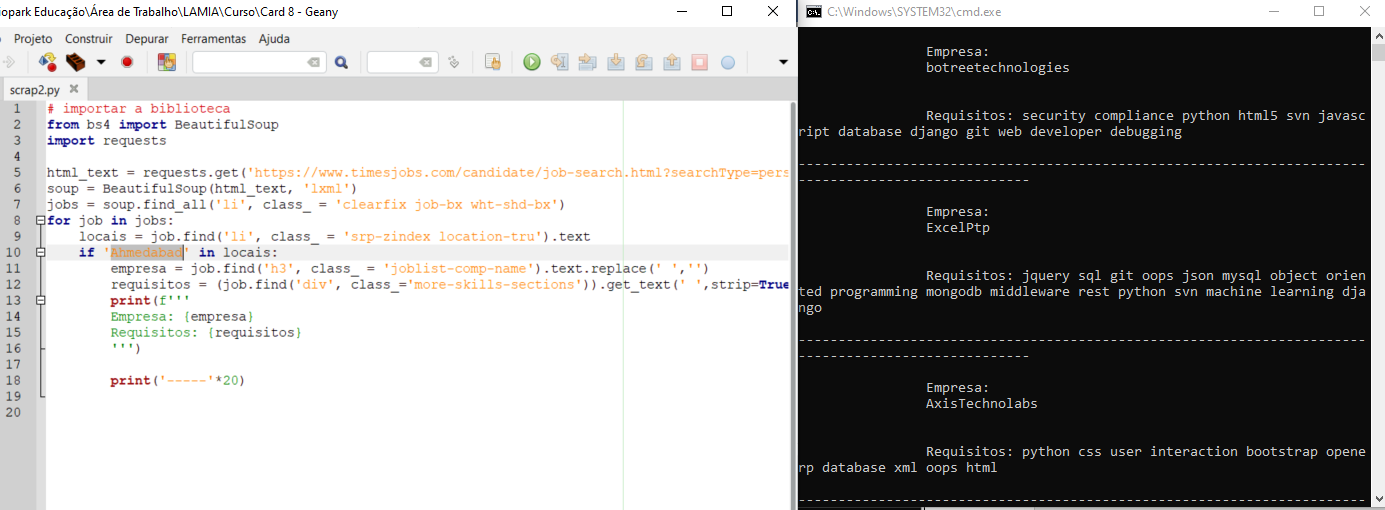
Primeiro alteramos o nome da varia´vel ‘job’ para ‘jobs’, depois modificamos o método de busca ‘*find*’ para ‘*find\_all’* e depois executamos um loop ‘for’ para iterar sobre ‘jobs’.

IMAGEM 18 – ITERANDO SOBRE JOBS

Elaborado pelo autor, 2025.

Além disso, podemos aplicar um filtro para por exemplo, buscar os trabalhos de determinada cidade apenas, isso diminui a busca e é uma forma eficiente de focar o scrap em uma informação específica. No nosso caso, vamos buscar apenas vagas do país ‘**Ahmedabad**’.

IMAGEM 19 – VAGAS APENAS DE UM LOCAL

Elaborado pelo autor, 2025.

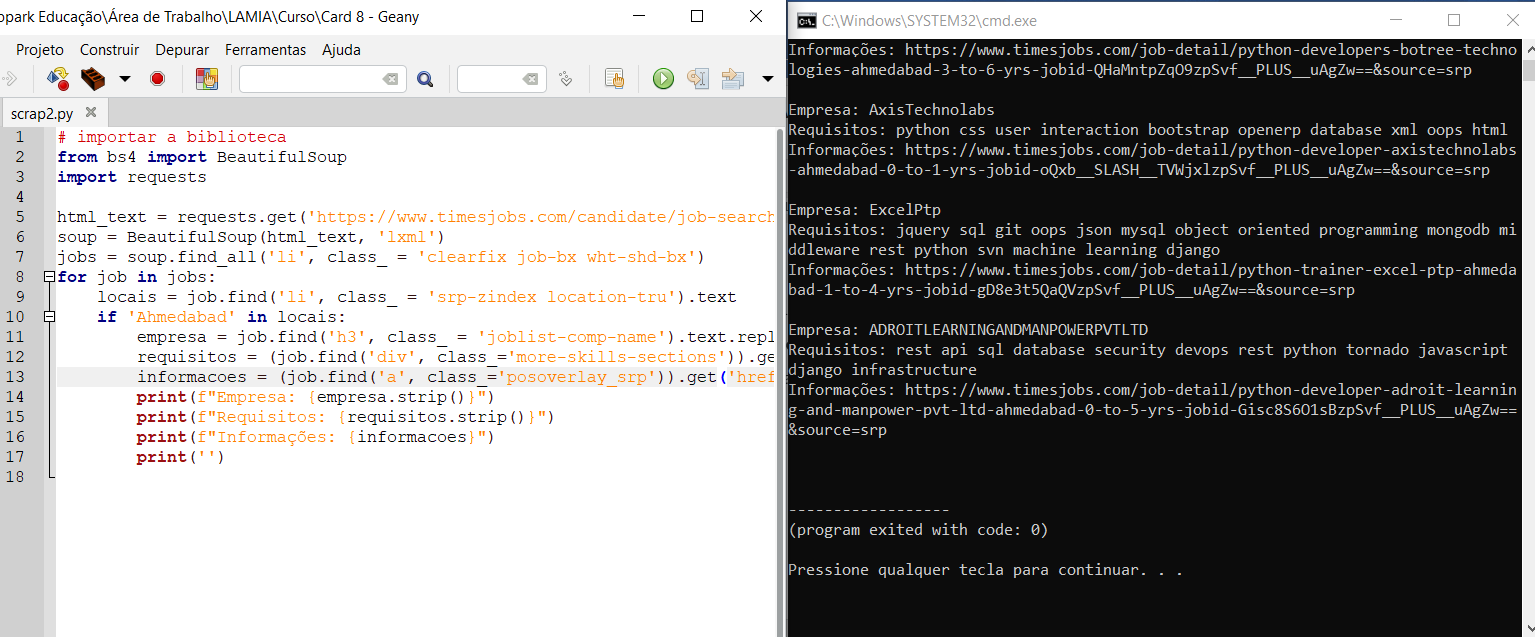
## Features addition

### Looping through similar soup.find\_all() objects

Agora que nosso código está organizado precisamos melhorar a exibição dos resultados, primeiro ajustamos a forma como os resultados são exibidos alterando o código ‘*print*’ e depois iremos inserir o link para a pessoa acessar as vagas diretamente do scrap.

Para ajustar o print apenas criamos um print para cada variável e inserimos o método *‘.strip()’*, o qual elimina espaços em branco antes e depois do texto. Depois buscamos o elemento que contenha o link e extraímos da mesma maneira que os anteriores, usando *‘find’* e o método *‘.get()’* para podermos extrair o link contido em ‘href’.

IMAGEM 20 – PRINT E LINK

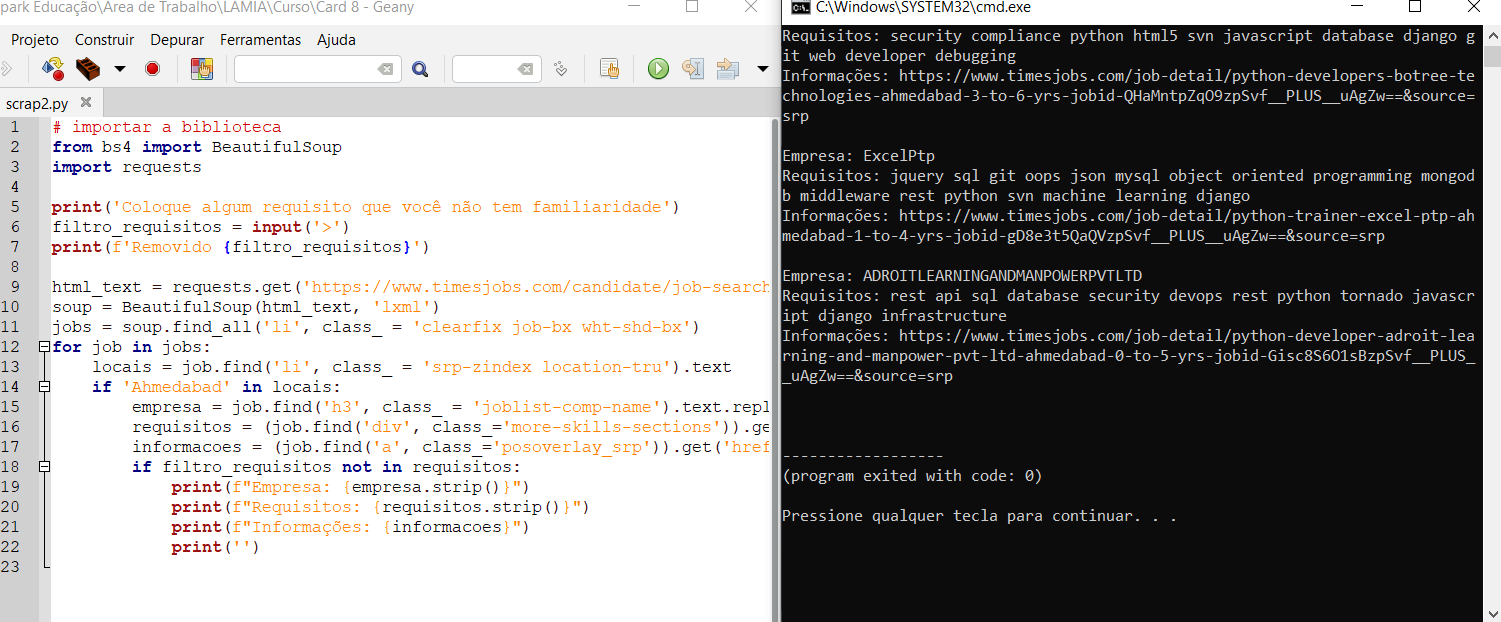
Elaborado pelo autor, 2025.

### Jobs Filtration by owned skills

Agora implementaremos a opção de o usuário poder remover resultados do scrap, para isso, precisamos implementar um comando *‘input()’* a qual irá receber a variável que está sendo removida da pesquisa de vagas, no nosso caso, um requisito.

Depois, inserimos um comando *‘if’* dentro do loop o qual funciona da seguitne maneira: caso o valor digitado no input não esteja dentro da variável *requisitos* imprime o resultado, caso contrário não imprime.

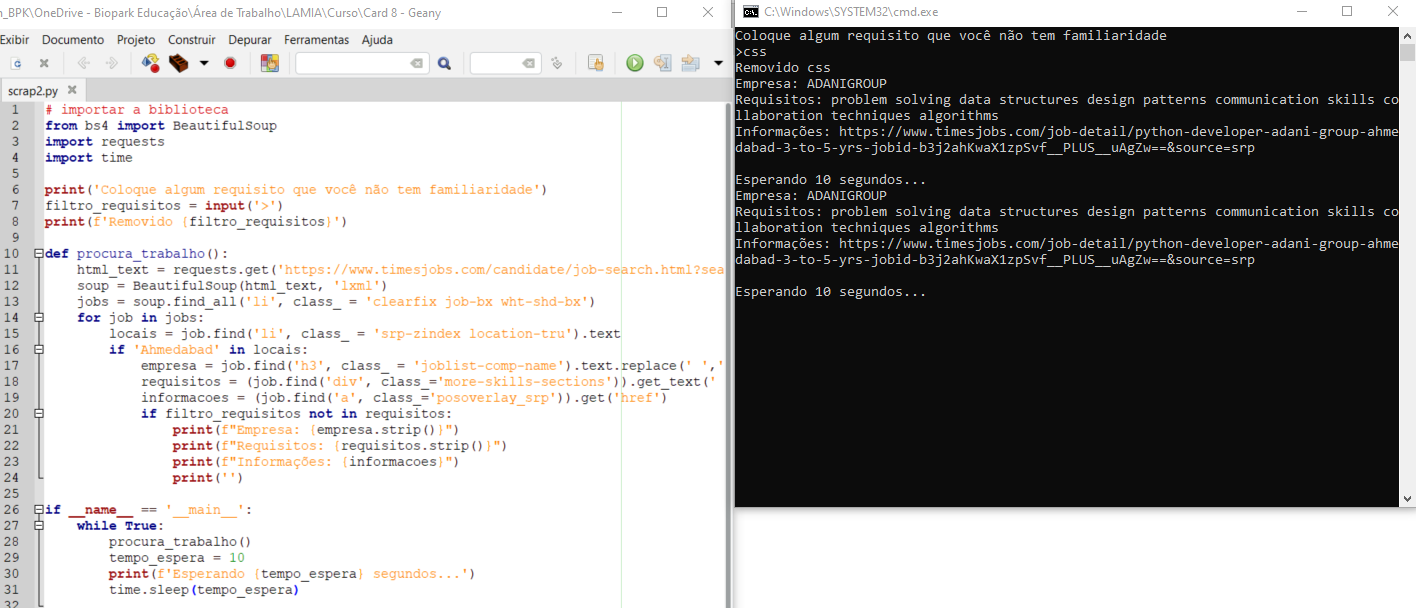
IMAGEM 21 – FILTRO DE REQUISITOS

Elaborado pelo autor, 2025.

### Setting up the Project to scrape every 10 minutes

Agora vamos ajustar para o código executar a cada 10 minutos uma busca no site. Para isso, precisamos importar a biblioteca ‘*time’*, criamos uma função para receber a parte do código que de fato faz o scrap no site, definimos esse arquivo ‘.py’ como arquivo principal através de *‘\_\_name\_\_ = \_\_main\_\_’* e criamos um loop *‘while True’*, o qual irá ficar executando infinitamente buscando de dez em dez segundos no site.

IMAGEM 22 – EXECUÇÃO A CADA 10 SEGUNDOS

Elaborado pelo autor, 2025.

### Storing the jobs paragraph in text files

Por último iremos armazenar os resultados da busca em um arquivo ‘.txt’, para isso precisamos alterar o loop ‘*for’*, para usar o método *enumerate()* e uma nova variável *index*. Depois, dentro do segundo *if,* inserimos o comando ‘*with open()’,* onde configuramos a pasta onde ele irá armazenar o arquivo *.txt* com o seu respectivo nome e a permissão para poder escrever no arquivo.

Por último usamos *‘as f:’* para definir que tudo isso é um arquivo, e colocamos dentro do *with* os prints configurados anteriormente, alterando para *f.write* e ao final exibimos uma mensagem de finalização do processo.

IMAGEM 23 – CRIANDO ARQUIVOS A CADA 10S

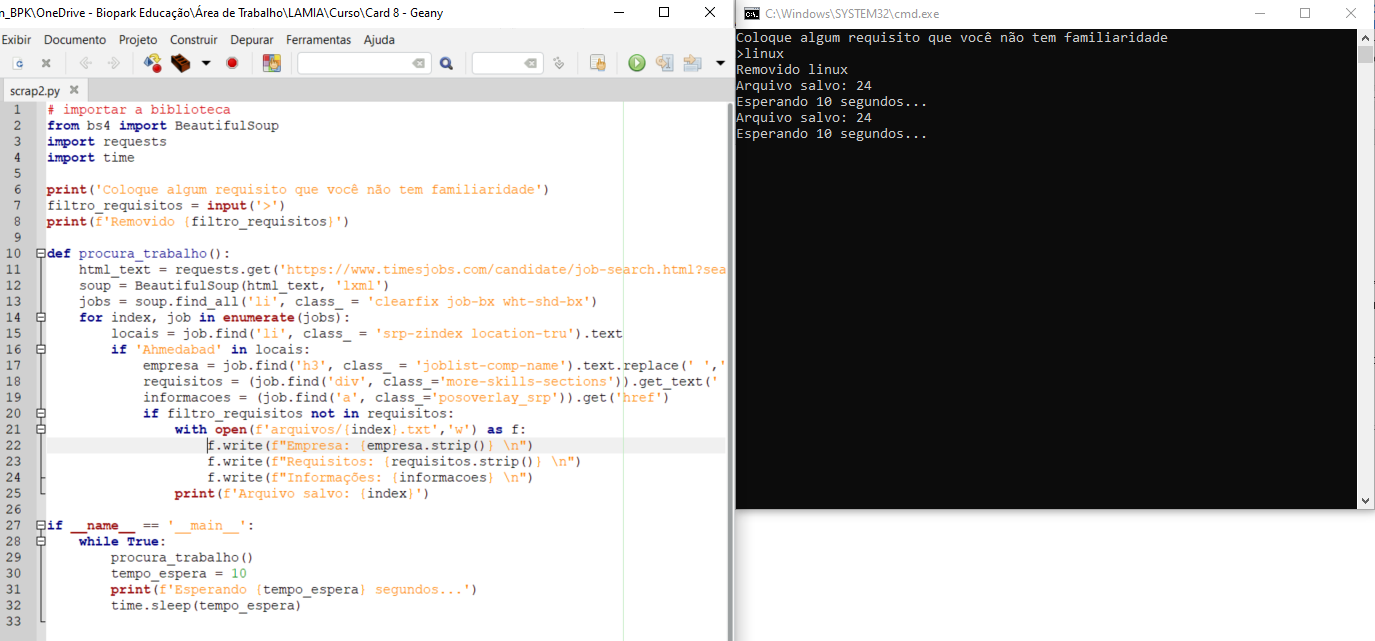
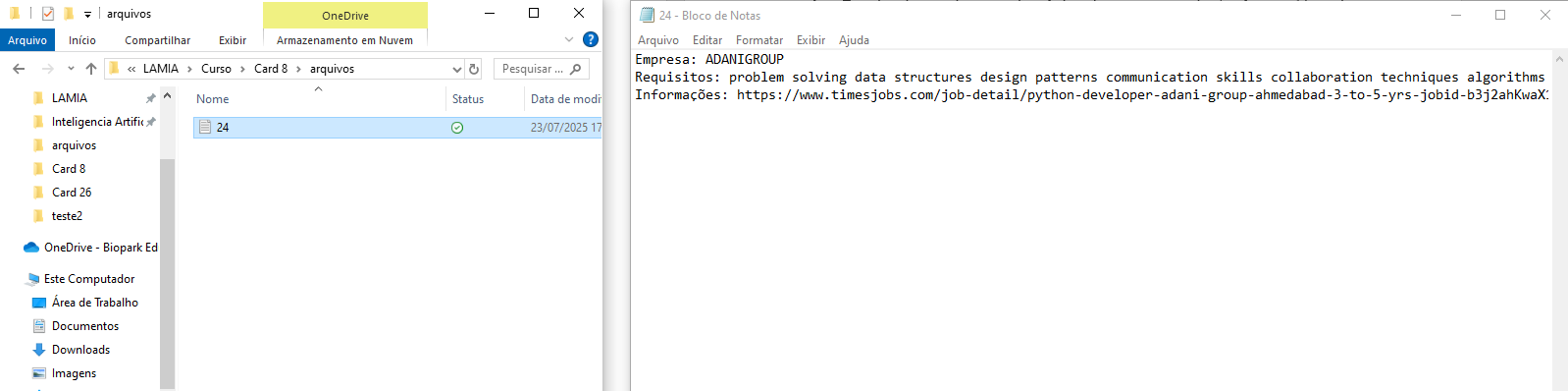
Elaborado pelo autor, 2025.

IMAGEM 24 – EXEMPLO DE ARQUIVO CRIADO

Elaborado pelo autor, 2025.

# Conclusões

O scrap é uma técnica muito importante no que tange trabalhos repetitivos, pois com ela conseguimos criar automações que vão diariamente buscar a informação em determinado site, de forma contínua e persistente, liberando seu usuário para outros processos.

Para ter um scrap efetivo é preciso saber e entender o código da página, logo, ter um conhecimento de *html* é imprescindível para conseguir programar bons códigos de scrap. Além disso, saber como usar a ferramenta de análise de página web, é extremamente importante, pois é ela que vai facilitar o entendimento do código e possibilitar entender a estrutura de como foi construído o site.

Para além desses pontos, é importante sempre estar praticando a escrita de códigos, durante esse processo, é uma boa prática, iniciar a construção do código de forma gradual ao invés de já tentar construir algo muito complexo. Também é importante notar que o exemplo dado no vídeo já permite aplicar para uma gama alta de opções, por exemplo, monitoradores de preço de um determinado item, monitoramento de vagas, monitoramento de postagens, etc.

# Referencias

As referências foram apenas os vídeos da atividade.