



# Cento Universitário UNA

Sistemas de Informação

Recuperação de Informação

Práticas de Laboratório

Wesley Dias Maciel

2019/01



Centro Universitário UNA  
Sistemas de Informação  
Recuperação de Informação  
Prática de Laboratório  
Wesley Dias Maciel  
2019/01

# Python



# Prática 07

## requests

O módulo “requests” permite realizar requisições HTTP. Instale o módulo requests, executando o comando “pip install requests”.

```
(maquinadebusca) C:\Users\wesle\Documents\Aulas\TecnologiasEmergentes\RecuperaçãoDeInformação\201901\práticas\RI - Python\código\maquinadebusca\controller>pip install requests
Collecting requests
  Downloading https://files.pythonhosted.org/packages/51/bd/23c926cd341ea6b7dd0b2a00aba99ae0f828be89d72b2190f27c11d4b7fb/requests-2.22.0-py2.py3-none-any.whl (57kB)
    |#####| 61kB 975kB/s
Collecting certifi>=2017.4.17 (from requests)
  Downloading https://files.pythonhosted.org/packages/60/75/f692a584e85b7eaba0e03827b3d51f45f571c2e793dd731e598828d380aa/certifi-2019.3.9-py2.py3-none-any.whl (158kB)
    |#####| 163kB 1.3MB/s
Collecting idna<2.9,>=2.5 (from requests)
  Downloading https://files.pythonhosted.org/packages/14/2c/cd551d81dbe15200be1cf41cd03869a46fe7226e7450af7a6545bfc474c9/idna-2.8-py2.py3-none-any.whl (58kB)
    |#####| 61kB 1.3MB/s
Collecting urllib3!=1.25.0,!1.25.1,<1.26,>=1.21.1 (from requests)
  Downloading https://files.pythonhosted.org/packages/39/ec/d93dfc69617a028915df914339ef66936ea976ef24fa62940fd86ba0326e/urllib3-1.25.2-py2.py3-none-any.whl (150kB)
    |#####| 153kB 1.6MB/s
Collecting chardet<3.1.0,>=3.0.2 (from requests)
  Downloading https://files.pythonhosted.org/packages/bc/a9/01ffebfb562e4274b6487b4bb1dddec7ca55ec7510b22e4c51f14098443b8/chardet-3.0.4-
```

## BeautifulSoup

O módulo “BeautifulSoup” permite tratar arquivos no formato HTML e XML. Instale o módulo BeautifulSoup, executando o comando “pip install beautifulsoup4”.



The screenshot shows the Visual Studio Code interface. The Explorer panel on the left shows a project named 'maquinadebusca' with a subfolder 'controller' containing a file 'ColetorController.py'. The Terminal panel at the bottom shows the following output:

```
(maquinadebusca) C:\Users\wesle\Documents\Aulas\TecnologiasEmergentes\RecuperaçãoDeInformação\201901\práticas\RI - Python\código\maquinadebusca\controller>pip install beautifulsoup4
Collecting beautifulsoup4
  Downloading https://files.pythonhosted.org/packages/1d/5d/3260694a59df0ec52f8b4883f5d23b130bc237602a1411fa670eae12351e/beautifulsoup4-4.7.1-py3-none-any.whl (94kB)
    | 102kB 1.1MB/s
Collecting soupsieve>=1.2 (from beautifulsoup4)
  Downloading https://files.pythonhosted.org/packages/b9/a5/7ea40d0f8676bde6e464a6435a48bc5db09b1a8f4f06d41dd997b8f3c616/soupsieve-1.9.1-py2.py3-none-any.whl
Installing collected packages: soupsieve, beautifulsoup4
Successfully installed beautifulsoup4-4.7.1 soupsieve-1.9.1

(maquinadebusca) C:\Users\wesle\Documents\Aulas\TecnologiasEmergentes\RecuperaçãoDeInformação\201901\práticas\RI - Python\código\maquinadebusca\controller>
```

A partir da prática 06, altere o código do coletor, como apresentado abaixo:

```
from flask import Flask, jsonify
import requests
from bs4 import BeautifulSoup

app = Flask(__name__)

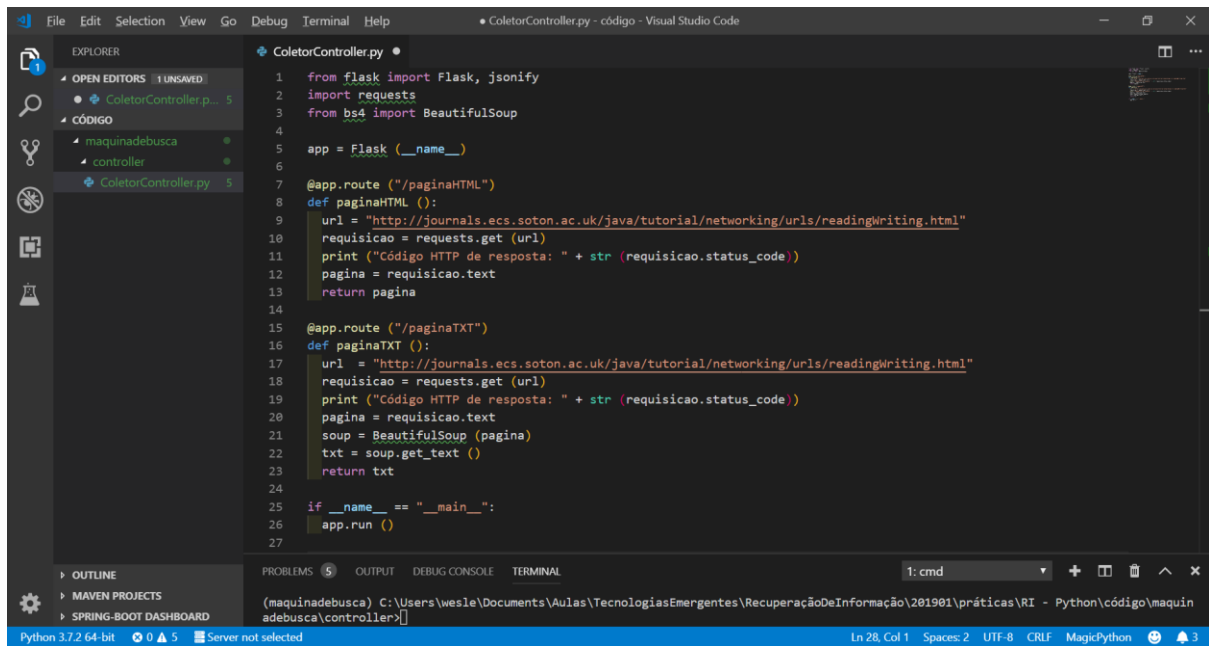
@app.route("/paginaHTML")
def paginaHTML():
    url =
"http://journals.ecs.soton.ac.uk/java/tutorial/networking/urls/readingWriting.html"
    requisicao = requests.get(url)
    print("Código HTTP de resposta: " + str(requisicao.status_code))
    pagina = requisicao.text
    return pagina

@app.route("/paginaTXT")
def paginaTXT():
    url =
"http://journals.ecs.soton.ac.uk/java/tutorial/networking/urls/readingWriting.html"
    requisicao = requests.get(url)
    print("Código HTTP de resposta: " + str(requisicao.status_code))
    pagina = requisicao.text
```



```
soup = BeautifulSoup (pagina)
txt = soup.get_text ()
return txt

if __name__ == "__main__":
    app.run ()
```



Inicie o servidor, executando o comando “python ColetorController.py”.



Centro Universitário UNA  
Sistemas de Informação  
Recuperação de Informação  
Prática de Laboratório  
Wesley Dias Maciel  
2019/01

```
1 from flask import Flask, jsonify
2 import requests
3 from bs4 import BeautifulSoup
4
5 app = Flask(__name__)
6
7 @app.route("/paginaHTML")
8 def paginaHTML():
9     url = "http://journals.ecs.soton.ac.uk/java/tutorial/networking/urls/readingWriting.html"
10    requisicao = requests.get(url)
11    print("Código HTTP de resposta: " + str(requisicao.status_code))
12    pagina = requisicao.text
13    return pagina
14
15 @app.route("/paginaTXT")
16 def paginaTXT():
17     url = "http://journals.ecs.soton.ac.uk/java/tutorial/networking/urls/readingWriting.html"
18     requisicao = requests.get(url)
19     print("Código HTTP de resposta: " + str(requisicao.status_code))
20     pagina = requisicao.text
21     soup = BeautifulSoup(pagina)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

1: python

(maquinadebusca) C:\Users\wesle\Documents\Aulas\TecnologiasEmergentes\RecuperaçãoDeInformação\201901\práticas\RI - Python\código\maquinadebusca\controller>python ColetorController.py

\* Serving Flask app "ColetorController" (lazy loading)

\* Environment: production

WARNING: This is a development server. Do not use it in a production deployment.

Use a production WSGI server instead.

\* Debug mode: off

\* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)

Teste a API nos navegadores Chrome ou FireFox.

## Reading from and Writing to a URLConnection [notes](#)

If you've successfully used `openConnection()` to initiate communications with a URL, then you have a reference to a `URLConnection` object. The `URLConnection` class contains many methods that let you communicate with the URL over the network. `URLConnection` is an HTTP-centric class--many of its methods are useful only when working with HTTP URLs. However, most URL protocols let you read from and write to the connection so this page shows you how to both read from and write to a URL through a `URLConnection` object.

### Reading from a URLConnection [notes](#)

The following program performs the same function as the program shown in [Reading Directly from a URL](#). However, rather than opening a stream directly from the URL, this program explicitly opens a connection to the URL, gets an input stream on the connection, and reads from the input stream:

```
import java.net.*;
import java.io.*;

class ConnectionTest {
    public static void main(String[] args) {
        try {
            URL yahoo = new URL("http://www.yahoo.com/");
            URLConnection yahooConnection = yahoo.openConnection();
            DataInputStream dis = new DataInputStream(yahooConnection.getInputStream());
            String inputLine;

            while ((inputLine = dis.readLine()) != null) {
                System.out.println(inputLine);
            }
            dis.close();
        } catch (MalformedURLException me) {
            System.out.println("MalformedURLException: " + me);
        } catch (IOException ioe) {
            System.out.println("IOException: " + ioe);
        }
    }
}
```



Centro Universitário UNA  
Sistemas de Informação  
Recuperação de Informação  
Prática de Laboratório  
Wesley Dias Maciel  
2019/01

127.0.0.1:5000/paginaTXT

127.0.0.1:5000/paginaTXT

Reading from and Writing to a URLConnection Working with URLs Reading from and Writing to a URLConnection (notes) If you've successfully used openConnection() to initiate communications with a URL, then you have a reference to a URLConnection object. The URLConnection class contains many methods that let you communicate with the URL over the network. URLConnection is an HTTP-centric class--many of its methods are useful only when working with HTTP URLs. However, most URL protocols let you read from and write to the connection so this page shows you how to both read from and write to a URL through a URLConnection object. Reading from a URLConnection(notes) The following program performs the same function as the program shown in Reading Directly from a URL. However, rather than opening a stream directly from the URL, this program explicitly opens a connection to the URL, gets an input stream on the connection, and reads from the input stream: import java.net.\*; import java.io.\*; class ConnectionTest { public static void main(String[] args) { try { URL yahoo = new URL("http://www.yahoo.com/"); URLConnection yahooConnection = yahoo.openConnection(); DataInputStream dis = new DataInputStream(yahooConnection.getInputStream()); String inputLine; while ((inputLine = dis.readLine()) != null) { System.out.println(inputLine); } dis.close(); } catch (MalformedURLException me) { System.out.println("MalformedURLException: " + me); } catch (IOException ioe) { System.out.println("IOException: " + ioe); } } } The output from this program should be identical to the output from the program that opens a stream directly from the URL. You can use either way to read from a URL. However, sometimes reading from a URLConnection instead of reading directly from a URL might be more useful to you as you can use the URLConnection object for other tasks (like writing to the URL) at the same time. Again if, instead of output from the program, you see the following error message: IOException: java.net.UnknownHostException: www.yahoo.com you may have to set the proxy host so that the program can find the www.yahoo.com server. Writing to a URLConnection(notes) Many HTML pages contain forms--text fields and other GUI objects that let you enter data to the server. After you type in the required information and initiate the query by clicking on a button, the Web browser you're using writes the data to the URL over the network. At the other end, a (usually) cgi-bin script on the server the data, processes it, and then sends you back a response, usually in the shape of a new HTML page. This scenario is often used to support searching. Many cgi-bin scripts use the POST METHOD for reading the data from the client. Thus writing to a URL is often known as posting to a URL. Server-side scripts that use the POST METHOD read from their standard input. Note: Some server-side cgi-bin scripts use the GET METHOD to read your data. The POST METHOD is quickly making the GET METHOD obsolete because it's more versatile and has no limitations on the amount of data that can be sent through the connection. Your Java programs can also interact with cgi-bin scripts on the server-side. They simply must be able to write to a URL, thus providing data to the server. Your program can do this by following these steps: Create a URL. Open a connection to the URL. Get an output stream from the connection. This output stream is connected to the standard output stream of the cgi-bin script on the server. Write to the output stream. Close the output stream. Hassan Schroeder, a member of the Java team, wrote a small cgi-bin script, named backwards, and made it available at our Web site, java.sun.com. You can use this script to test the following example program. If for some reason you can't get to our Web site; you can put the script somewhere on your network, name it backwards, and test the program locally. The script at our Web site reads a string from its standard input, reverses the string, and writes the result to its standard output. The script requires input of the following form: string=string\_to\_reverse, where string\_to\_reverse is the string whose characters you want displayed in reverse order. Here's an example program that runs the backwards script over the network through a URLConnection: import java.io.\*; import java.net.\*; public class ReverseTest { public static void main(String[] args) { try { if (args.length != 1) { System.err.println("Usage: java ReverseTest string\_to\_reverse"); System.exit(1); } String stringToReverse = URLEncoder.encode(args[0]); URL url = new URL("http://java.sun.com/cgi-bin/backwards"); URLConnection connection = url.openConnection(); PrintStream outStream = new PrintStream(connection.getOutputStream()); outStream.println("string="+stringToReverse); outStream.close(); DataInputStream inStream = new DataInputStream(connection.getInputStream()); String inputLine; while ((inputLine = inStream.readLine()) != null) { System.out.println(inputLine); } inStream.close(); } catch (MalformedURLException me) { System.err.println("MalformedURLException: " + me); } catch (IOException ioe) { System.err.println("IOException: " + ioe); } } } Let's examine the program and see how it works. First, the program processes its command line arguments: if (args.length != 1) { System.err.println("Usage: java ReverseTest string\_to\_reverse"); System.exit(1); } String stringToReverse = URLEncoder.encode(args[0]); These lines ensure that the user provides one and only one command line argument to the program and encodes it. The command line argument is the string to be reversed by the cgi-bin script backwards. The command line argument may have spaces or other non-alphanumeric characters in it. Those characters must be encoded because various processing may happen on the string on its way to the server. This is achieved by the

Teste a API também no Postman.

Postman

File Edit View Help

New Import Runner My Workspace Invite

Filter

History Collections APIs BETA

Save Responses Clear all

Today

- GET http://127.0.0.1:5000/paginaHTML
- GET http://127.0.0.1:5000/paginaHTML
- GET http://127.0.0.1:5000/paginaTXT
- POST http://127.0.0.1:5000/paginaTXT
- POST http://127.0.0.1:5000/paginaTXT
- POST http://127.0.0.1:5000/coleta
- POST http://127.0.0.1:5000/coleta04
- GET http://127.0.0.1:5000/coleta04
- POST http://127.0.0.1:5000/coleta03
- GET http://127.0.0.1:5000/coleta02
- GET http://127.0.0.1:5000/coleta01
- GET http://127.0.0.1:5000/coleta02
- GET http://127.0.0.1:5000/coleta01

GET http://127.0.0.1:5000/paginaHTML

Params Authorization Headers (9) Body Pre-request Script Tests Cookies Code Comments (0)

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (4) Test Results Status: 200 OK Time: 514 ms Size: 11.38 KB Download

Pretty Raw Preview HTML

```
1
2
3 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
4 <!--NewPage-->
5 <html>
6 <head>
7 <title>Reading from and Writing to a URLConnection</title>
8 </head>
9 <body>
10 <table width="100%">
11 <tr>
12 <td align="left">
```





Centro Universitário UNA  
Sistemas de Informação  
Recuperação de Informação  
Prática de Laboratório  
Wesley Dias Maciel  
2019/01

Postman

File Edit View Help

New Import Runner

My Workspace Invite

No Environment

Filter

History Collections APIs BETA

Save Responses Clear all

Today

- GET http://127.0.0.1:5000/paginaTXT
- GET http://127.0.0.1:5000/paginaHTML
- GET http://127.0.0.1:5000/paginaTXT
- GET http://127.0.0.1:5000/paginaTXT
- POST http://127.0.0.1:5000/paginaTXT
- POST http://127.0.0.1:5000/paginaTXT
- POST http://127.0.0.1:5000/coleta
- POST http://127.0.0.1:5000/coleta04
- GET http://127.0.0.1:5000/coleta04
- POST http://127.0.0.1:5000/coleta03
- GET http://127.0.0.1:5000/coleta02
- GET http://127.0.0.1:5000/coleta01
- GET http://127.0.0.1:5000/coleta02
- GET http://127.0.0.1:5000/coleta01

GET http://127.0.0.1:5000/paginaTXT

Params Authorization Headers (9) Body Pre-request Script Tests

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (4) Test Results

Status: 200 OK Time: 532 ms Size: 8.44 KB Download

Pretty Raw Preview HTML

```
1
2
3
4
5 Reading from and Writing to a URLConnection
6
7
8
9
10
11
12
```

Bootcamp Build Browse